WHERE ……………… > ……….. AND ……….. < ……….. →

```
SELECT *
FROM actor
WHERE first_name = 'Penelope' AND last name = 'Monroe' ;
```

```
WHERE first_name = 'Penelope' OR first_name = 'Bob' ;
```

WHERE NOT → CLAUSE

```
WHERE NOT (rental_rate = 4.99 OR rental_rate = 2.99)
```

Homework-1

1- **Sort the data in the title and description columns in the first film table.**

   SELECT title, description FROM film;

2- **Sort the data in all columns in the movie table with the film length greater than 60 AND less than 75.**

   SELECT * FROM film
   WHERE length >60 and length < 75;

3- **Sort the data in all columns in the film table with rental_rate 0.99 AND replacement_cost 12.99 OR 28.99.**

   SELECT * from film
   WHERE rental_rate = 0.99
   AND replacement_cost = 28.99;

4- **What is the value in the last_name column of the customer whose value is 'Mary' in the first_name column of the customer table?**

   SELECT first_name, last_name FROM customer
   WHERE first_name = 'Mary';

5- **Sort the data in the movie table whose length is NOT greater than 50, but whose rental_rate is NOT 2.99 or 4.99.**

   SELECT * FROM film
   WHERE NOT (length<50)
   AND NOT (rental_rate = 2.99 OR rental_rate = 4.99);

BETWEEN AND SYNTAX

```
SELECT <column_name>, <column_name>, ...
FROM <table_name>
WHERE <condition>;
--------------------------------------------------------------------------------
SELECT *
FROM film
WHERE length BETWEEN 100 AND 140;
-------------------------------------------------------------------------------

-- WHERE length > 100 AND length < 140
```

IN SYNTAX

```
SELECT *
FROM film
WHERE length IN (30,60,90,120);
```

We can also use the NOT IN construct for values out of the list.

Homework-2

1- **Sort all column data in the film table provided that the replacement cost value is greater than 12.99, equal and less than 16.99 (Use BETWEEN - AND structure.)**

   SELECT * FROM film
   WHERE replacement_cost BETWEEN 12.98 AND 16.98;
   --12.99 and 16.99 included

2- **Sort the data in the first_name and last_name columns in the actor table provided that first_name is the values 'Penelope' or 'Nick' or 'Ed'. (Use the IN operator.)**

   SELECT first_name, last_name FROM actor
   WHERE first_name IN ('Penelope', 'Nick', 'Ed');

3- **Sort the data in all columns in the film table with rental_rate 0.99, 2.99, 4.99 AND replacement_cost 12.99, 15.99, 28.99. (Use the IN operator.)**

   SELECT * FROM film
   WHERE rental_rate IN (0.99, 2.99, 4.99)
   AND replacement_cost IN (12.99, 15.99, 28.99);

LIKE / NOT LIKE

For multi character use '%' but for single character use '_' symbol

```
SELECT *
FROM actor
WHERE first_name LIKE 'P%';
```

```
SELECT *
FROM actor
WHERE first_name -- 'P%';

Both uses are same ☺

--* ➔ ILIKE
--  ➔ LIKE
!-- ➔ NOT LIKE
!--* ➔ NOT ILIKE

NOTE : The ILIKE operator is the case - insensitive version of the LIKE operator!
```

Homework-3

1- **List the country names in the country column of the country table, starting with the 'A' character and ending with the 'a' character.**

SELECT * FROM country
WHERE country ILIKE 'A%a';

2- **List the country names in the country column of the country table, consisting of at least 6 characters and ending with the 'n' character.**

SELECT country FROM country
WHERE country ILIKE '_____%n';

3- **In the title column of the film table, list the movie names containing at least 4 'T' characters, regardless of upper- or lower-case letters.**

SELECT title FROM film
WHERE title ILIKE '%T%T%T%T%';

4- **From the data in all the columns in the film table, sort the data that starts with the title 'C' character, has a length greater than 90 and a rental_rate of 2.99.**

SELECT title, length, rental_rate FROM film
WHERE title LIKE 'C%' AND
length > 90 AND rental_rate = 2.99;

SELECT DISTINCT SYNTAX

```
SELECT DISTINCT <columnName>, <columnName>, ...
FROM <tableName>;
```

SELECT COUNT SYNTAX

```
SELECT COUNT(*)
FROM actor
WHERE first_name = 'Penelope';
```

MORE

```
SELECT COUNT(DISTINCT <columnName>)
FROM actor
```

Homework-4

1- **Sort the different values in the replacement cost column in the film table.**

```
SELECT DISTINCT replacement_cost FROM film;
```

2- **How many different data are there in the replacement cost column in the film table?**

```
SELECT COUNT(DISTINCT replacement_cost) FROM film;
```

3- **How many of the film titles in the film table start with the character T and at the same time the rating is equal to 'G'?**

```
SELECT COUNT(title) FROM film
WHERE title LIKE 'T%' AND
rating = 'G';
```

4- **How many of the country names (country) in the country table consist of 5 characters?**

```
SELECT COUNT(country) FROM country

WHERE country LIKE '_____';
```

5- **How many of the city names in the city table end with the character 'R' or r?**

```
SELECT COUNT(city) FROM city

WHERE city ILIKE '%r';
```

ORDER BY SYNTAX

```
SELECT <columnName>, <columnName>, ...
FROM <tableName>
ORDER BY <columnName>, <columnName>, ... ASC|DESC;
```

ASC ➔ INCREASING

DESC ➔ DECREASING

```
SELECT *
FROM film
WHERE title LIKE 'A%'
ORDER BY title ASC length DESC;
```

LIMIT

```
SELECT *
FROM film
WHERE title LIKE 'B%'
ORDER BY length DESC
LIMIT 10;
```

➔ Gives the 10 longest films.

OFFSET

```
SELECT *
FROM film
WHERE title LIKE 'B%'
ORDER BY length DESC
OFFSET 6
LIMIT 4;
```

➔ Skips the 6 longest film and gives other 4 film.

Homework-5

1- **List the 5 longest (length) films in the film table and the film title (title) ends with the 'n' character.**

SELECT * FROM film
WHERE title LIKE '%n'
ORDER BY length DESC
LIMIT 5;

2- **List the shortest (length) second (6,7,8,9,10) 5 films (6,7,8,9,10) in the film table and the film title ends with the 'n' character.**

SELECT * FROM film
WHERE title LIKE '%n'
ORDER BY length DESC
OFFSET 1
LIMIT 5;

3- **Sort the first 4 data, provided that store_id is 1 in the descending order according to the last_name column in the customer table.**

```
SELECT * from customer
WHERE store_id = 1
ORDER BY last_name DESC
LIMIT 4;
```

Aggregate Functions - MIN, MAX, SUM, AVG

```
SELECT AVG(length)
FROM film;
```

Homework-6

1- **What is the average of the values in the rental_rate column in the film table?**

```
SELECT AVG(rental_rate) FROM film;
```

2- **How many of the movies in the film table start with the character 'C'?**

```
SELECT COUNT(title) FROM film
WHERE title LIKE 'C%';
```

3- **Among the movies in the film table, how many minutes is the longest (length) film with a rental_rate equal to 0.99?**

```
SELECT MAX(length) FROM film
WHERE rental_rate = 0.99;
```

4- **How many different replacement_cost values are there for the films longer than 150 minutes in the film table?**

```
SELECT COUNT(replacement_cost) FROM film
WHERE length > 150 ;
```

GROUP BY

```
SELECT rental_rate, MAX(length)
FROM film
GROUP BY rental_rate;
```

HAVING

```
SELECT rental_rate, COUNT(*)
FROM film
GROUP BY rental_rate
HAVING COUNT(*) > 325;
```

Homework-7

1- Group the films in the film table according to their rating values.

```
SELECT rating FROM film
GROUP BY rating;
```

2- When we group the films in the film table according to the replacement_cost column, list the replacement_cost value with more than 50 films and the corresponding number of films.

```
SELECT replacement_cost, COUNT(*) FROM film
GROUP BY replacement_cost
HAVING COUNT(*) > 50;
```

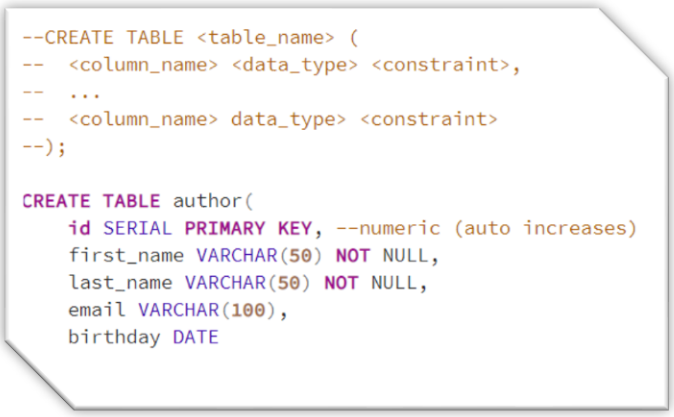
3- What are the customer numbers corresponding to the store_id values in the customer table?

```
SELECT store_id, COUNT(*) FROM customer
GROUP BY store_id;
```

4- After grouping the city data in the city table according to the country_id column, share the country_id information with the highest number of cities and the number of cities.

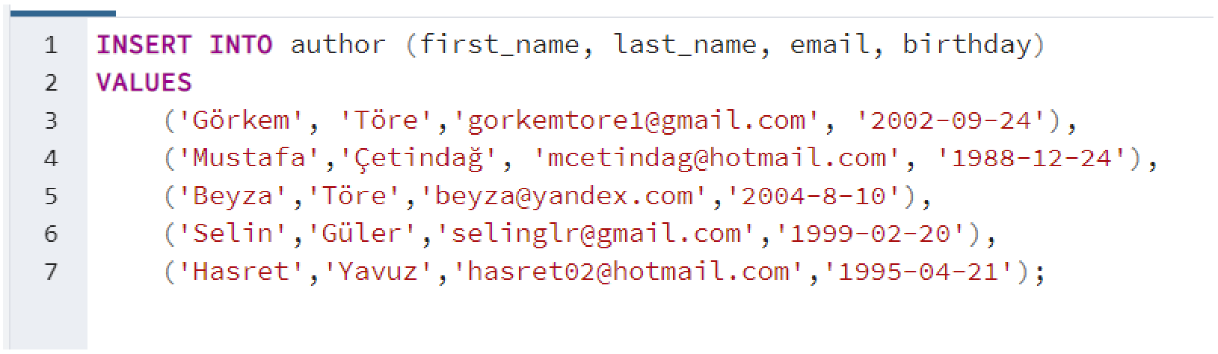
```
SELECT country_id, COUNT(*) FROM city
GROUP BY country_id
ORDER BY COUNT(*) DESC
LIMIT 1; --maximum city
```

CREATING TABLE



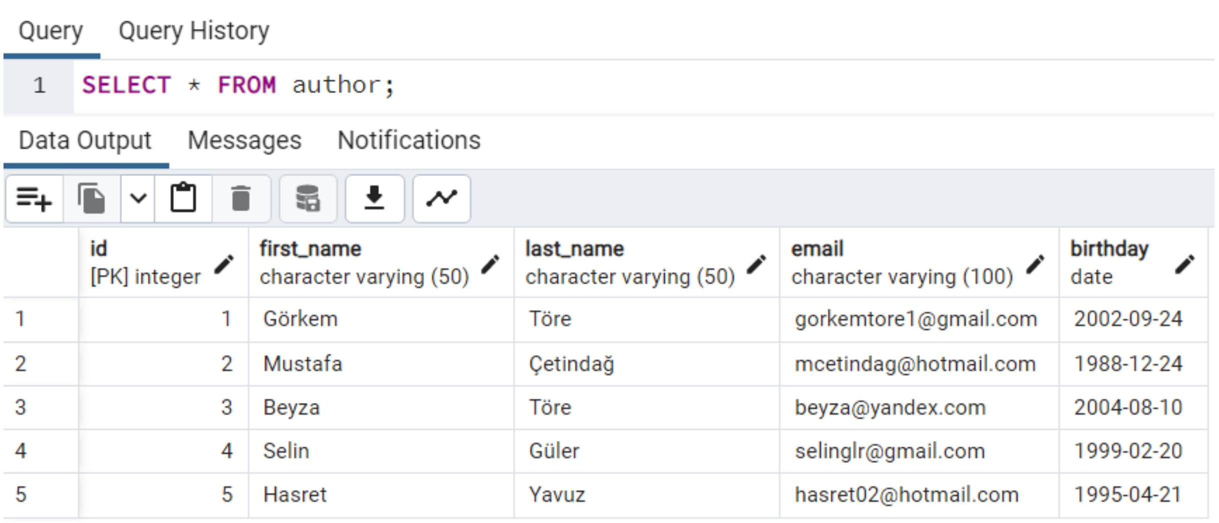
```
--CREATE TABLE <table_name> (
--  <column_name> <data_type> <constraint>,
--  ...
--  <column_name> data_type> <constraint>
--);

CREATE TABLE author(
    id SERIAL PRIMARY KEY, --numeric (auto increases)
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100),
    birthday DATE
```

INSERT USES:



```sql
INSERT INTO author (first_name, last_name, email, birthday)
VALUES
    ('Görkem', 'Töre','gorkemtore1@gmail.com', '2002-09-24'),
    ('Mustafa','Çetindağ', 'mcetindag@hotmail.com', '1988-12-24'),
    ('Beyza','Töre','beyza@yandex.com','2004-8-10'),
    ('Selin','Güler','selinglr@gmail.com','1999-02-20'),
    ('Hasret','Yavuz','hasret02@hotmail.com','1995-04-21');
```

OUTPUT :



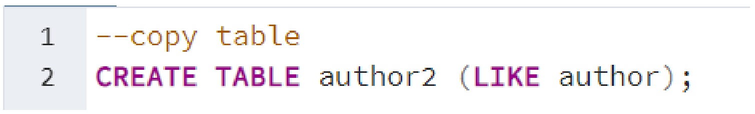
Query   Query History

```sql
SELECT * FROM author;
```

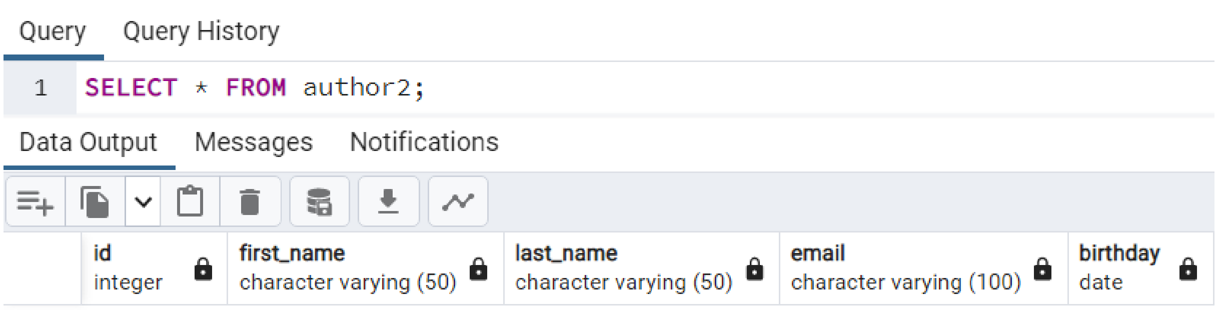
Data Output   Messages   Notifications

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	birthday date
1	1	Görkem	Töre	gorkemtore1@gmail.com	2002-09-24
2	2	Mustafa	Çetindağ	mcetindag@hotmail.com	1988-12-24
3	3	Beyza	Töre	beyza@yandex.com	2004-08-10
4	4	Selin	Güler	selinglr@gmail.com	1999-02-20
5	5	Hasret	Yavuz	hasret02@hotmail.com	1995-04-21

COPYING TABLE SCHEMA :



```sql
--copy table
CREATE TABLE author2 (LIKE author);
```



Query   Query History

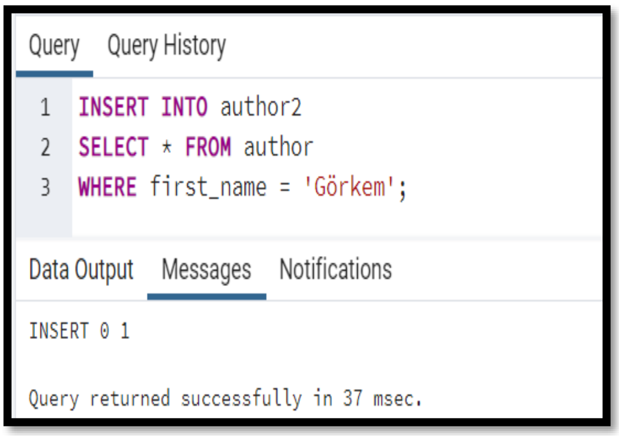
```sql
SELECT * FROM author2;
```

Data Output   Messages   Notifications

	id integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	birthday date

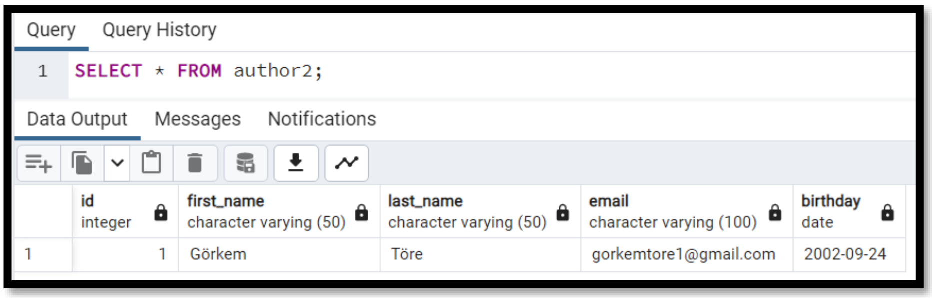
- ***Copied schema but author2 table has not any data!***

INSERTING author TO author2 :



```sql
INSERT INTO author2
SELECT * FROM author
WHERE first_name = 'Görkem';
```

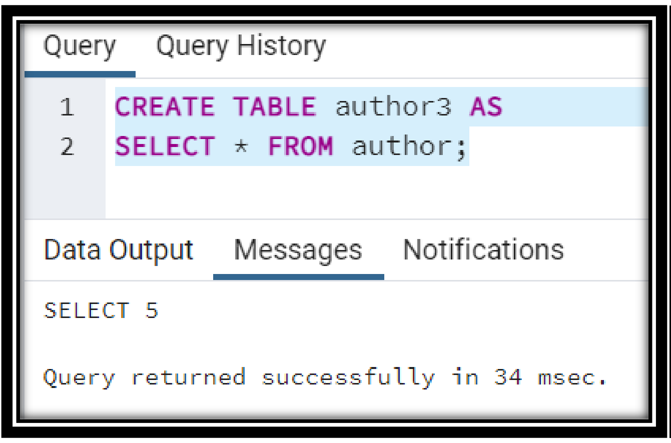
```
INSERT 0 1

Query returned successfully in 37 msec.
```



```sql
SELECT * FROM author2;
```

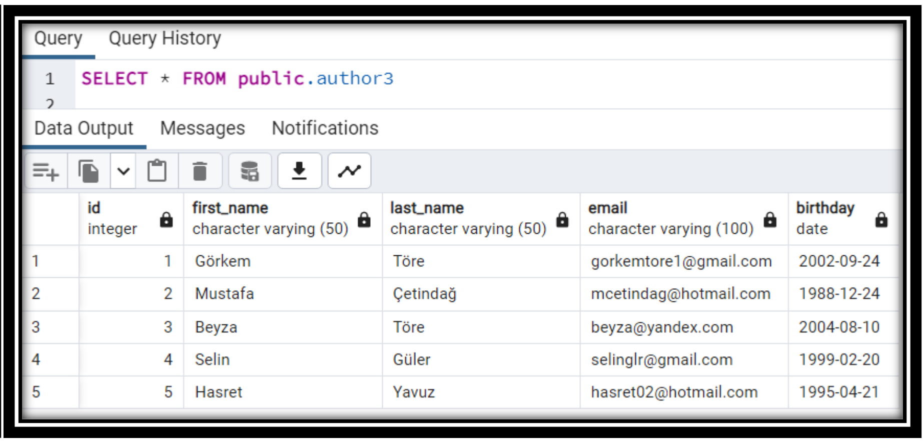
	id integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	birthday date
1	1	Görkem	Töre	gorkemtore1@gmail.com	2002-09-24

COPYING TABLE WITH DATAS:



```sql
CREATE TABLE author3 AS
SELECT * FROM author;
```

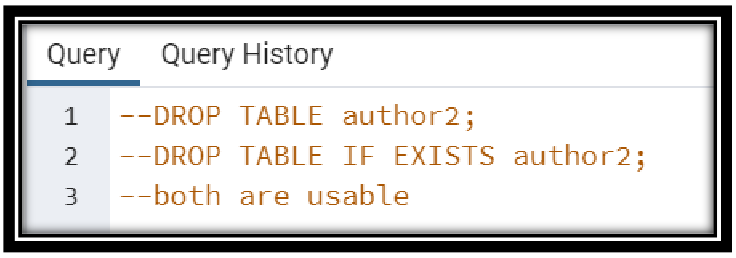
```
SELECT 5

Query returned successfully in 34 msec.
```



```sql
SELECT * FROM public.author3
```

	id integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	birthday date
1	1	Görkem	Töre	gorkemtore1@gmail.com	2002-09-24
2	2	Mustafa	Çetindağ	mcetindag@hotmail.com	1988-12-24
3	3	Beyza	Töre	beyza@yandex.com	2004-08-10
4	4	Selin	Güler	selinglr@gmail.com	1999-02-20
5	5	Hasret	Yavuz	hasret02@hotmail.com	1995-04-21

DROP TABLE:



```sql
--DROP TABLE author2;
--DROP TABLE IF EXISTS author2;
--both are usable
```

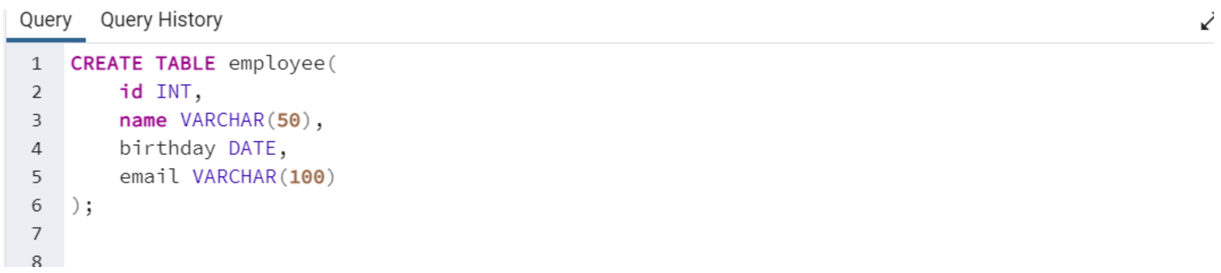
UPDATE SYNTAX

```sql
UPDATE <tableName>
SET <columnName> = value,
    <columnName> = value,
    ----
WHERE <condition>;
```

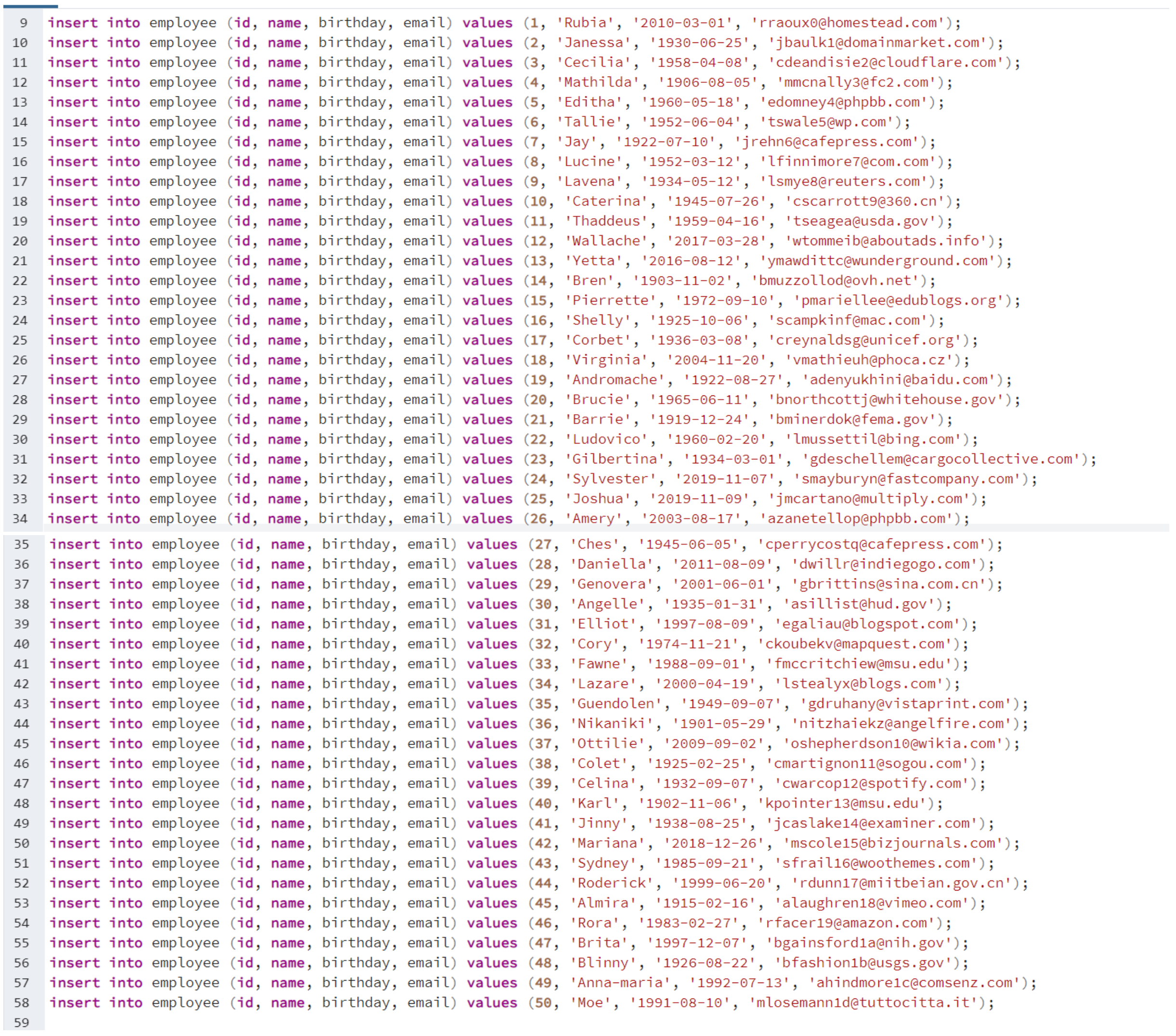
DELETE SYNTAX

```sql
DELETE FROM <tablo_adı>
WHERE <koşul_adı>;
```

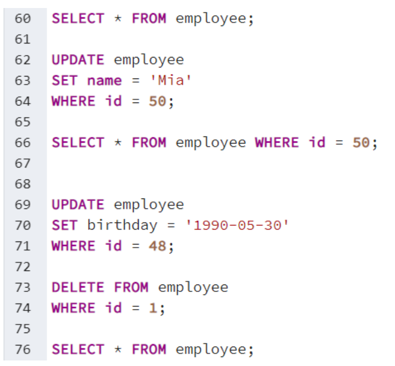
Homework-8



```sql
CREATE TABLE employee(
    id INT,
    name VARCHAR(50),
    birthday DATE,
    email VARCHAR(100)
);
```

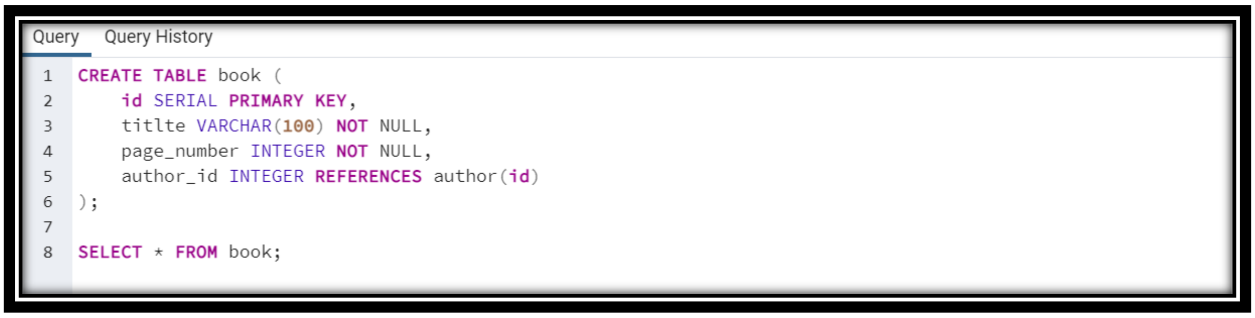


```sql
insert into employee (id, name, birthday, email) values (1, 'Rubia', '2010-03-01', 'rraoux0@homestead.com');
insert into employee (id, name, birthday, email) values (2, 'Janessa', '1930-06-25', 'jbaulk1@domainmarket.com');
insert into employee (id, name, birthday, email) values (3, 'Cecilia', '1958-04-08', 'cdeandisie2@cloudflare.com');
insert into employee (id, name, birthday, email) values (4, 'Mathilda', '1906-08-05', 'mmcnally3@fc2.com');
insert into employee (id, name, birthday, email) values (5, 'Editha', '1960-05-18', 'edomney4@phpbb.com');
insert into employee (id, name, birthday, email) values (6, 'Tallie', '1952-06-04', 'tswale5@wp.com');
insert into employee (id, name, birthday, email) values (7, 'Jay', '1922-07-10', 'jrehn6@cafepress.com');
insert into employee (id, name, birthday, email) values (8, 'Lucine', '1952-03-12', 'lfinnimore7@com.com');
insert into employee (id, name, birthday, email) values (9, 'Lavena', '1934-05-12', 'lsmye8@reuters.com');
insert into employee (id, name, birthday, email) values (10, 'Caterina', '1945-07-26', 'cscarrott9@360.cn');
insert into employee (id, name, birthday, email) values (11, 'Thaddeus', '1959-04-16', 'tseagea@usda.gov');
insert into employee (id, name, birthday, email) values (12, 'Wallache', '2017-03-28', 'wtommeib@aboutads.info');
insert into employee (id, name, birthday, email) values (13, 'Yetta', '2016-08-12', 'ymawdittc@wunderground.com');
insert into employee (id, name, birthday, email) values (14, 'Bren', '1903-11-02', 'bmuzzollod@ovh.net');
insert into employee (id, name, birthday, email) values (15, 'Pierrette', '1972-09-10', 'pmariellee@edublogs.org');
insert into employee (id, name, birthday, email) values (16, 'Shelly', '1925-10-06', 'scampkinf@mac.com');
insert into employee (id, name, birthday, email) values (17, 'Corbet', '1936-03-08', 'creynaldsg@unicef.org');
insert into employee (id, name, birthday, email) values (18, 'Virginia', '2004-11-20', 'vmathieuh@phoca.cz');
insert into employee (id, name, birthday, email) values (19, 'Andromache', '1922-08-27', 'adenyukhini@baidu.com');
insert into employee (id, name, birthday, email) values (20, 'Brucie', '1965-06-11', 'bnorthcottj@whitehouse.gov');
insert into employee (id, name, birthday, email) values (21, 'Barrie', '1919-12-24', 'bminerdok@fema.gov');
insert into employee (id, name, birthday, email) values (22, 'Ludovico', '1960-02-20', 'lmussettil@bing.com');
insert into employee (id, name, birthday, email) values (23, 'Gilbertina', '1934-03-01', 'gdeschellem@cargocollective.com');
insert into employee (id, name, birthday, email) values (24, 'Sylvester', '2019-11-07', 'smayburyn@fastcompany.com');
insert into employee (id, name, birthday, email) values (25, 'Joshua', '2019-11-09', 'jmcartano@multiply.com');
insert into employee (id, name, birthday, email) values (26, 'Amery', '2003-08-17', 'azanetellop@phpbb.com');
insert into employee (id, name, birthday, email) values (27, 'Ches', '1945-06-05', 'cperrycostq@cafepress.com');
insert into employee (id, name, birthday, email) values (28, 'Daniella', '2011-08-09', 'dwillr@indiegogo.com');
insert into employee (id, name, birthday, email) values (29, 'Genovera', '2001-06-01', 'gbrittins@sina.com.cn');
insert into employee (id, name, birthday, email) values (30, 'Angelle', '1935-01-31', 'asillist@hud.gov');
insert into employee (id, name, birthday, email) values (31, 'Elliot', '1997-08-09', 'egaliau@blogspot.com');
insert into employee (id, name, birthday, email) values (32, 'Cory', '1974-11-21', 'ckoubekv@mapquest.com');
insert into employee (id, name, birthday, email) values (33, 'Fawne', '1988-09-01', 'fmccritchiew@msu.edu');
insert into employee (id, name, birthday, email) values (34, 'Lazare', '2000-04-19', 'lstealyx@blogs.com');
insert into employee (id, name, birthday, email) values (35, 'Guendolen', '1949-09-07', 'gdruhany@vistaprint.com');
insert into employee (id, name, birthday, email) values (36, 'Nikaniki', '1901-05-29', 'nitzhaiekz@angelfire.com');
insert into employee (id, name, birthday, email) values (37, 'Ottilie', '2009-09-02', 'oshepherdson10@wikia.com');
insert into employee (id, name, birthday, email) values (38, 'Colet', '1925-02-25', 'cmartignon11@sogou.com');
insert into employee (id, name, birthday, email) values (39, 'Celina', '1932-09-07', 'cwarcop12@spotify.com');
insert into employee (id, name, birthday, email) values (40, 'Karl', '1902-11-06', 'kpointer13@msu.edu');
insert into employee (id, name, birthday, email) values (41, 'Jinny', '1938-08-25', 'jcaslake14@examiner.com');
insert into employee (id, name, birthday, email) values (42, 'Mariana', '2018-12-26', 'mscole15@bizjournals.com');
insert into employee (id, name, birthday, email) values (43, 'Sydney', '1985-09-21', 'sfrail16@woothemes.com');
insert into employee (id, name, birthday, email) values (44, 'Roderick', '1999-06-20', 'rdunn17@miitbeian.gov.cn');
insert into employee (id, name, birthday, email) values (45, 'Almira', '1915-02-16', 'alaughren18@vimeo.com');
insert into employee (id, name, birthday, email) values (46, 'Rora', '1983-02-27', 'rfacer19@amazon.com');
insert into employee (id, name, birthday, email) values (47, 'Brita', '1997-12-07', 'bgainsford1a@nih.gov');
insert into employee (id, name, birthday, email) values (48, 'Blinny', '1926-08-22', 'bfashion1b@usgs.gov');
insert into employee (id, name, birthday, email) values (49, 'Anna-maria', '1992-07-13', 'ahindmore1c@comsenz.com');
insert into employee (id, name, birthday, email) values (50, 'Moe', '1991-08-10', 'mlosemann1d@tuttocitta.it');
```



```sql
SELECT * FROM employee;

UPDATE employee
SET name = 'Mia'
WHERE id = 50;

SELECT * FROM employee WHERE id = 50;


UPDATE employee
SET birthday = '1990-05-30'
WHERE id = 48;

DELETE FROM employee
WHERE id = 1;

SELECT * FROM employee;
```

PRIMARY KEY – FOREIGN KEY



```
CREATE TABLE book (
    id SERIAL PRIMARY KEY,
    titlte VARCHAR(100) NOT NULL,
    page_number INTEGER NOT NULL,
    author_id INTEGER REFERENCES author(id)
);

SELECT * FROM book;
```

ALTER

The ALTER keyword is used to modify an existing table.

```
ALTER TABLE <table_name>
ALTER COLUMN <column_name>
SET --NOT NULL–(constraint);
```

UNIQUE

```
CREATE TABLE Employees (
    ---
    email VARCHAR(100) UNIQUE,
    ----
);
```

ALTER and UNIQUE

```
ALTER TABLE <table_name>
ADD UNIQUE <column_name>
```

CHECK

```
CREATE TABLE Employees (
    ---
    age INTEGER CHECK (age>=18)
    ----
);
```

ALTER and CHECK

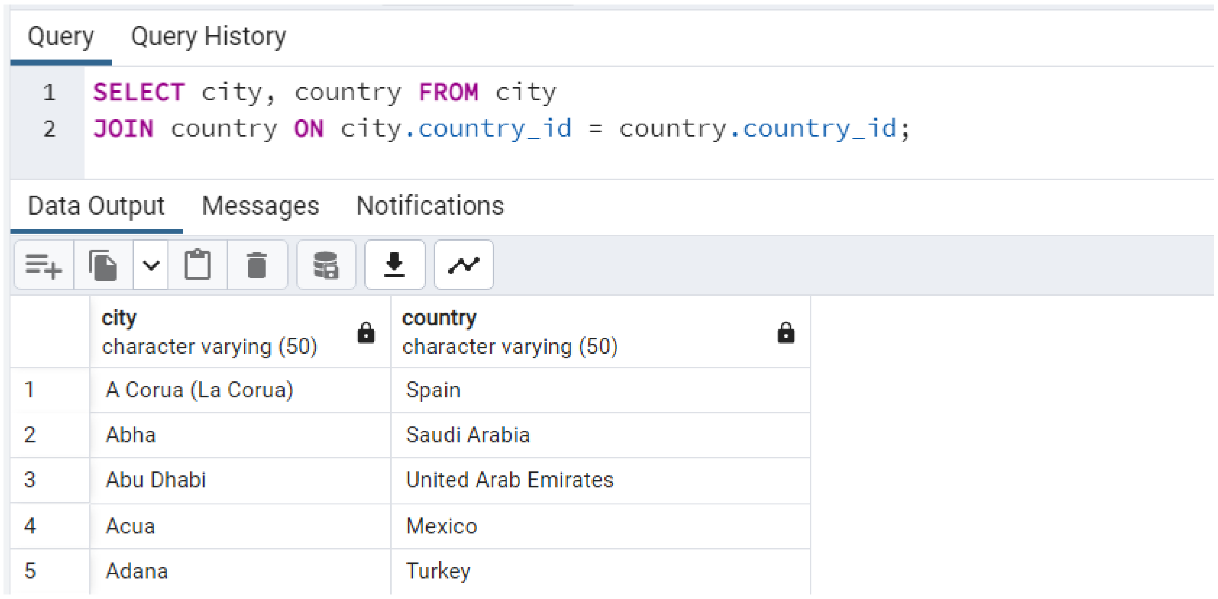
```
ALTER TABLE <table_name>
ADD CHECK (age>=18)
```

INNER JOIN = JOIN

```sql
SELECT book.title, author.first_name, author.last_name
FROM book
JOIN author ON author.id = book.author_id;
```

Homework-9

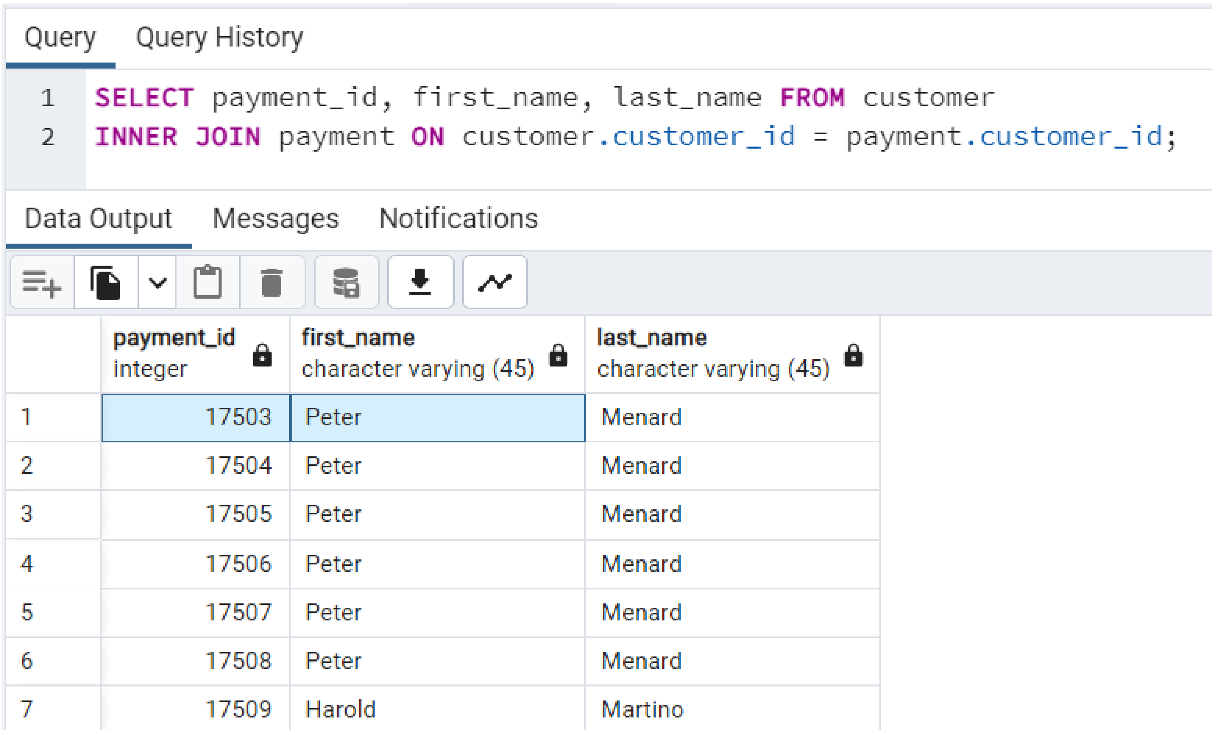
1- **Write the INNER JOIN query where we can see the city and country names in the city table and the country table together.**



```sql
SELECT city, country FROM city
JOIN country ON city.country_id = country.country_id;
```

	city character varying (50)	country character varying (50)
1	A Corua (La Corua)	Spain
2	Abha	Saudi Arabia
3	Abu Dhabi	United Arab Emirates
4	Acua	Mexico
5	Adana	Turkey

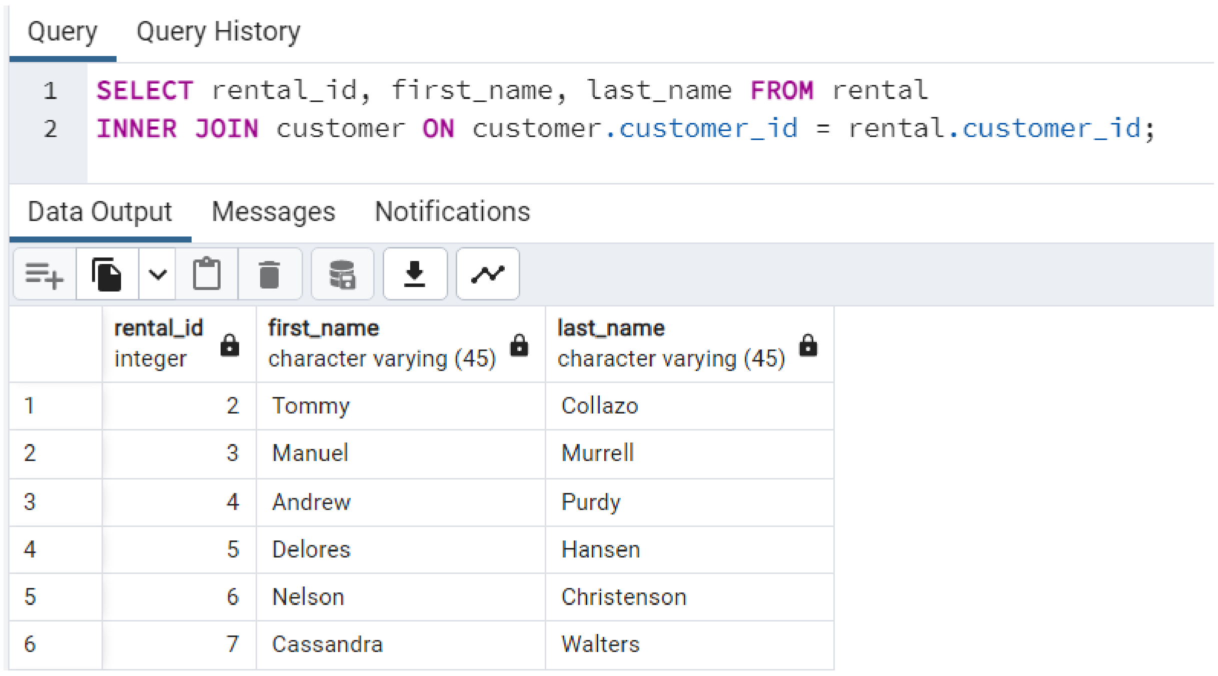
2- **Write the INNER JOIN query where we can see the customer table and the payment_id in the payment table and the first_name and last_name names in the customer table together.**



```sql
SELECT payment_id, first_name, last_name FROM customer
INNER JOIN payment ON customer.customer_id = payment.customer_id;
```

	payment_id integer	first_name character varying (45)	last_name character varying (45)
1	17503	Peter	Menard
2	17504	Peter	Menard
3	17505	Peter	Menard
4	17506	Peter	Menard
5	17507	Peter	Menard
6	17508	Peter	Menard
7	17509	Harold	Martino

3- **Write the INNER JOIN query where we can see the customer table and the rental_id in the rental table and the first_name and last_name names in the customer table together.**

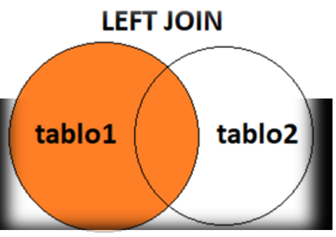


```
SELECT rental_id, first_name, last_name FROM rental
INNER JOIN customer ON customer.customer_id = rental.customer_id;
```

	rental_id integer	first_name character varying (45)	last_name character varying (45)
1	2	Tommy	Collazo
2	3	Manuel	Murrell
3	4	Andrew	Purdy
4	5	Delores	Hansen
5	6	Nelson	Christenson
6	7	Cassandra	Walters

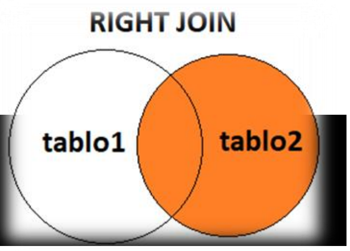
LEFT JOIN

```
SELECT book.title, author.first_name, author.last_name
FROM book
LEFT JOIN author
ON author.id = book.author_id;
```



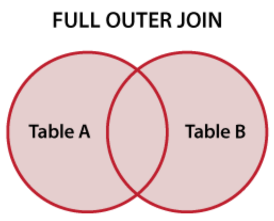
RIGHT JOIN

```
SELECT book.title, author.first_name, author.last_name
FROM book
RIGHT JOIN author
ON author.id = book.author_id;
```

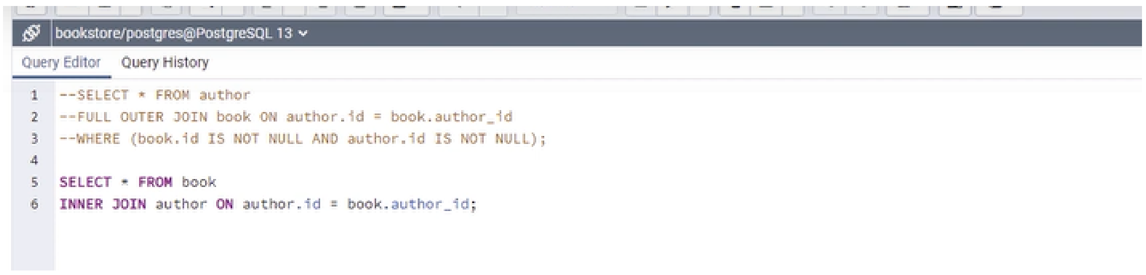


FULL JOIN = FULL OUTER JOIN

```
SELECT book.title, author.first_name, author.last_name
FROM book
FULL JOIN author
ON author.id = book.author_id;
```



We can use, FULL JOIN WITH WHERE CONDITION as INNER JOIN ! 😊



bookstore/postgres@PostgreSQL 13

Query Editor Query History

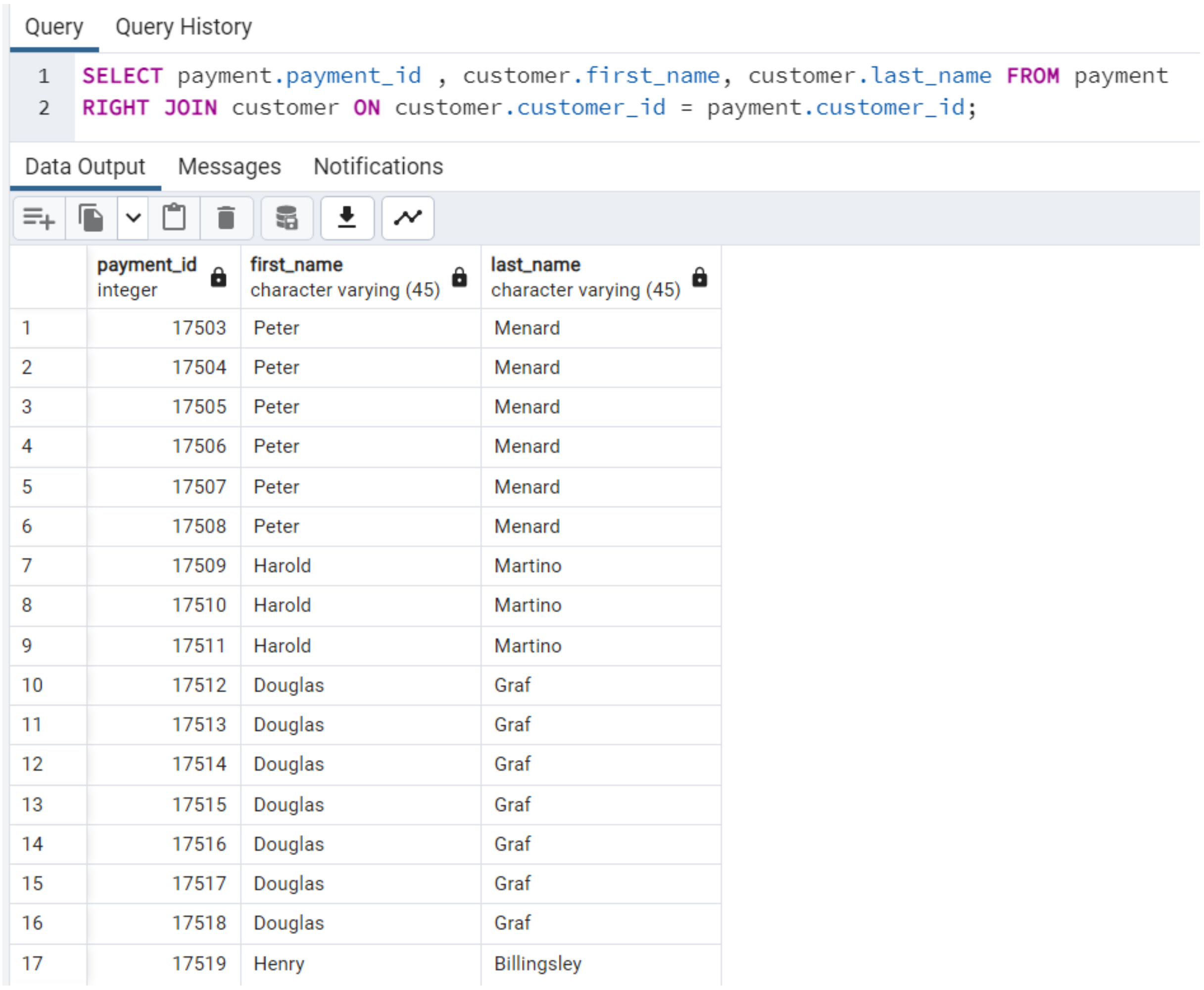
```
--SELECT * FROM author
--FULL OUTER JOIN book ON author.id = book.author_id
--WHERE (book.id IS NOT NULL AND author.id IS NOT NULL);

SELECT * FROM book
INNER JOIN author ON author.id = book.author_id;
```

Homework-10

1- **Write the LEFT JOIN query where we can see the city and country names in the city table and the country table together.**

```
SELECT city.city, country.country FROM city
LEFT JOIN country ON city.country_id = country.country_id;
```

2- **Write the RIGHT JOIN query where we can see the customer table and the payment_id in the payment table and the first_name and last_name names in the customer table together.**



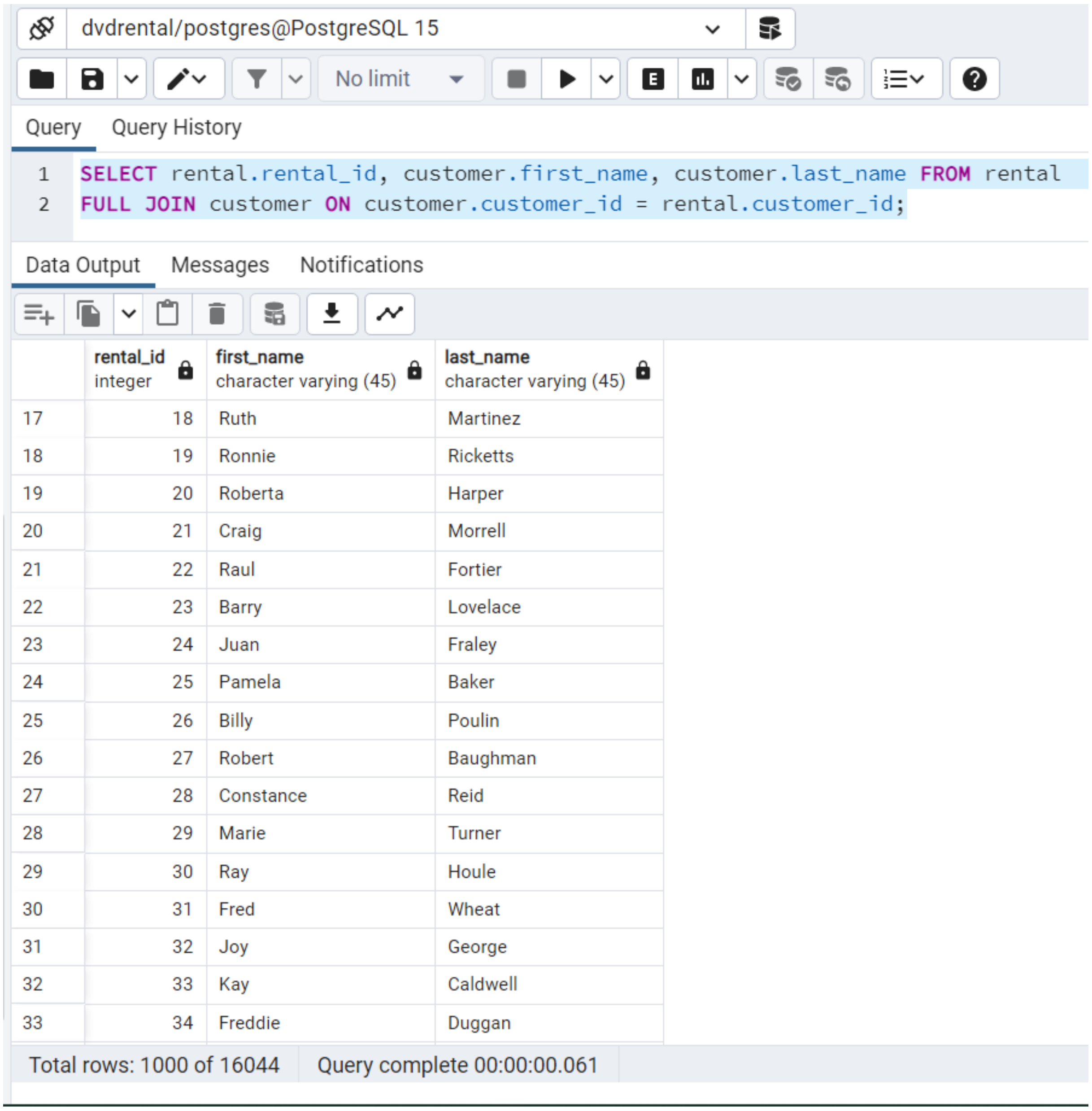
Query Query History

```
SELECT payment.payment_id , customer.first_name, customer.last_name FROM payment
RIGHT JOIN customer ON customer.customer_id = payment.customer_id;
```

Data Output Messages Notifications

	payment_id integer	first_name character varying (45)	last_name character varying (45)
1	17503	Peter	Menard
2	17504	Peter	Menard
3	17505	Peter	Menard
4	17506	Peter	Menard
5	17507	Peter	Menard
6	17508	Peter	Menard
7	17509	Harold	Martino
8	17510	Harold	Martino
9	17511	Harold	Martino
10	17512	Douglas	Graf
11	17513	Douglas	Graf
12	17514	Douglas	Graf
13	17515	Douglas	Graf
14	17516	Douglas	Graf
15	17517	Douglas	Graf
16	17518	Douglas	Graf
17	17519	Henry	Billingsley

3- **Write the FULL JOIN query where we can see the customer table and the rental_id in the rental table and the first_name and last_name names in the customer table together.**



```sql
SELECT rental.rental_id, customer.first_name, customer.last_name FROM rental
FULL JOIN customer ON customer.customer_id = rental.customer_id;
```

	rental_id integer	first_name character varying (45)	last_name character varying (45)
17	18	Ruth	Martinez
18	19	Ronnie	Ricketts
19	20	Roberta	Harper
20	21	Craig	Morrell
21	22	Raul	Fortier
22	23	Barry	Lovelace
23	24	Juan	Fraley
24	25	Pamela	Baker
25	26	Billy	Poulin
26	27	Robert	Baughman
27	28	Constance	Reid
28	29	Marie	Turner
29	30	Ray	Houle
30	31	Fred	Wheat
31	32	Joy	George
32	33	Kay	Caldwell
33	34	Freddie	Duggan

Total rows: 1000 of 16044   Query complete 00:00:00.061

UNION SYNTAX

```
SELECT <sütun_adı>, <sütun_adı>...
FROM <table1>
UNION
SELECT <sütun_adı>, <sütun_adı>...
FROM <table2>
```

NOTE THAT :

UNION, gets unduplicated lines. But if we want to duplicate the same lines, we can use UNION ALL!

INTERSECT(ION)

```
(
SELECT *
FROM book
ORDER BY title
LIMIT 5
)
INTERSECT
(
SELECT *
FROM book
ORDER BY page_number DESC
LIMIT 5
);
```

➔ INTERSECT keyword, gives to us intersection of 2 queries.

EXCEPT

```
(
SELECT *
FROM book
ORDER BY title
LIMIT 5
)
EXCEPT
(
SELECT *
FROM book
ORDER BY page_number DESC
LIMIT 5
);
```

➔ EXCEPT keyword, returns rows that are not in the intersection set.

Homework-11

1- **Let's sort all the data for the first_name columns in the actor and customer tables.**

```
SELECT first_name FROM actor
UNION
SELECT first_name FROM customer;
```

2- **Let's sort the intersecting data for the first_name columns in the actor and customer tables.**

```
SELECT first_name FROM actor
INTERSECT
SELECT first_name FROM customer;
```

3- **For the first_name columns in the actor and customer tables, let's sort the data in the first table but not in the second table.**

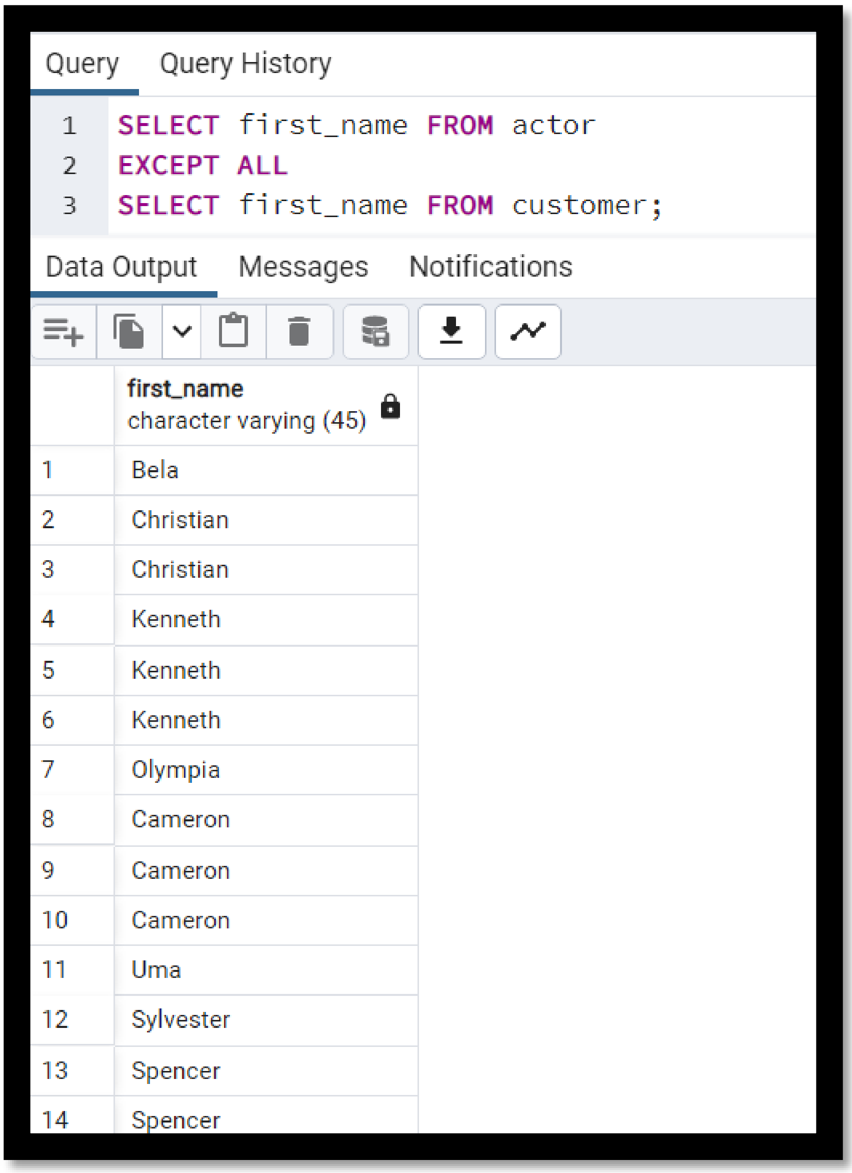
SELECT first_name FROM actor
EXCEPT
SELECT first_name FROM customer;

4- Let's also do the first 3 queries for repeating data.

1- SELECT first_name FROM actor
UNION ALL
SELECT first_name FROM customer;

2- INTERSECT = INTERSECT ALL, so we can use same query.

3-



```sql
SELECT first_name FROM actor
EXCEPT ALL
SELECT first_name FROM customer;
```

	first_name character varying (45)
1	Bela
2	Christian
3	Christian
4	Kenneth
5	Kenneth
6	Kenneth
7	Olympia
8	Cameron
9	Cameron
10	Cameron
11	Uma
12	Sylvester
13	Spencer
14	Spencer

HACKER RANK QUESTION:

Query the *Name* of any student in **STUDENTS** who scored higher than *Marks*. Order your output by the *last three characters* of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending *ID*.

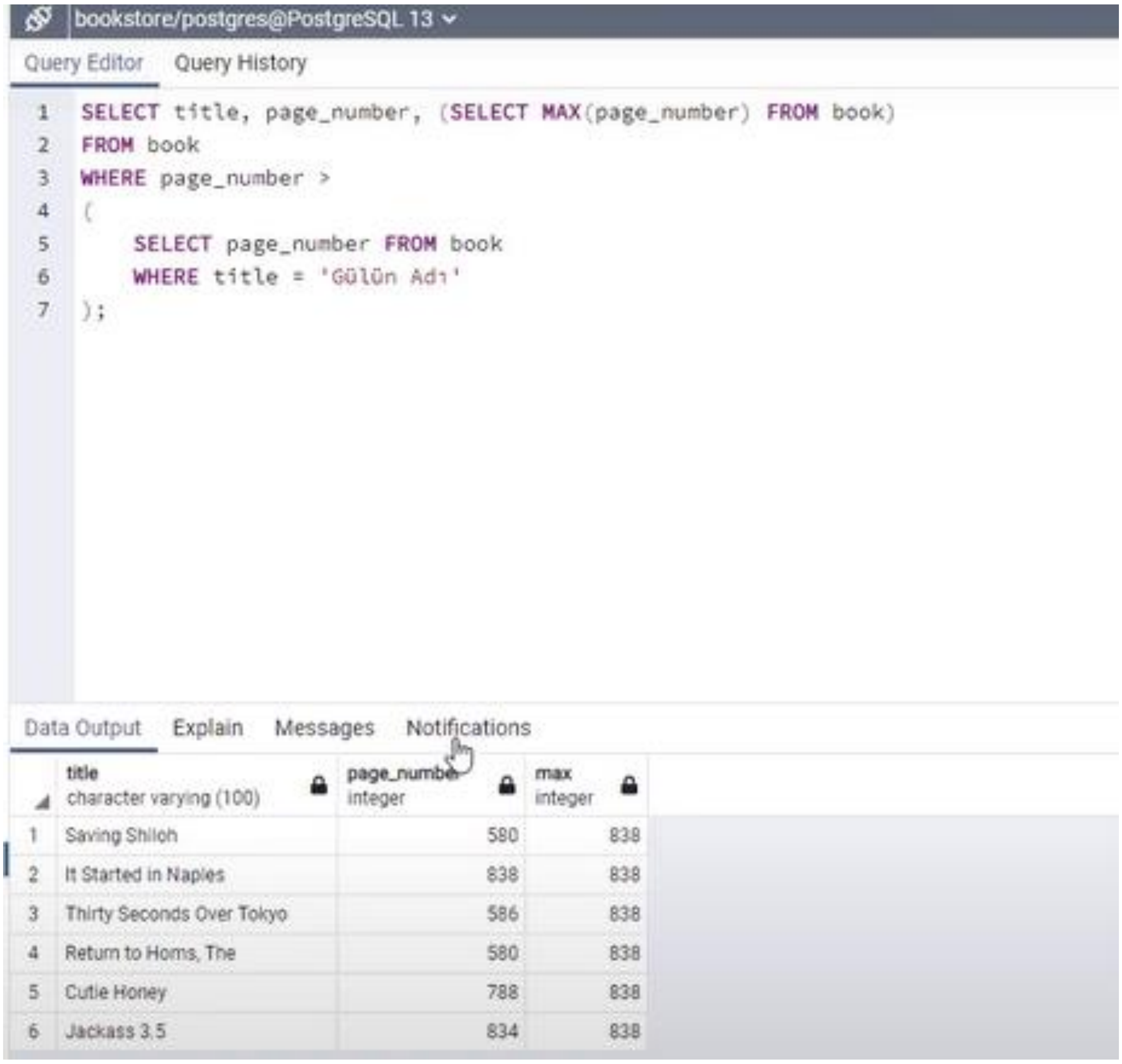
ANSWER:

SELECT name FROM STUDENTS

WHERE Marks > 75

ORDER BY RIGHT(Name,3) , ID;

SUBQUERIES



bookstore/postgres@PostgreSQL 13

Query Editor | Query History

```
SELECT title, page_number, (SELECT MAX(page_number) FROM book)
FROM book
WHERE page_number >
(
    SELECT page_number FROM book
    WHERE title = 'Gülün Adı'
);
```

Data Output | Explain | Messages | Notifications

	title character varying (100)	page_number integer	max integer
1	Saving Shiloh	580	838
2	It Started in Naples	838	838
3	Thirty Seconds Over Tokyo	586	838
4	Return to Homs, The	580	838
5	Cutie Honey	788	838
6	Jackass 3.5	834	838

ANY

```
SELECT first_name, last_name
FROM author
WHERE id = ANY
(
  SELECT id
  FROM book
  WHERE title = 'Abe Lincoln in Illinois' OR title = 'Saving Shiloh'
)
```

Gives **id of** ‘Abe Lincoln in Illinois’ and ‘Saving Shiloh’.

- ⇨ Any condition must be met

ALL

```
SELECT first_name, last_name
FROM author
WHERE id = ALL
(
  SELECT id
  FROM book
  WHERE title = 'Abe Lincoln in Illinois' OR title = 'Saving Shiloh'
)
```

- ⇨ ALL condition must be met

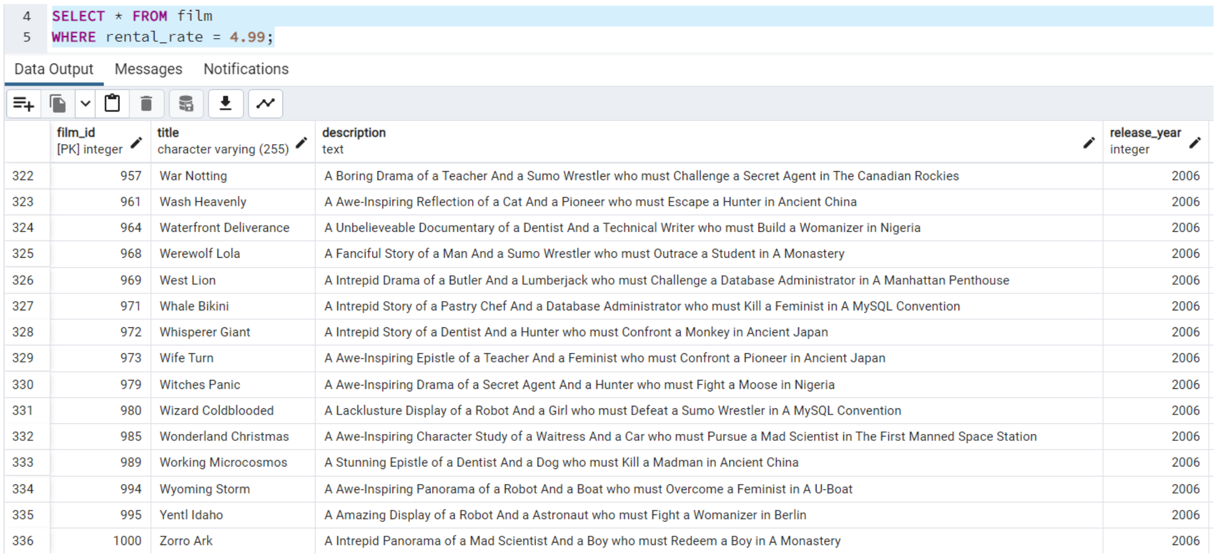
Homework-12

1- **In the film table, the film length is shown in the length column. How many movies are longer than the average film length?**

```
SELECT COUNT(*) FROM film
WHERE length > (SELECT AVG(length) FROM film);
```

2- **How many films have the highest rental_rate in the film table?**

```
SELECT COUNT(*) FROM film
WHERE rental_rate = (SELECT MAX(rental_rate) FROM film);
```



```
SELECT * FROM film
WHERE rental_rate = 4.99;
```

Data Output   Messages   Notifications

	film_id [PK] integer	title character varying (255)	description text	release_year integer
322	957	War Notting	A Boring Drama of a Teacher And a Sumo Wrestler who must Challenge a Secret Agent in The Canadian Rockies	2006
323	961	Wash Heavenly	A Awe-Inspiring Reflection of a Cat And a Pioneer who must Escape a Hunter in Ancient China	2006
324	964	Waterfront Deliverance	A Unbelieveable Documentary of a Dentist And a Technical Writer who must Build a Womanizer in Nigeria	2006
325	968	Werewolf Lola	A Fanciful Story of a Man And a Sumo Wrestler who must Outrace a Student in A Monastery	2006
326	969	West Lion	A Intrepid Drama of a Butler And a Lumberjack who must Challenge a Database Administrator in A Manhattan Penthouse	2006
327	971	Whale Bikini	A Intrepid Story of a Pastry Chef And a Database Administrator who must Kill a Feminist in A MySQL Convention	2006
328	972	Whisperer Giant	A Intrepid Story of a Dentist And a Hunter who must Confront a Monkey in Ancient Japan	2006
329	973	Wife Turn	A Awe-Inspiring Epistle of a Teacher And a Feminist who must Confront a Pioneer in Ancient Japan	2006
330	979	Witches Panic	A Awe-Inspiring Drama of a Secret Agent And a Hunter who must Fight a Moose in Nigeria	2006
331	980	Wizard Coldblooded	A Lacklusture Display of a Robot And a Girl who must Defeat a Sumo Wrestler in A MySQL Convention	2006
332	985	Wonderland Christmas	A Awe-Inspiring Character Study of a Waitress And a Car who must Pursue a Mad Scientist in The First Manned Space Station	2006
333	989	Working Microcosmos	A Stunning Epistle of a Dentist And a Dog who must Kill a Madman in Ancient China	2006
334	994	Wyoming Storm	A Awe-Inspiring Panorama of a Robot And a Boat who must Overcome a Feminist in A U-Boat	2006
335	995	Yentl Idaho	A Amazing Display of a Robot And a Astronaut who must Fight a Womanizer in Berlin	2006
336	1000	Zorro Ark	A Intrepid Panorama of a Mad Scientist And a Boy who must Redeem a Boy in A Monastery	2006

➔ As you can see, there are 336 movies with a rental_rate of 4.99.

3- In the film table, list the films with the lowest rental rate and the lowest replacement cost.

```
SELECT * FROM film
WHERE rental_rate =
(SELECT MIN(rental_rate) FROM film)
AND replacement_cost =
(SELECT MIN(replacement_cost) FROM film);
```

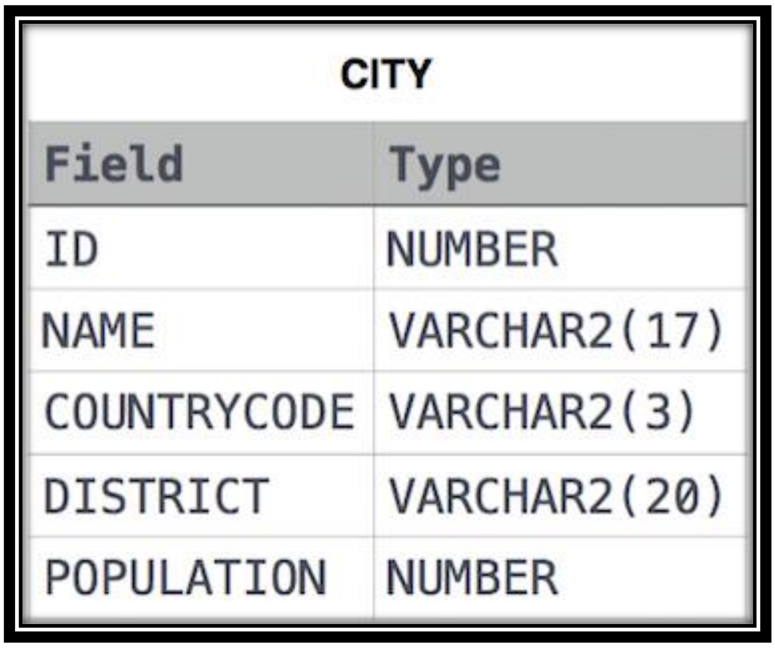
4- In the payment table, list the customers who make the most purchases.

```
SELECT COUNT(payment.customer_id) AS Purchases, customer.first_name,
customer.last_name FROM payment
INNER JOIN customer ON payment.customer_id = customer.customer_id
GROUP BY payment.customer_id, customer.first_name, customer.last_name
ORDER BY COUNT(payment.customer_id) DESC
LIMIT 5
```

HackerRank Question :

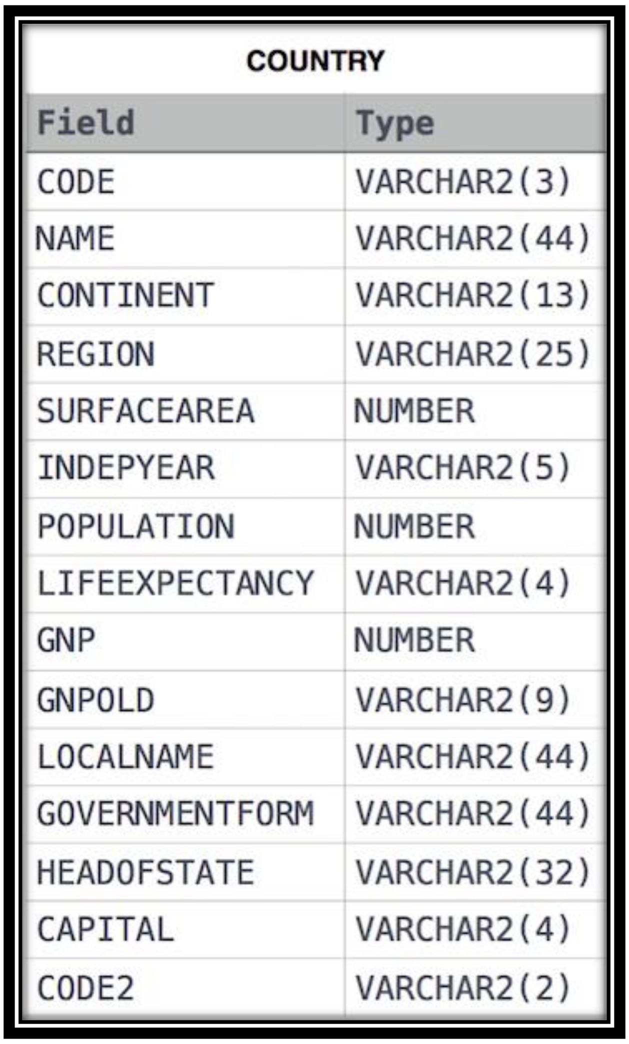
Given the CITY and COUNTRY tables, query the names of all the continents (COUNTRY.Continent) and their respective average city populations (CITY.Population) rounded down to the nearest integer.

Note: CITY.CountryCode and COUNTRY.Code are matching key columns.



CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER



COUNTRY

Field	Type
CODE	VARCHAR2(3)
NAME	VARCHAR2(44)
CONTINENT	VARCHAR2(13)
REGION	VARCHAR2(25)
SURFACEAREA	NUMBER
INDEPYEAR	VARCHAR2(5)
POPULATION	NUMBER
LIFEEXPECTANCY	VARCHAR2(4)
GNP	NUMBER
GNPOLD	VARCHAR2(9)
LOCALNAME	VARCHAR2(44)
GOVERNMENTFORM	VARCHAR2(44)
HEADOFSTATE	VARCHAR2(32)
CAPITAL	VARCHAR2(4)
CODE2	VARCHAR2(2)

Answer :

```
SELECT COUNTRY.CONTINENT,
FLOOR(AVG(CITY.POPULATION)) FROM COUNTRY

INNER JOIN CITY ON CITY.CountryCode =
COUNTRY.Code

GROUP BY COUNTRY.CONTINENT
```

THANK YOU FOR READING