SQL - Assignment

Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA.

SELECT * FROM CITY WHERE population > 100000 and countrycode = 'USA';

Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.

SELECT name FROM CITY WHERE population > 120000 and countrycode = 'USA';

Q3. Query all columns (attributes) for every row in the CITY table.

SELECT * FROM city;

Q4. Query all columns for a city in CITY with the ID 1661.

SELECT * FROM city WHERE ID = 1661;

Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.

SELECT * FROM city WHERE countrycode = 'JPN';

Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is JPN.

SELECT name FROM city WHERE countrycode = 'JPN';

To list the table details: **SELECT * FROM station;**

	ID int	CITY varchar	STATES varchar	LAT_N int	LONG_W int
1	1	Bengaluru	KA	39	52
2	2	Chennai	TN	34	55
3	3	Mumbai	MA	30	10
4	4	Kottayam	KL	54	15
5	5	Arunachal	AP	39	52
6	6	Chennai	TN	19	32

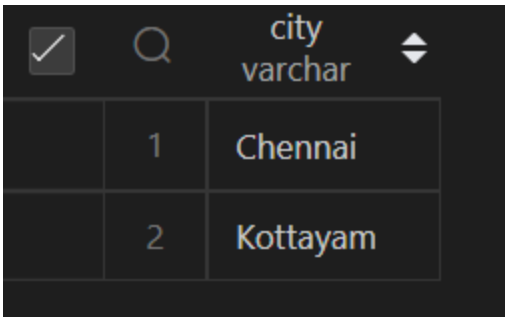
Q7. Query a list of CITY and STATE from the STATION table.

SELECT city,state FROM Station;

	city varchar	states varchar
1	Bengaluru	KA
2	Chennai	TN
3	Mumbai	MA
4	Kottayam	KL
5	Arunachal	AP
6	Chennai	TN

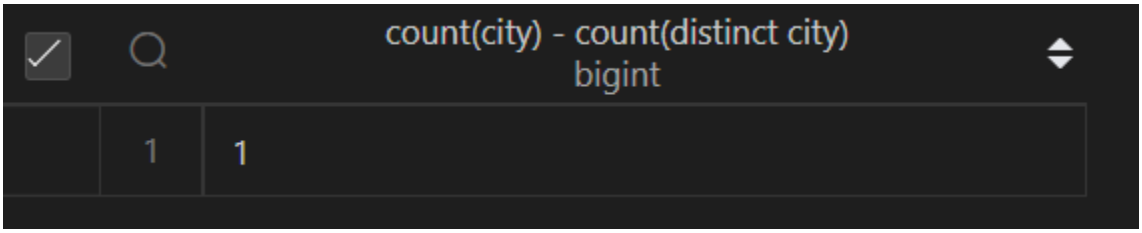
Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order, but exclude duplicates from the answer.

SELECT DISTINCT city FROM Station WHERE mod(ID,2)=0 ORDER BY city ASC;



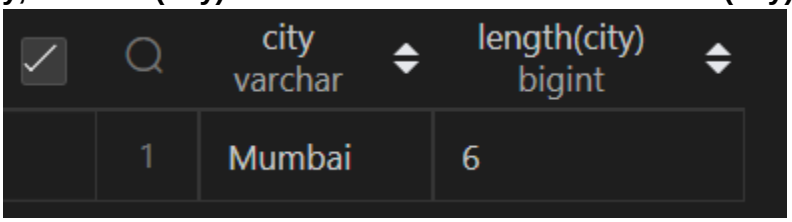
Q9. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table.

SELECT COUNT(city) - COUNT(DISTINCT city) FROM station;

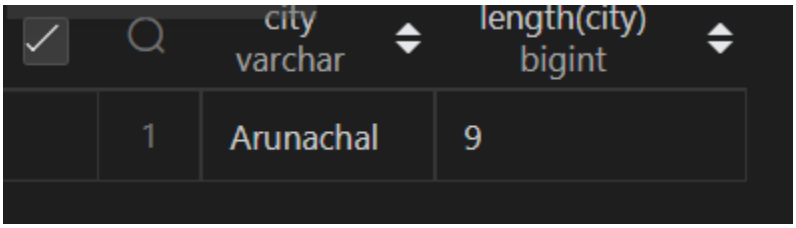


Q10. Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.

SELECT city, LENGTH(city) FROM station ORDER BY LENGTH(city) ASC, city LIMIT 1;



SELECT city, LENGTH(city) FROM station ORDER BY LENGTH(city) DESC, city LIMIT 1;



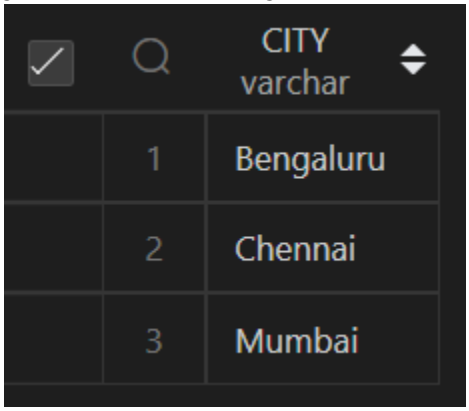
Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from STATION. Your result cannot contain duplicates.

SELECT DISTINCT (city) FROM station WHERE city LIKE 'a%', OR city LIKE 'e%', OR city LIKE 'i%', OR city LIKE 'o%', OR city LIKE 'u%';

	CITY varchar
1	Arunachal

Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.

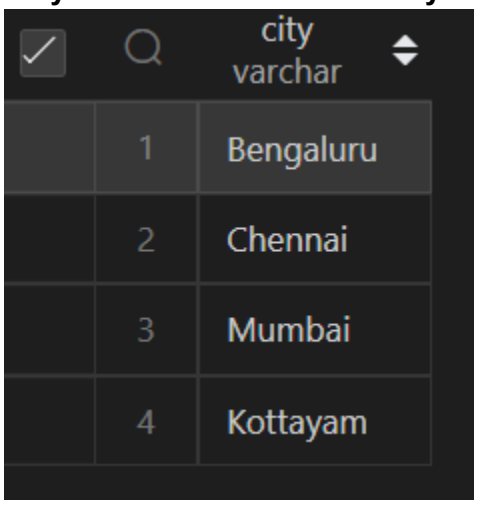
SELECT DISTINCT (city) FROM station WHERE city LIKE '%a', OR city LIKE '%e', OR city LIKE '%i', OR city LIKE '%o', OR city LIKE '%u';



	CITY varchar
1	Bengaluru
2	Chennai
3	Mumbai

Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.

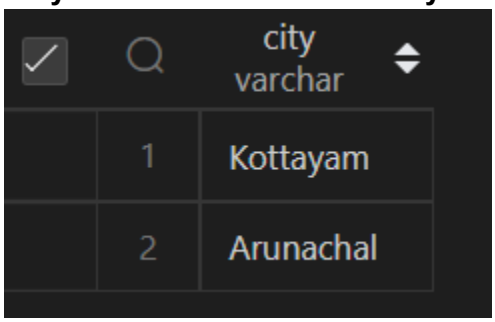
SELECT DISTINCT city FROM station WHERE city NOT RLIKE '^[aeiouAEIOU]';



	city varchar
1	Bengaluru
2	Chennai
3	Mumbai
4	Kottayam

Q14. Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.

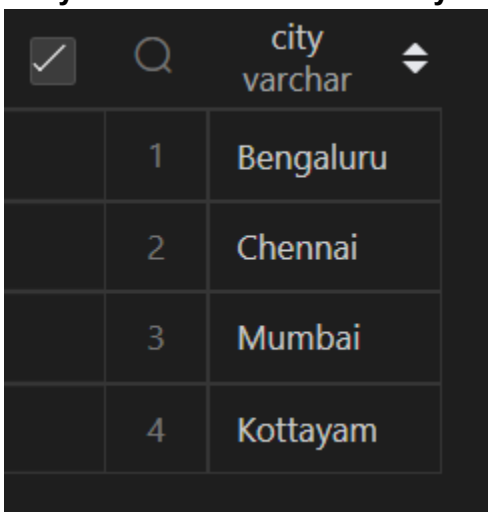
SELECT DISTINCT city FROM station WHERE city NOT RLIKE '[aeiouAEIOU]$';



	city varchar
1	Kottayam
2	Arunachal

Q15. Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.

SELECT DISTINCT city FROM station WHERE city NOT RLIKE '^[aeiouAEIOU].*$';



	city varchar
1	Bengaluru
2	Chennai
3	Mumbai
4	Kottayam

Q16. Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

SELECT DISTINCT city FROM station WHERE city REGEXP '^[^aeiouAEIOU].*[^aeiouAEIOU]$';

	city varchar
1	Kottayam

Q17. Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive. Return the result table in any order.

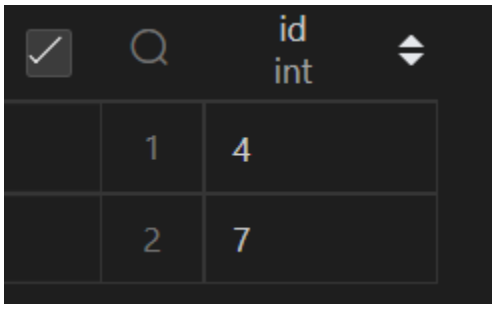
SELECT product_id, product_name FROM Product WHERE product_id NOT IN

```
(SELECT prod_id  FROM   Sales  WHERE  sale_date
        NOT BETWEEN  '2019-01-01' AND '2019-03-31');
```

	product_id int	product_name varchar
1	1	S8

Q18. Write an SQL query to find all the authors that viewed at least one of their own articles. Return the result table sorted by id in ascending order.

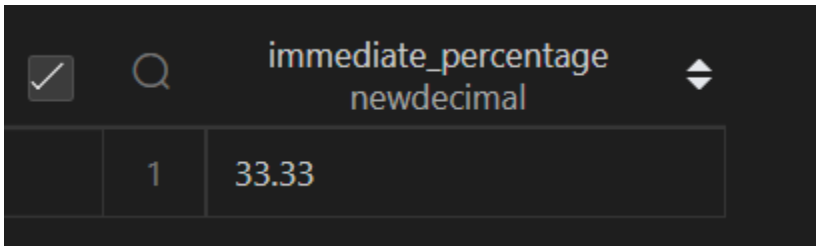
```
SELECT author_id as id FROM Views
WHERE author_id = viewer_id
    GROUP BY author_id ORDER BY author_id asc;
```



	id int
1	4
2	7

Q19. Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

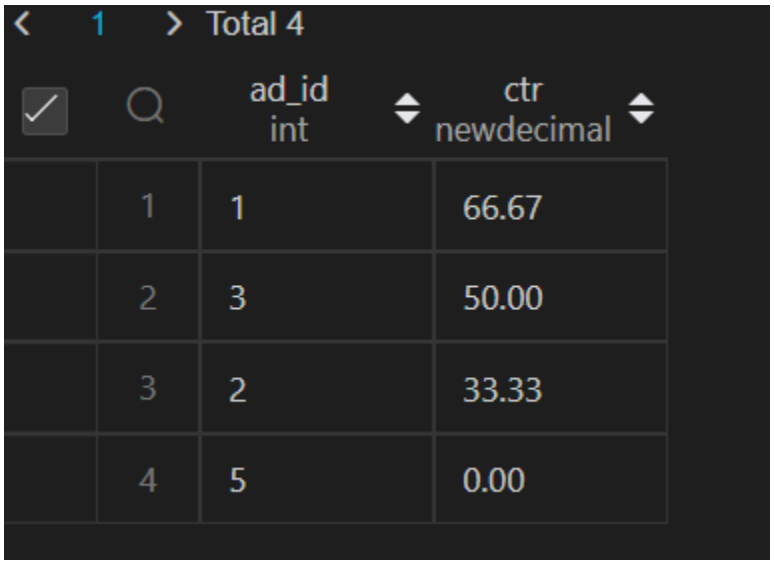
```
SELECT ROUND(100*SUM(CASE WHEN order_date=customer_pref_delivery_date
        THEN 1 ELSE 0 END)/COUNT(1), 2) as immediate_percentage FROM
Delivery;
```



	immediate_percentage newdecimal
1	33.33

Q20. Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points. Return the result table ordered by ctr in descending order and by ad_id in ascending order in case of a tie.

```
        SELECT ad_id,
   ROUND(IFNULL(SUM(CASE WHEN actions='Clicked' THEN 1 ELSE 0 END) * 100 /
   (SUM(CASE WHEN actions='Clicked' THEN 1 ELSE 0 END) + SUM(CASE WHEN
actions='Viewed' THEN 1 ELSE 0 END)), 0), 2)
   as ctr
FROM ads
GROUP BY ad_id
ORDER BY ctr DESC;
```



Q21.Write an SQL query to find the team size of each of the employees. Return result table in any order.

```
select e.employee_id, (select count(team_id) from Employee WHERE
                e.team_id = team_id ) as team_size from Employee e;
```

	employee_id int	team_size bigint
1	1	3
2	2	3
3	3	3
4	4	1
5	5	2
6	6	2

Q22. Write an SQL query to find the type of weather in each country for November 2019. The type of weather is:

- Cold if the average weather_state is less than or equal 15,
- Hot if the average weather_state is greater than or equal to 25, and
- Warm otherwise.

```
select country_name, case when avg(weather_state) <= 15 then "Cold"
                          when avg(weather_state) >= 25 then "Hot"
                          else "Warm" end as weather_type
from Countries inner join Weather
on Countries.country_id = Weather.country_id
where left(day, 7) = '2019-11' group by country_name;
```

	country_name varchar	weather_type varchar
1	USA	Cold
2	Australia	Cold
3	China	Warm
4	Peru	Hot
5	Morocco	Hot

Q23. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places.

SELECT a.product_id, round(SUM(a.units * b.price) / SUM(a.units), 2) AS average_price FROM UnitsSold a JOIN prices b
ON (a.product_id = b.product_id
AND a.purchase_date >= b.start_dates
AND a.purchase_date <= b.end_date) group by product_id;

	product_id int	average_price newdecimal
1	1	6.96
2	2	16.96

Q24. Write an SQL query to report the first login date for each player.

SELECT player_id, MIN(event_date) AS first_login FROM Activity
GROUP BY player_id;

```
▷ Execute
create table activity(
    player_id int,
    device_id int,
    event_date date,
    games_played int,
    primary key(player_id, event_date)
);
▷ Execute
insert into activity values
(1, 2, '2016-03-01', 5),
(1, 2, '2016-05-02', 6),
(2, 3, '2017-06-25', 1),
(3, 1, '2016-03-02', 0),
(3, 4, '2018-07-03', 5);
▷ Execute
select player_id, min(event_date) as first_login from activity group by player_id;  3ms
```

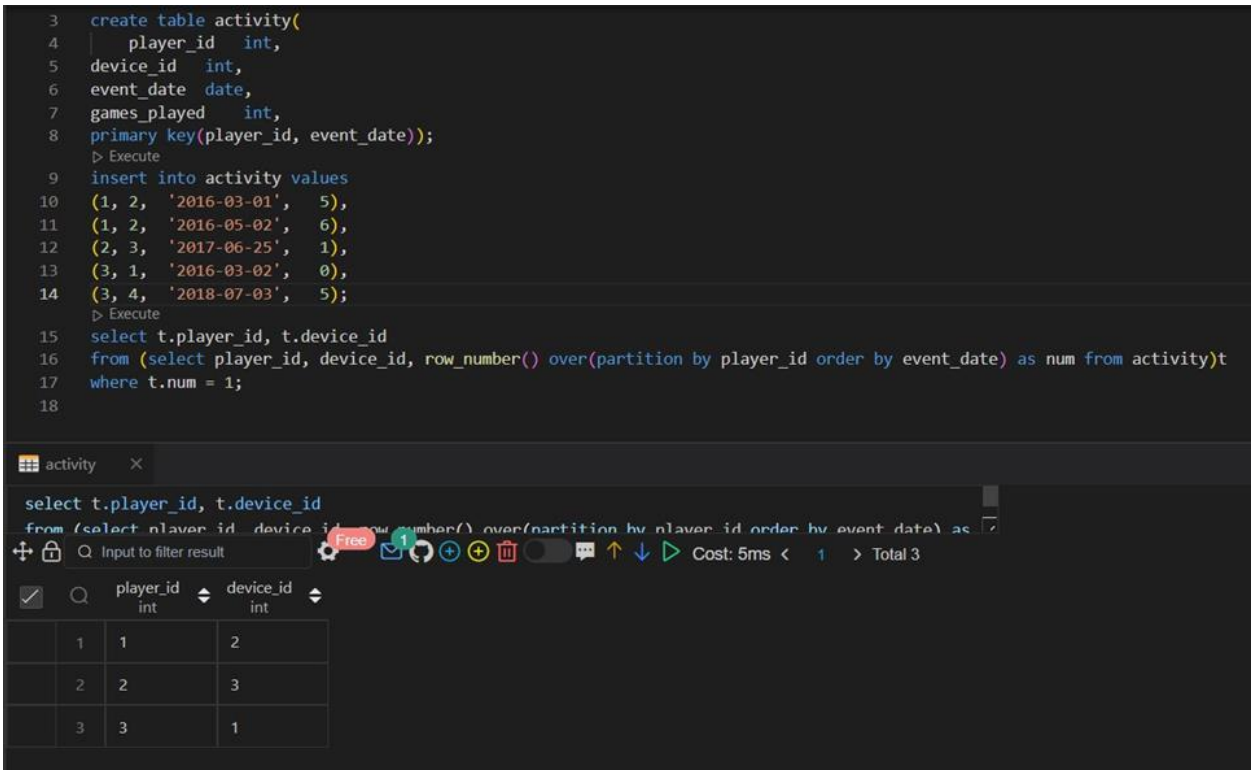
Result

Search results Free Cost: 4ms 1 Total 3

	player_id int	first_login date
1	1	2016-03-01
2	2	2017-06-25
3	3	2016-03-02

Q25. Write an SQL query to report the device that is first logged in for each player.
Return the result table in any order.

select t.player_id, t.device_id from (select player_id, device_id, row_number() over(partition by player_id order by event_date) as num from activity)t where t.num = 1;



26. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.
Return result table in any order.

select p.product_name, sum(o.unit) as unit from Products p left join Orders o on p.product_id = o.product_id
where month(o.order_date) = 2 and year(o.order_date) = 2020
group by p.product_id having unit >= 100;

```sql
create table Orders
(product_id int,
order_date  date,
Unit    int,
foreign key(product_id) references Products(product_id)
);
▷ Execute
insert into Products values
(1, 'Leetcode Solutions', 'Book'),
(2, 'Jewels of Stringology', 'Book'),
(3, 'HP',    'Laptop'),
(4, 'Lenovo',    'Laptop'),
(5, 'Leetcode Kit', 'T-shirt');
▷ Execute
insert into Orders values
(1, '2020-02-05',    60),
(1, '2020-02-10',    70),
(2, '2020-01-18',    30),
(2, '2020-02-11',    80),
(3, '2020-02-17',    2),
(3, '2020-02-24',    3),
(4, '2020-03-01',    20),
(4, '2020-03-04',    30),
(4, '2020-03-04',    60),
(5, '2020-02-25',    50),
(5, '2020-02-27',    50),
(5, '2020-03-01',    50);
▷ Execute
select p.product_name, sum(o.unit) as unit
from
Products p
left join
Orders o
on p.product_id = o.product_id
where month(o.order_date) = 2 and year(o.order_date) = 2020
group by p.product_id
having unit >= 100;
```

Product

```sql
select p.product_name, sum(o.unit) as unit
from
```

Input to filter result | Free | Cost: 2ms

	product_name varchar	unit newdecimal
1	Leetcode Solutions	130
2	Leetcode Kit	100

27. Write an SQL query to find the users who have valid emails. A valid e-mail has a prefix name and a domain where:

- The prefix name is a string that may contain letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'. The prefix name must start with a letter. ● The domain is '@leetcode.com'.

select user_id, name, mail from Users where mail regexp '^[a-zA-Z]+[a-zA-Z0-9_\.\-]*@leetcode[\.]com' order by user_id;

```
2   use test;
    ▷ Execute
3   create table Users
4   (user_id    int,
5   name    varchar(15),
6   mail    varchar(30)
7   );
    ▷ Execute
8   insert into Users values
9   (1, 'Winston',  'winston@leetcode.com'),
10  (2, 'Jonathan', 'jonathanisgreat'),
11  (3, 'Annabelle',    'bella-@leetcode.com'),
12  (4, 'Sally',    'sally.come@leetcode.com'),
13  (5, 'Marwan',   'quarz#2020@leetcode.com'),
14  (6, 'David',    'david69@gmail.com'),
15  (7, 'Shapiro',  '.shapo@leetco de.com');
    ▷ Execute
16  select user_id, name, mail from Users
17  where
18  mail regexp '^[a-zA-Z]+[a-zA-Z0-9_\.\-]*@leetcode[\.]com'
19  order by user_id;
20
```

Users

```
select user_id, name, mail from Users
where
```

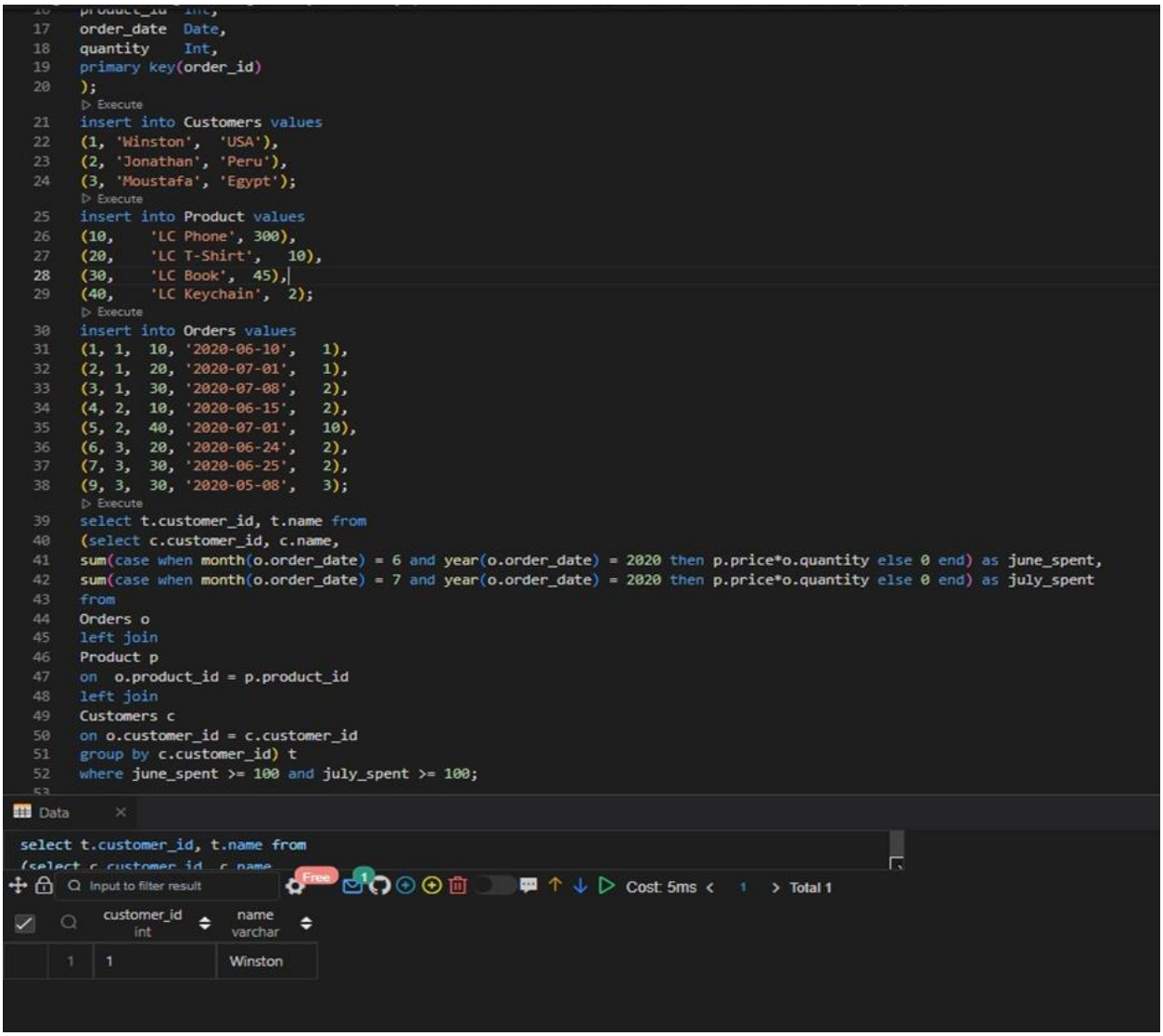
Input to filter result

Free 1

	user_id int	name varchar	mail varchar
1	1	Winston	winston@leetcode.com
2	3	Annabelle	bella-@leetcode.com
3	4	Sally	sally.come@leetcode.com

28.Write an SQL query to report the customer_id and customer_name of customers who have spent at least $100 in each month of June and July 2020.
Return the result table in any order.

```
select t.customer_id, t.name from (select
c.customer_id, c.name,
sum(case when month(o.order_date) = 6 and year(o.order_date) = 2020 then
p.price*o.quantity else 0 end) as june_spent,
sum(case when month(o.order_date) = 7 and year(o.order_date) = 2020 then
p.price*o.quantity else 0 end) as july_spent from
Orders o left join Product p on  o.product_id =
p.product_id left join Customers c
on o.customer_id = c.customer_id group by c.customer_id) t where june_spent >= 100
and july_spent >= 100;
```



29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020. Return the result table in any order.

select c.Title from Content c left join TVProgram t on c.content_id = t.content_id where c.Kids_content = 'Y' and c.content_type = 'Movies' and month(t.program_date) = 6 and year(t.program_date) = 2020;

```sql
create table TVProgram
(program_date   Date,
content_id  Int,
channel Varchar(20),
primary key(program_date, content_id)
);
▷ Execute
create table Content
(content_id int,
Title   Varchar(20),
Kids_content    Varchar(1),
content_type    Varchar(15),
primary key(content_id)
);
▷ Execute
insert into TVProgram values
('2020-06-10 08:00',    1,  'LC-Channel'),
('2020-05-11 12:00',    2,  'LC-Channel'),
('2020-05-12 12:00',    3,  'LC-Channel'),
('2020-05-13 14:00',    4,  'Disney Ch'),
('2020-06-18 14:00',    4,  'Disney Ch'),
('2020-07-15 16:00',    5,  'Disney Ch');
▷ Execute
insert into Content values
(1, 'Leetcode Movie', 'N', 'Movies'),
(2, 'Alg. for Kids', 'Y',   'Series'),
(3, 'Database Sols',    'N',    'Series'),
(4, 'Aladdin',  'Y',    'Movies'),
(5, 'Cinderella',   'Y',    'Movies');
▷ Execute
select c.Title from
Content c
left join
TVProgram t
on c.content_id = t.content_id
where c.Kids_content = 'Y' and c.content_type = 'Movies' and month(t.program_date) = 6  and year(t.program_date) = 2020;
```

TVProgram

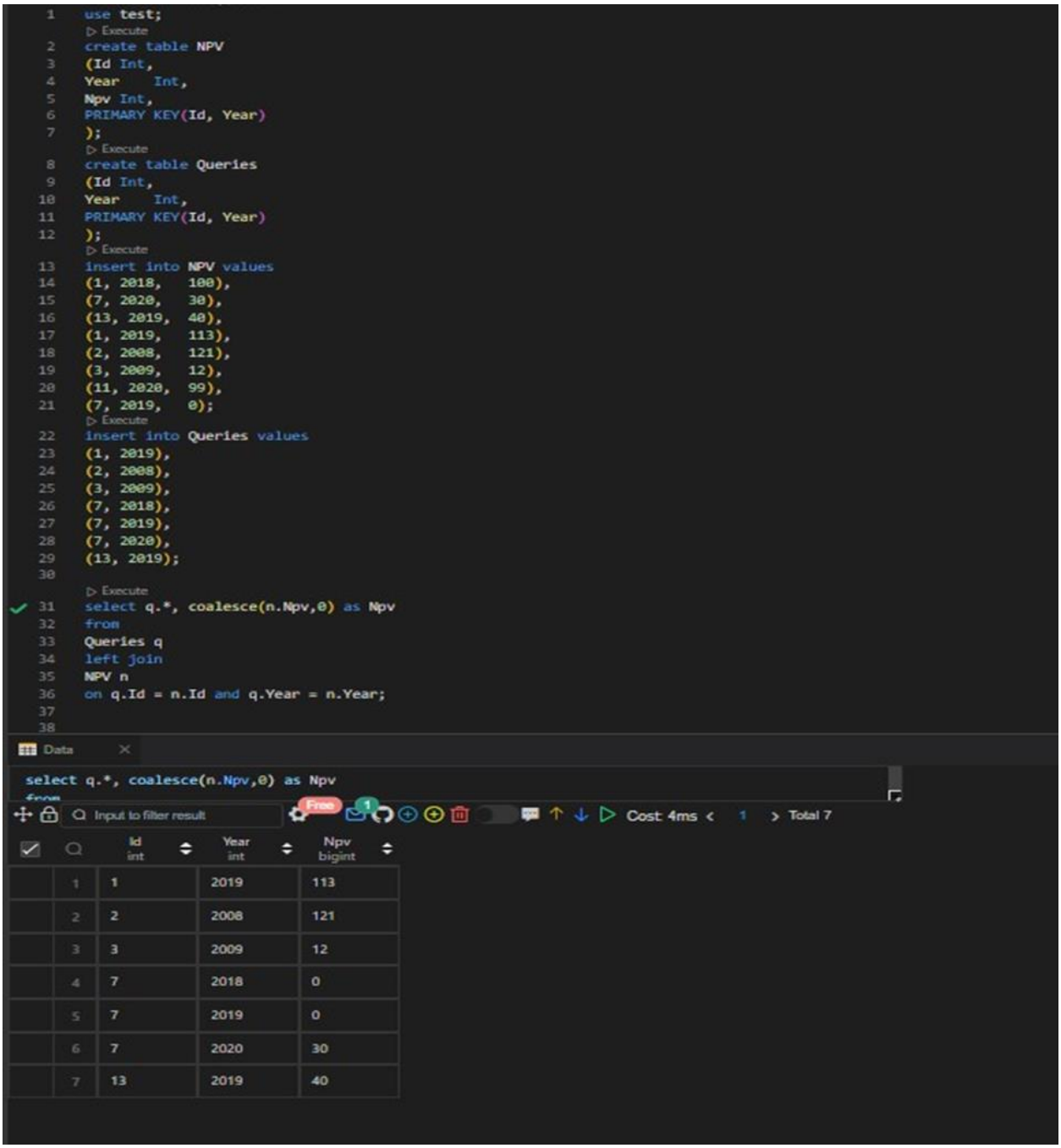
```
select c.Title from
Content c
```

Input to filter result | Free | Cost: 3ms | Total 1

	Title varchar
1	Aladdin

30. Write an SQL query to find the npv of each query of the Queries table. Return the result table in any order.

Select q.*,coalesce(n.Npv,0) as Npv from Queries q left join NPV n on q.Id=n.Id and q.Year=n.Year;



```
use test;
create table NPV
(Id Int,
Year    Int,
Npv Int,
PRIMARY KEY(Id, Year)
);
create table Queries
(Id Int,
Year    Int,
PRIMARY KEY(Id, Year)
);
insert into NPV values
(1, 2018,   100),
(7, 2020,   30),
(13, 2019,  40),
(1, 2019,   113),
(2, 2008,   121),
(3, 2009,   12),
(11, 2020,  99),
(7, 2019,   0);
insert into Queries values
(1, 2019),
(2, 2008),
(3, 2009),
(7, 2018),
(7, 2019),
(7, 2020),
(13, 2019);

select q.*, coalesce(n.Npv,0) as Npv
from
Queries q
left join
NPV n
on q.Id = n.Id and q.Year = n.Year;
```

Data

```
select q.*, coalesce(n.Npv,0) as Npv
```

Input to filter result | Free | Cost: 4ms | 1 | Total 7

	Id int	Year int	Npv bigint
1	1	2019	113
2	2	2008	121
3	3	2009	12
4	7	2018	0
5	7	2019	0
6	7	2020	30
7	13	2019	40

31. Write an SQL query to find the npv of each query of the Queries table. Return the result table in any order.

select q.*, coalesce(n.Npv,0) as Npv from Queries q left join NPV n on q.Id = n.Id and q.Year = n.Year;

```sql
use test;
create table NPV
(Id Int,
Year    Int,
Npv Int,
PRIMARY KEY(Id, Year)
);
create table Queries
(Id Int,
Year    Int,
PRIMARY KEY(Id, Year)
);
insert into NPV values
(1, 2018,   100),
(7, 2020,   30),
(13, 2019,  40),
(1, 2019,   113),
(2, 2008,   121),
(3, 2009,   12),
(11, 2020,  99),
(7, 2019,   0);
insert into Queries values
(1, 2019),
(2, 2008),
(3, 2009),
(7, 2018),
(7, 2019),
(7, 2020),
(13, 2019);

select q.*, coalesce(n.Npv,0) as Npv
from
Queries q
left join
NPV n
on q.Id = n.Id and q.Year = n.Year;
```

Data

```sql
select q.*, coalesce(n.Npv,0) as Npv
```

Input to filter result | Cost: 4ms | Total 7

	Id int	Year int	Npv bigint
1	1	2019	113
2	2	2008	121
3	3	2009	12
4	7	2018	0
5	7	2019	0
6	7	2020	30
7	13	2019	40

32. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

Select u.unique_id, e.name from employees e left join employeeUNI u on e.id=u.id;

```sql
create table employeeUNI
(id Int,
unique_id   Int,
primary key(id, unique_id)
);
▷ Execute
insert into employees values
(1, 'Alice'),
(7, 'Bob'),
(11, 'Meir'),
(90, 'Winston'),
(3, 'Jonathan');
▷ Execute
insert into employeeUNI values
(3, 1),
(11, 2),
(90, 3);

▷ Execute
select u.unique_id, e.name
from
employees e
left join
employeeUNI u
on e.id = u.id;
```

Data

```sql
select u.unique_id, e.name
from
```

Input to filter result | Free | Cost: 0ms | 1 | Total 5

	unique_id int	name varchar
1	(NULL)	Alice
2	(NULL)	Bob
3	1	Jonathan
4	2	Meir
5	3	Winston

33. Write an SQL query to report the distance traveled by each user.
Return the result table ordered by travelled_distance in descending order, if two or more users traveled the same distance, order them by their name in ascending order.

Select u.name, coalesce(sum(r.distance),0) as travelled_distance from users u left join rides r on u.id=r.user_id group by u.name order by travelled_distance desc, u.name;

```
11  distance    Int,
12  primary key(id));
    ▷ Execute
13  insert into users values
14  (1, 'Alice'),
15  (2, 'Bob'),
16  (3, 'Alex'),
17  (4, 'Donald'),
18  (7, 'Lee'),
19  (13, 'Jonathan'),
20  (19, 'Elvis');
    ▷ Execute
21  insert into rides values
22  (1, 1,  120),
23  (2, 2,  317),
24  (3, 3,  222),
25  (4, 7,  100),
26  (5, 13, 312),
27  (6, 19, 50),
28  (7, 7,  120),
29  (8, 19, 400),
30  (9, 7,  230);
    ▷ Execute
31  select u.name, coalesce(sum(r.distance),0) as travelled_distance
32  from
33  users u
34  left join
35  rides r
36  on u.id = r.user_id
37  group by u.name
38  order by travelled_distance desc, u.name;
```

Data

```
select u.name, coalesce(sum(r.distance),0) as travelled_distance
from
```

Input to filter result | Free | Cost: 3ms | Total 7

	name varchar	travelled_distance newdecimal
1	Elvis	450
2	Lee	450
3	Bob	317
4	Jonathan	312
5	Alex	222
6	Alice	120
7	Donald	0

34. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount. Return result table in any order.

select p.product_name, sum(o.unit) as unit from Products p left join Orders o on p.product_id = o.product_id where month(o.order_date) = 2 and year(o.order_date) = 2020 group by p.product_id having unit >= 100;

```sql
create table Orders
(product_id int,
order_date  date,
Unit    int,
foreign key(product_id) references Products(product_id)
);
▷ Execute
insert into Products values
(1, 'Leetcode Solutions', 'Book'),
(2, 'Jewels of Stringology', 'Book'),
(3, 'HP',    'Laptop'),
(4, 'Lenovo',    'Laptop'),
(5, 'Leetcode Kit', 'T-shirt');
▷ Execute
insert into Orders values
(1, '2020-02-05',   60),
(1, '2020-02-10',   70),
(2, '2020-01-18',   30),
(2, '2020-02-11',   80),
(3, '2020-02-17',   2),
(3, '2020-02-24',   3),
(4, '2020-03-01',   20),
(4, '2020-03-04',   30),
(4, '2020-03-04',   60),
(5, '2020-02-25',   50),
(5, '2020-02-27',   50),
(5, '2020-03-01',   50);
▷ Execute
select p.product_name, sum(o.unit) as unit
from
Products p
left join
Orders o
on p.product_id = o.product_id
where month(o.order_date) = 2 and year(o.order_date) = 2020
group by p.product_id
having unit >= 100;
```

Product

```
select p.product_name, sum(o.unit) as unit
from
```

Input to filter result — Free — Cost: 2ms

	product_name varchar	unit newdecimal
1	Leetcode Solutions	130
2	Leetcode Kit	100

35. Write an SQL query to:

- Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.
- Find the movie name with the highest average rating in February 2020. In case of a tie, return the lexicographically smaller movie name.

(select t1.name as Results from
(select u.name, count(u.user_id), dense_rank() over(order by count(user_id) desc, u.name) as r1 FROM Users u left join MovieRating m on u.user_id = m.user_id group by u.user_id) t1 where r1 = 1) union
(select t2.title as Results from
(select mo.title, avg(m.rating), dense_rank() over(order by avg(m.rating)desc, mo.title) as r2 from Movies mo left join MovieRating m on mo.movie_id = m.movie_id where month(m.created_at) = 2 and year(m.created_at) = 2020 group by m.movie_id) t2 where r2 = 1);

```
20  insert into Movies values
21  (1, 'Avengers'),
22  (2, 'Frozen 2'),
23  (3, 'Joker');
    ▷ Execute
24  insert into Users values
25  (1, 'Daniel'),
26  (2, 'Monica'),
27  (3, 'Maria'),
28  (4, 'James');
    ▷ Execute
29  insert into MovieRating values
30  (1, 1,  3,  '2020-01-12'),
31  (1, 2,  4,  '2020-02-11'),
32  (1, 3,  2,  '2020-02-12'),
33  (1, 4,  1,  '2020-01-01'),
34  (2, 1,  5,  '2020-02-17'),
35  (2, 2,  2,  '2020-02-01'),
36  (2, 3,  2,  '2020-03-01'),
37  (3, 1,  3,  '2020-02-22'),
38  (3, 2,  4,  '2020-02-25');
    ▷ Execute
✓ 39  (select t1.name as Results from
40  (select u.name, count(u.user_id), dense_rank() over(order by count(user_id) desc, u.name) as r1 FROM
41  Users u
42  left join
43  MovieRating m
44  on u.user_id = m.user_id
45  group by u.user_id) t1
46  where r1 = 1)
47  union
48  (select t2.title as Results from
49  (select mo.title, avg(m.rating), dense_rank() over(order by avg(m.rating)desc, mo.title) as r2 from
50  Movies mo
51  left join
52  MovieRating m
53  on mo.movie_id = m.movie_id
54  where month(m.created_at) = 2 and year(m.created_at) = 2020
55  group by m.movie_id) t2
56  where r2 = 1);
```

Data

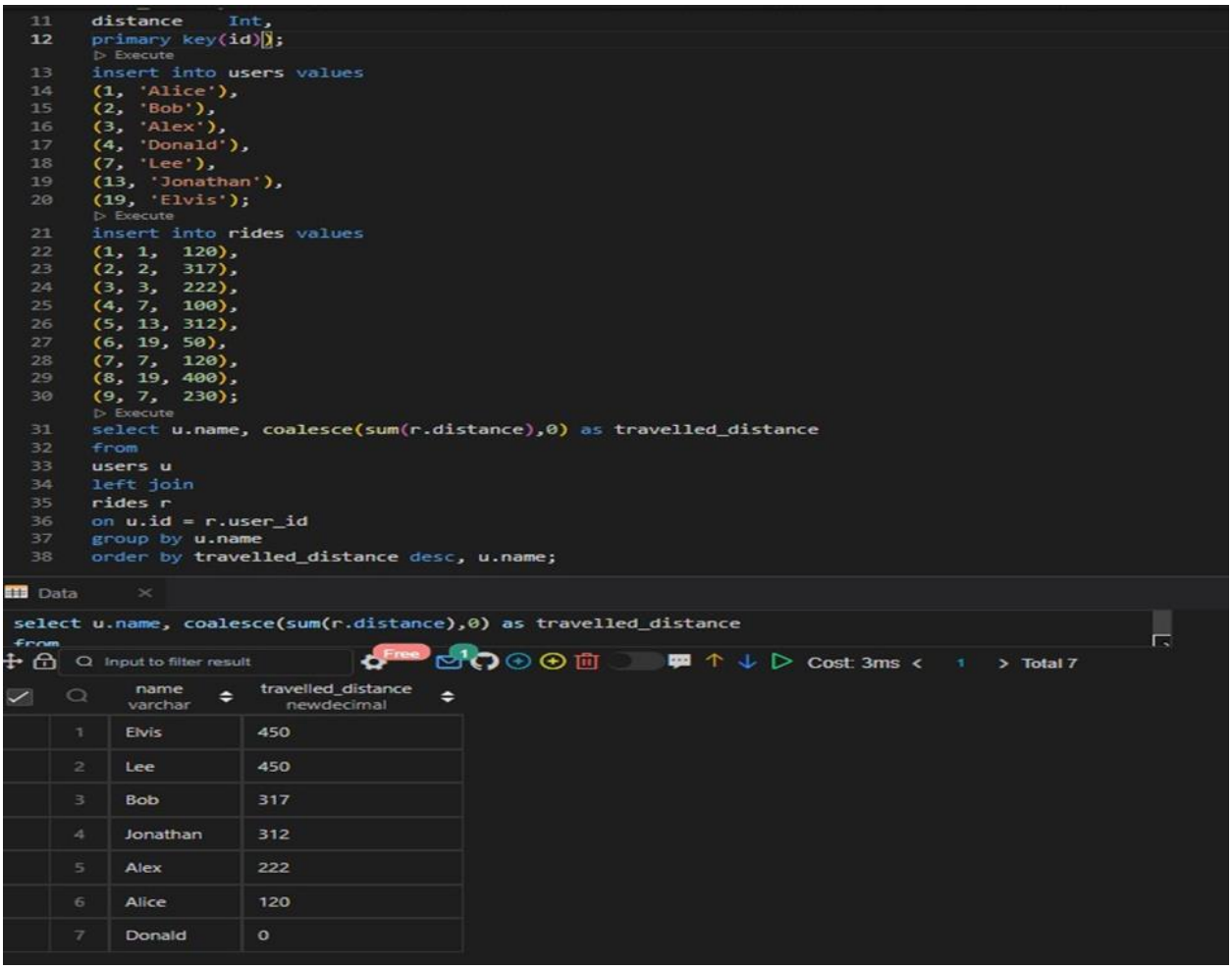
```
(select t1.name as Results from
(select u.name, count(u.user_id), dense_rank() over(order by count(user_id) desc, u.name) as r1
```

Input to filter result | Free | Cost: 2ms | 1 | Total 2

	Results varchar
1	Daniel
2	Frozen 2

36. Write an SQL query to report the distance traveled by each user.
Return the result table ordered by travelled_distance in descending order, if two or more users traveled the same distance, order them by their name in ascending order.

select u.name, coalesce(sum(r.distance),0) as travelled_distance
from users u left join rides r on u.id = r.user_id group by u.name
order by travelled_distance desc, u.name;



```
distance    Int,
primary key(id));
▷ Execute
insert into users values
(1, 'Alice'),
(2, 'Bob'),
(3, 'Alex'),
(4, 'Donald'),
(7, 'Lee'),
(13, 'Jonathan'),
(19, 'Elvis');
▷ Execute
insert into rides values
(1, 1,  120),
(2, 2,  317),
(3, 3,  222),
(4, 7,  100),
(5, 13, 312),
(6, 19, 50),
(7, 7,  120),
(8, 19, 400),
(9, 7,  230);
▷ Execute
select u.name, coalesce(sum(r.distance),0) as travelled_distance
from
users u
left join
rides r
on u.id = r.user_id
group by u.name
order by travelled_distance desc, u.name;
```

Data

```
select u.name, coalesce(sum(r.distance),0) as travelled_distance
from
```

Input to filter result | Free | Cost: 3ms | Total 7

	name varchar	travelled_distance newdecimal
1	Elvis	450
2	Lee	450
3	Bob	317
4	Jonathan	312
5	Alex	222
6	Alice	120
7	Donald	0

37. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null. Return the result table in any order.

select u.unique_id, e.name from employees e left join employeeUNI u on e.id = u.id;

```sql
create table employeeUNI
(id Int,
unique_id   Int,
primary key(id, unique_id)
);
▷ Execute
insert into employees values
(1, 'Alice'),
(7, 'Bob'),
(11, 'Meir'),
(90, 'Winston'),
(3, 'Jonathan');
▷ Execute
insert into employeeUNI values
(3, 1),
(11, 2),
(90, 3);

▷ Execute
select u.unique_id, e.name
from
employees e
left join
employeeUNI u
on e.id = u.id;
```

Data

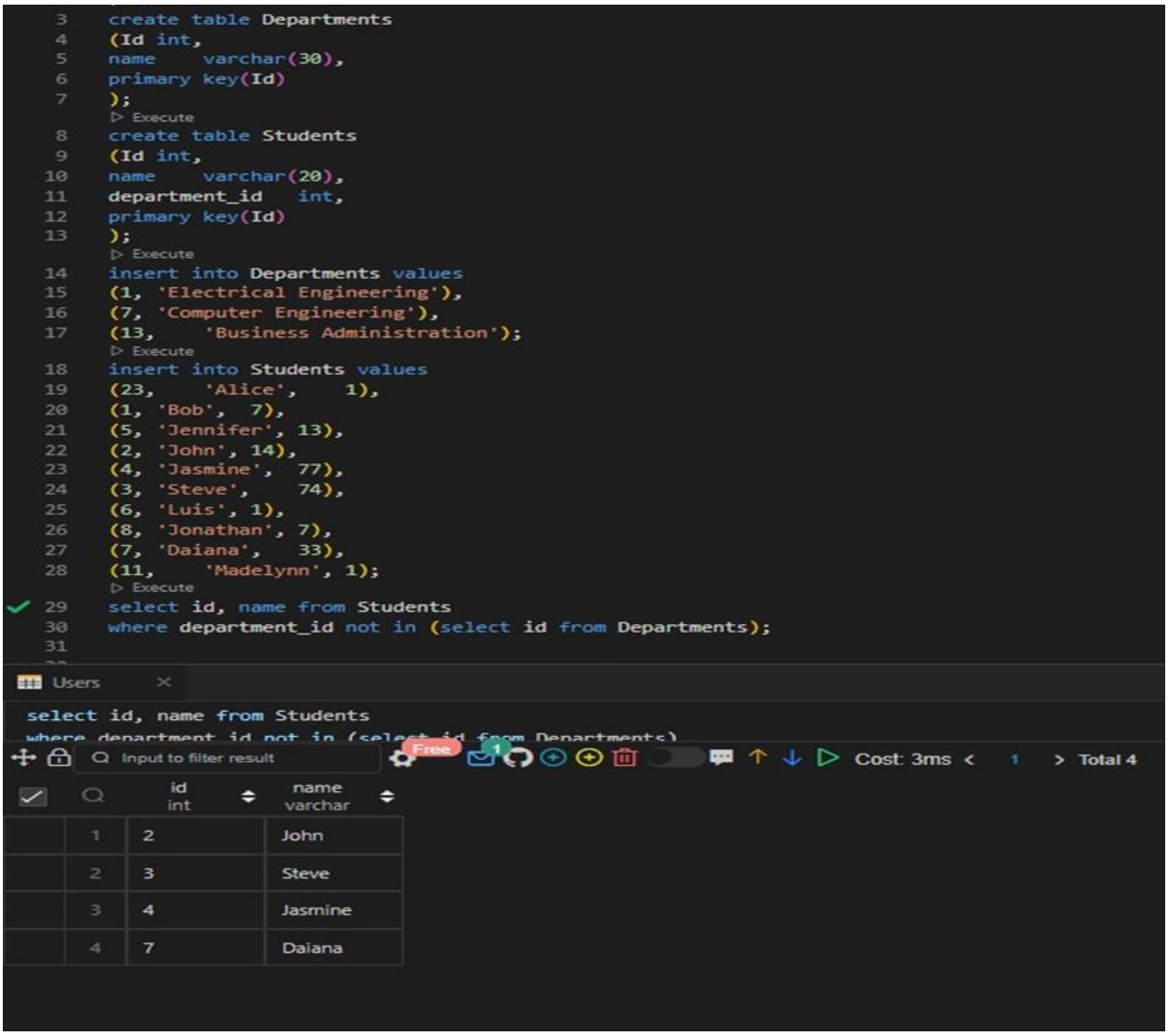
```sql
select u.unique_id, e.name
from
```

Input to filter result | Free | Cost: 0ms | 1 | Total 5

	unique_id int	name varchar
1	(NULL)	Alice
2	(NULL)	Bob
3	1	Jonathan
4	2	Meir
5	3	Winston

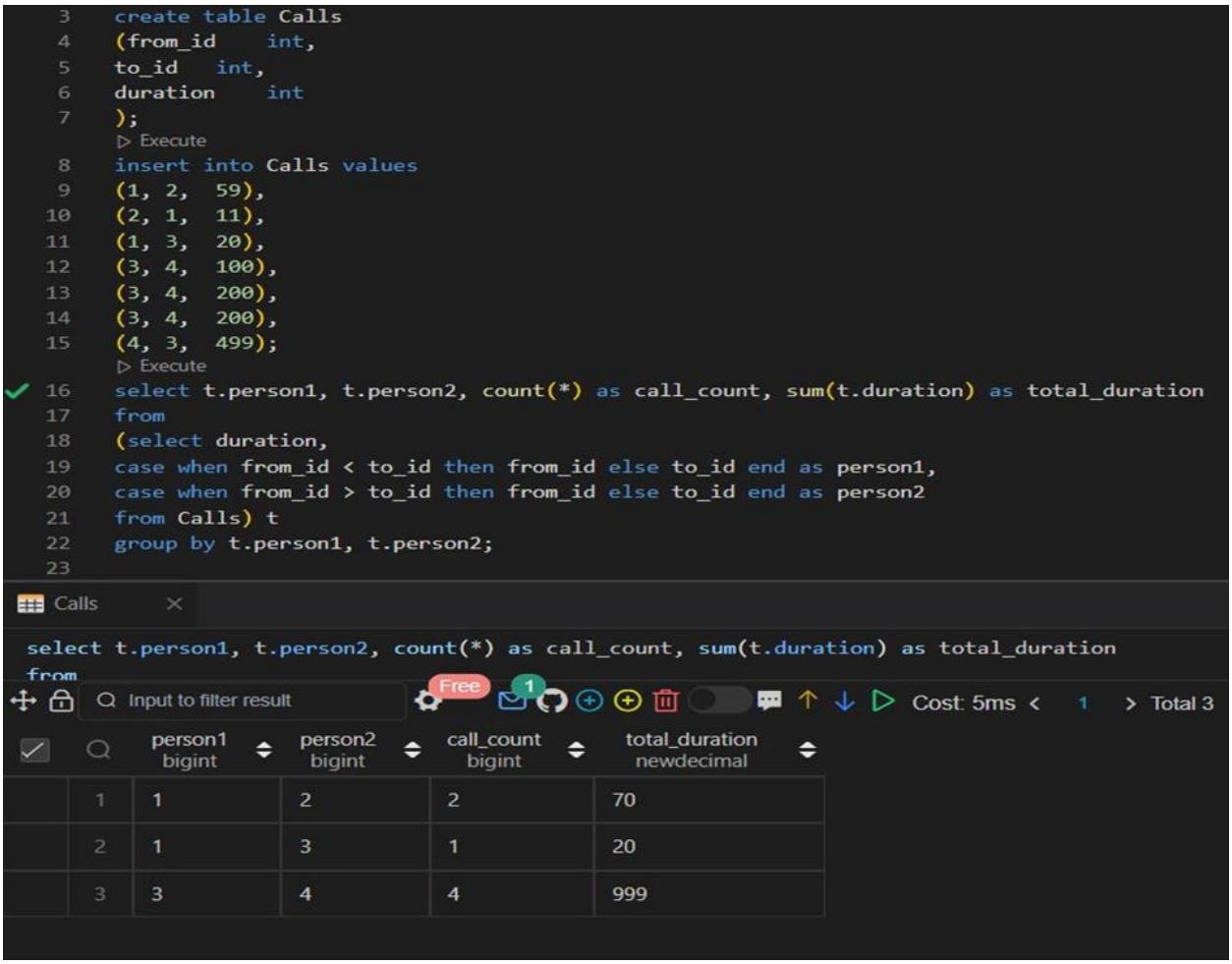
38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist. Return the result table in any order.

select id, name from Students where department_id not in (select id from Departments);



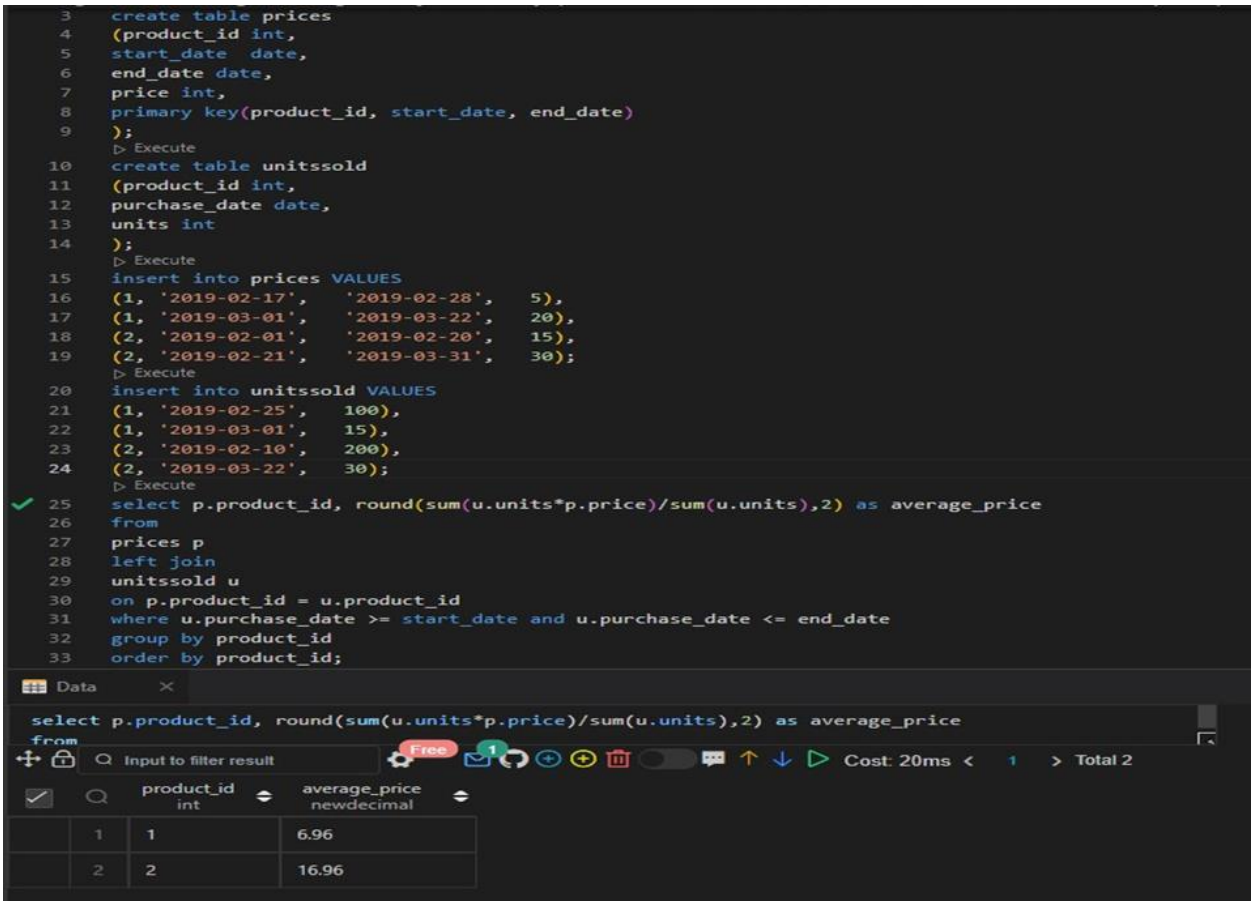
39. Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2. Return the result table in any order.

select t.person1, t.person2, count(*) as call_count, sum(t.duration) as total_duration from
(select duration, case when from_id < to_id then from_id else to_id end as person1, case when from_id > to_id then from_id else to_id end as person2 from Calls) t group by t.person1, t.person2;



40. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places. Return the result table in any order.

Select p.product_id, round(sum(u.units*p.price)/sum(u.units),2) as average_price from prices p left join unitssold u on p.product_id=u.product_id where u.purchase_date >= start_date and u.purchase_date <= end_date group by product_id order by product_id;



```
create table prices
(product_id int,
start_date  date,
end_date date,
price int,
primary key(product_id, start_date, end_date)
);
create table unitssold
(product_id int,
purchase_date date,
units int
);
insert into prices VALUES
(1, '2019-02-17',   '2019-02-28',   5),
(1, '2019-03-01',   '2019-03-22',   20),
(2, '2019-02-01',   '2019-02-20',   15),
(2, '2019-02-21',   '2019-03-31',   30);
insert into unitssold VALUES
(1, '2019-02-25',   100),
(1, '2019-03-01',   15),
(2, '2019-02-10',   200),
(2, '2019-03-22',   30);
select p.product_id, round(sum(u.units*p.price)/sum(u.units),2) as average_price
from
prices p
left join
unitssold u
on p.product_id = u.product_id
where u.purchase_date >= start_date and u.purchase_date <= end_date
group by product_id
order by product_id;
```

Data

```
select p.product_id, round(sum(u.units*p.price)/sum(u.units),2) as average_price
from
```

Input to filter result | Free | Cost: 20ms | Total 2

	product_id int	average_price newdecimal
1	1	6.96
2	2	16.96

41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse. Return the result table in any order.

select w.name as warehouse_name, sum(p.width*p.length*p.height*w.units) as volume from warehouse w left join products p
on w.product_id = p.product_id
group by w.name order by
w.name;

```sql
create table warehouse
(name   varchar(15),
product_id  int,
units   int,
primary key(name, product_id)
);
```
Execute
```sql
create table products
(product_id int,
product_name varchar(15),
Width   int,
Length  int,
Height  int,
primary key(product_id)
);
```
Execute
```sql
insert into warehouse values
('LCHouse1',    1,  1),
('LCHouse1',    2,  10),
('LCHouse1',    3,  5),
('LCHouse2',    1,  2),
('LCHouse2',    2,  2),
('LCHouse3',    4,  1);
```
Execute
```sql
insert into products values
(1, 'LC-TV',    5,  50, 40),
(2, 'LC-KeyChain',  5,    5,    5),
(3, 'LC-Phone', 2,  10, 10),
(4, 'LC-T-Shirt',   4,  10, 20);
```
Execute
```sql
select w.name as warehouse_name, sum(p.width*p.length*p.height*w.units) as volume
from
warehouse w
left join
products p
on w.product_id = p.product_id
group by w.name
order by w.name;
```

Data

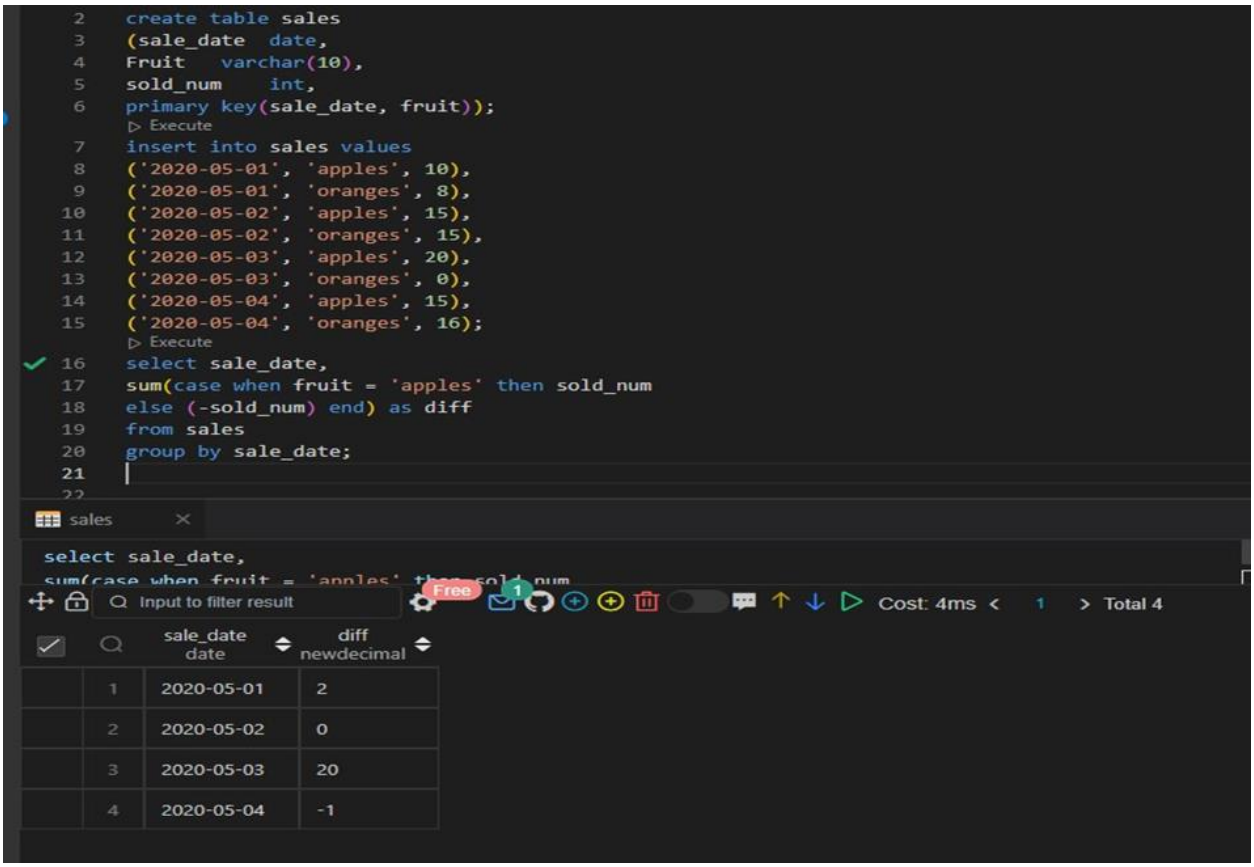
```sql
select w.name as warehouse_name, sum(p.width*p.length*p.height*w.units) as volume
from
```

Input to filter result | Free | Cost: 23ms | 1 | Total 3

	warehouse_name varchar	volume newdecimal
1	LCHouse1	12250
2	LCHouse2	20250
3	LCHouse3	800

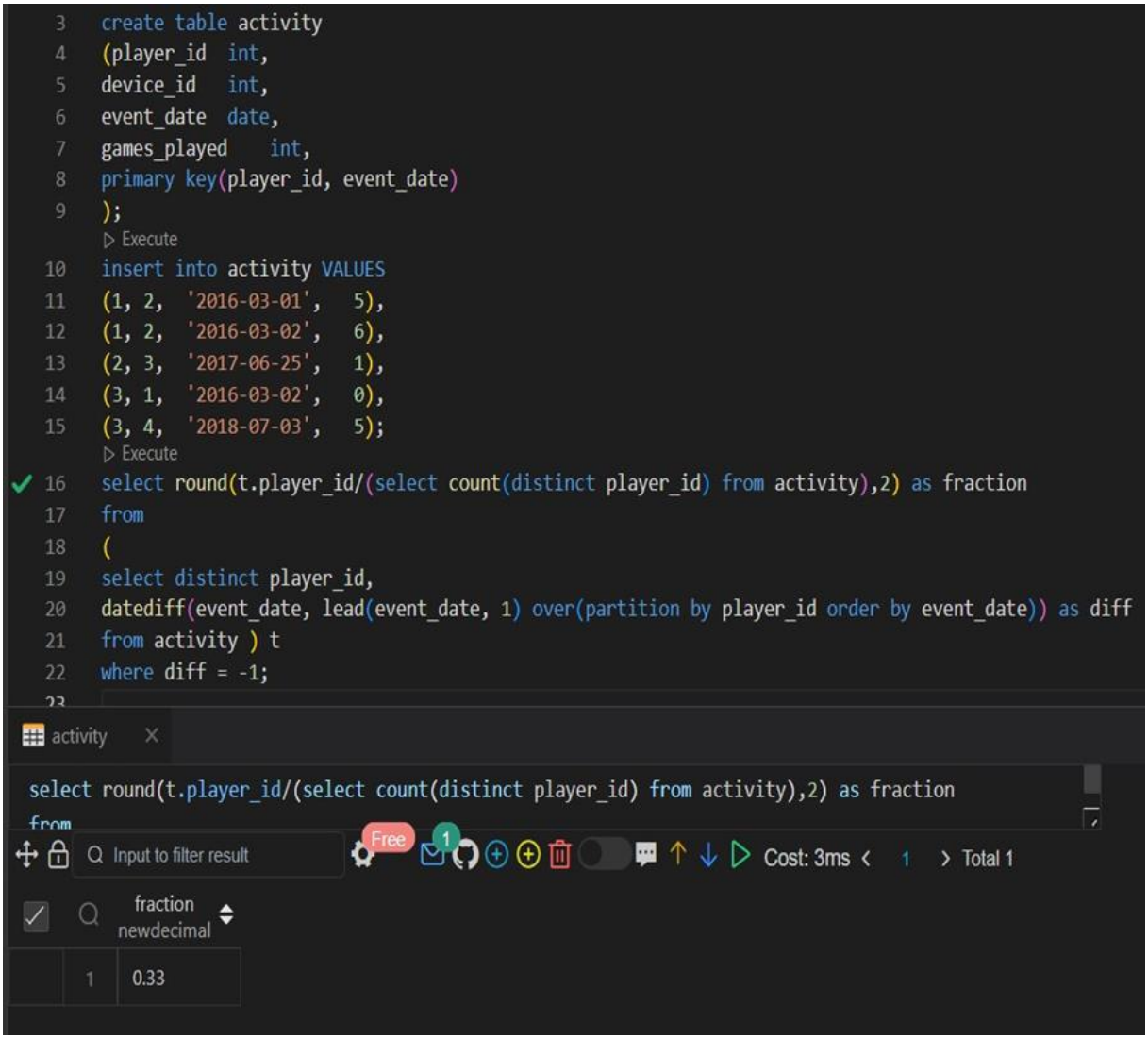
42. Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale_date.

select sale_date, sum(case when fruit = 'apples' then sold_num else (-sold_num) end) as diff from sales group by sale_date;



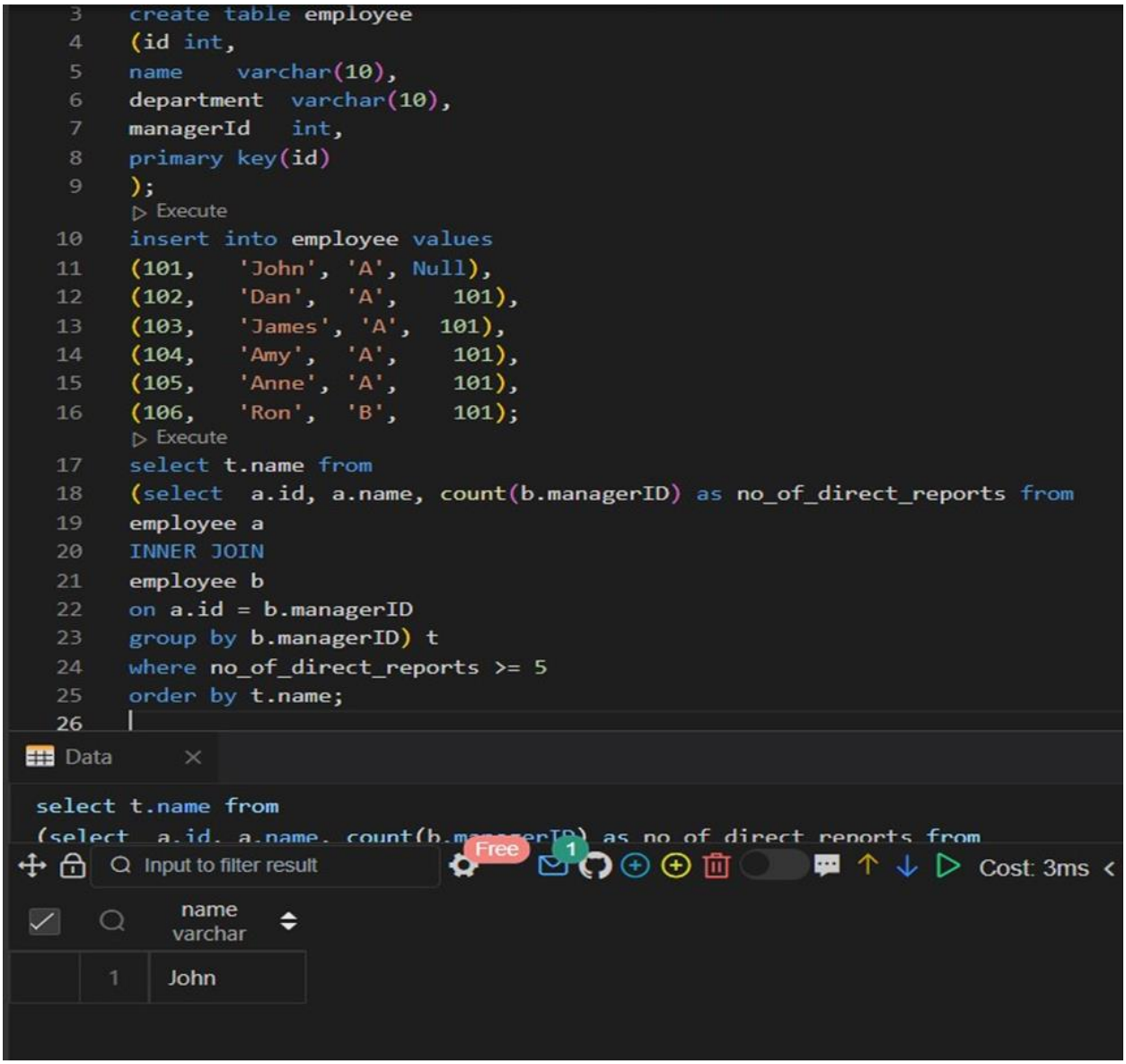
43. Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction from (select distinct player_id, datediff(event_date, lead(event_date, 1) over(partition by player_id order by event_date)) as diff from activity) t where diff = -1;



44. Write an SQL query to report the managers with at least five direct reports.
Return the result table in any order.

Select t.name from (select a.id, a.name, count(b.managerID) as no_of_direct_reports from employee a INNER JOIN employee b on a.id = b.managerID group by b.managerID) t where no_of_direct_reports >= 5 order by t.name;



45. Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students). Return the result table ordered by student_number in descending order. In case of a tie, order them by dept_name alphabetically.

select d.dept_name, count(s.dept_id) as student_number from department d left join student s on s.dept_id = d.dept_id group by d.dept_id order by student_number desc, dept_name;

```sql
create table department
(dept_id    Int,
dept_name   Varchar(15),
primary key(dept_id)
);
▷ Execute
insert into student values
(1, 'Jack', 'M',    1),
(2, 'Jane', 'F',    1),
(3, 'Mark', 'M',    2);
▷ Execute
insert into department values
(1, 'Engineering'),
(2, 'Science'),
(3, 'Law');
▷ Execute
select * from student;
▷ Execute
select d.dept_name, count(s.dept_id) as student_number from
department d
left join
student s
on s.dept_id = d.dept_id
group by d.dept_id
order by student_number desc, dept_name;
```

Data

```sql
select d.dept_name, count(s.dept_id) as student_number from
department d
```

Input to filter result

	dept_name varchar	student_number bigint
1	Engineering	2
2	Science	1
3	Law	0

46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table. Return the result table in any order.

Select customer_id from customer group by customer_id having count(distinct product_key)=(select count(*) from product);

```
create table product
(product_key int,
primary key(product_key));
▷ Execute
create table customer
(customer_id     int,
product_key int,
foreign key(product_key) references product(product_key)
);
▷ Execute
insert into product values
(5),
(6);
▷ Execute
insert into customer values
(1, 5),
(2, 6),
(3, 5),
(3, 6),
(1, 6);
▷ Execute
select customer_id
from
customer
group by customer_id
having count(distinct product_key)=(select count(*) from product);
```

customer

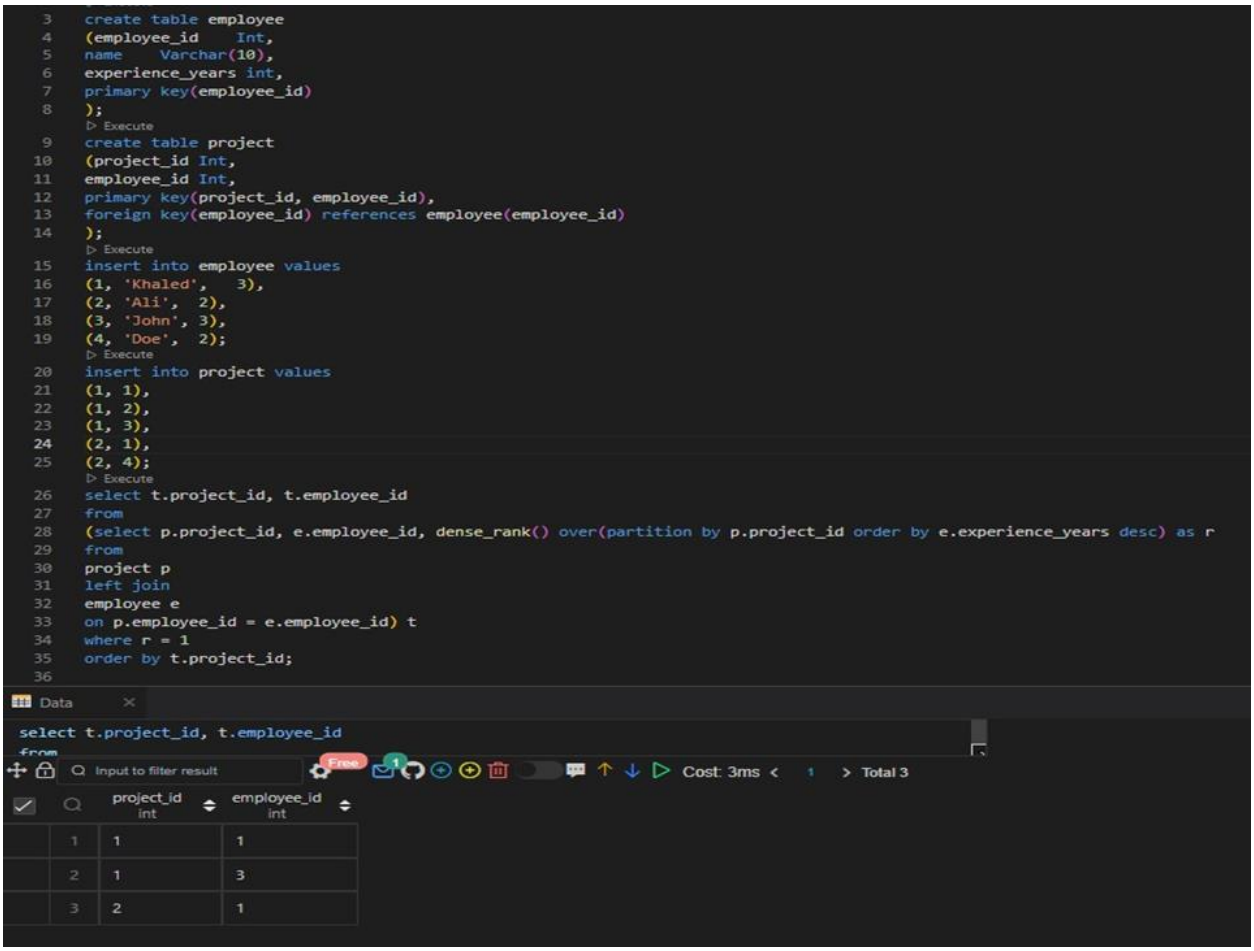
```
select customer_id
from
```

Input to filter result | Free | 1 | Cost: 4ms

	customer_id int
1	1
2	3

47. Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years.
Return the result table in any order.

select t.project_id, t.employee_id
from
(select p.project_id, e.employee_id, dense_rank() over(partition by p.project_id order by e.experience_years desc) as r from project p left join employee e on p.employee_id = e.employee_id) t where r = 1 order by t.project_id;



```
create table employee
(employee_id    Int,
name    Varchar(10),
experience_years int,
primary key(employee_id)
);
create table project
(project_id Int,
employee_id Int,
primary key(project_id, employee_id),
foreign key(employee_id) references employee(employee_id)
);
insert into employee values
(1, 'Khaled',   3),
(2, 'Ali',  2),
(3, 'John', 3),
(4, 'Doe',  2);
insert into project values
(1, 1),
(1, 2),
(1, 3),
(2, 1),
(2, 4);
select t.project_id, t.employee_id
from
(select p.project_id, e.employee_id, dense_rank() over(partition by p.project_id order by e.experience_years desc) as r
from
project p
left join
employee e
on p.employee_id = e.employee_id) t
where r = 1
order by t.project_id;
```

Data

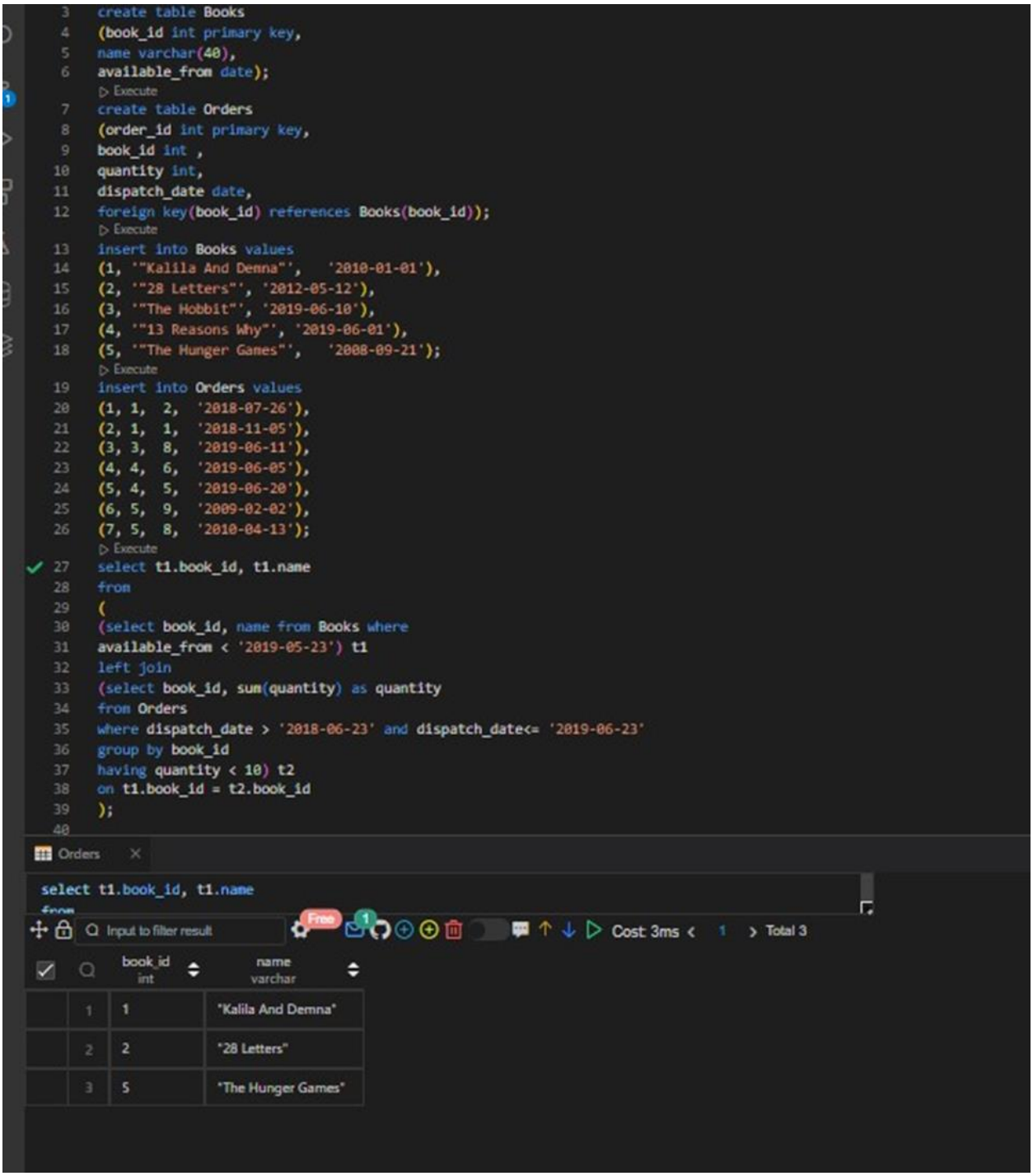
```
select t.project_id, t.employee_id
from
```

Input to filter result | Cost: 3ms | Total 3

	project_id int	employee_id int
1	1	1
2	1	3
3	2	1

48. Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23. Return the result table in any order.

select t1.book_id, t1.name
from
(
(select book_id, name from Books where
available_from < '2019-05-23') t1 left join
(select book_id, sum(quantity) as quantity from Orders where
dispatch_date > '2018-06-23' and dispatch_date<= '2019-06-23' group by
book_id having quantity < 10) t2 on t1.book_id = t2.book_id
);



```
create table Books
(book_id int primary key,
name varchar(40),
available_from date);
▷ Execute
create table Orders
(order_id int primary key,
book_id int ,
quantity int,
dispatch_date date,
foreign key(book_id) references Books(book_id));
▷ Execute
insert into Books values
(1, '"Kalila And Demna"',   '2010-01-01'),
(2, '"28 Letters"', '2012-05-12'),
(3, '"The Hobbit"', '2019-06-10'),
(4, '"13 Reasons Why"', '2019-06-01'),
(5, '"The Hunger Games"',   '2008-09-21');
▷ Execute
insert into Orders values
(1, 1,  2,  '2018-07-26'),
(2, 1,  1,  '2018-11-05'),
(3, 3,  8,  '2019-06-11'),
(4, 4,  6,  '2019-06-05'),
(5, 4,  5,  '2019-06-20'),
(6, 5,  9,  '2009-02-02'),
(7, 5,  8,  '2010-04-13');
▷ Execute
select t1.book_id, t1.name
from
(
(select book_id, name from Books where
available_from < '2019-05-23') t1
left join
(select book_id, sum(quantity) as quantity
from Orders
where dispatch_date > '2018-06-23' and dispatch_date<= '2019-06-23'
group by book_id
having quantity < 10) t2
on t1.book_id = t2.book_id
);
```

Orders

```
select t1.book_id, t1.name
```

Input to filter result | Free | Cost: 3ms | Total 3

	book_id int	name varchar
1	1	"Kalila And Demna"
2	2	"28 Letters"
3	5	"The Hunger Games"

49. Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course_id. Return the result table ordered by student_id in ascending order. The query result format is in the following example.

select t.student_id, t.course_id, t.grade from
(select student_id, course_id, grade, dense_rank() over(partition by student_id order by grade desc, course_id) as r from enrollments) t where r = 1

order by t.student_id;

```
3   create table enrollments
4   (student_id Int,
5   course_id   Int,
6   Grade   Int,
7   primary key(student_id, course_id)
8   );
    ▷ Execute
9   insert into enrollments values
10  (2, 2,  95),
11  (2, 3,  95),
12  (1, 1,  90),
13  (1, 2,  99),
14  (3, 1,  80),
15  (3, 2,  75),
16  (3, 3,  82);
17
    ▷ Execute
✓ 18  select t.student_id, t.course_id, t.grade
19  from
20  (select student_id, course_id, grade, dense_rank() over(partition by student_id order by grade desc, course_id) as r
21  from enrollments) t
22  where r = 1
23  order by t.student_id;
```

enrollments ×

```
select t.student_id, t.course_id, t.grade
from
```

Input to filter result | Free | Cost: 5ms | 1 | Total 3

	student_id int	course_id int	grade int
1	1	2	99
2	2	2	95
3	3	3	82

50. Write an SQL query to find the winner in each group. Return the result table in any order.

```
select t2.group_id, t2.player_id from
(       select t1.group_id, t1.player_id,  dense_rank() over(partition by group_id order
by score desc, player_id) as r from (       select p.*, case when p.player_id =
m.first_player then m.first_score when p.player_id = m.second_player then
m.second_score end as score from
Players p, Matches m where player_id in (first_player,
second_player)
        ) t1 )
t2
where r = 1;
```

```sql
use test;
```
▷ Execute
```sql
create table Players
(
player_id   Int primary key,
group_id    Int
);
```
▷ Execute
```sql
create table Matches
(
    match_id    Int primary key,
first_player    Int,
second_player   Int,
first_score Int,
second_score    Int
);
```
▷ Execute
```sql
insert into Players values
(15,    '1'),
(25,    '1'),
(30,    '1'),
(45,    '1'),
(10,    '2'),
(35,    '2'),
(50,    '2'),
(20,    '3'),
(40,    '3');
```
▷ Execute
```sql
insert into Matches values
(1, 15, 45, 3,  0),
(2, 30, 25, 1,  2),
(3, 30, 15, 2,  0),
(4, 40, 20, 5,  2),
(5, 35, 50, 1,  1);
```
▷ Execute
```sql
select t2.group_id, t2.player_id from
(
    select t1.group_id, t1.player_id,
dense_rank() over(partition by group_id order by score desc, player_id) as r
from
(
    select p.*, case when p.player_id = m.first_player then m.first_score
when p.player_id = m.second_player then m.second_score
end as score
from
Players p, Matches m
where player_id in (first_player, second_player)
    ) t1
) t2
where r = 1;
```

Players

```sql
select t2.group_id, t2.player_id from
```

Input to filter result | Free | Cost 12ms | 1 | Total 3

	group_id int	player_id int
1	1	15
2	2	35
3	3	40