First create a database:

```
CREATE DATABASE SQL_Challenge;
```

-- Select the above created Database

```
USE SQL_Challenge;
```

For Sample Dataset 1

Create the CITY table

```
CREATE TABLE CITY
( ID INT,
  NAME VARCHAR(17),
  COUNTRYCODE VARCHAR(3),
  DISTRICT VARCHAR(20),
  POPULATION INT
);
```

Insert records into CITY table

```
INSERT INTO CITY VALUES(6,"Rotterdam","NLD","Zuid-Holland", 593321);
INSERT INTO CITY VALUES(3878,"Scottsdale","USA","Arizona",202705);
INSERT INTO CITY VALUES(3965,"Corona","USA","California",124966);
INSERT INTO CITY VALUES(3973,"Concord","USA","California",121780);
INSERT INTO CITY VALUES(3977,"Cedar Rapids","USA","Iowa",120758);
INSERT INTO CITY VALUES(3982,"Coral Springs","USA","Florida",117549);
INSERT INTO CITY VALUES(4054,"Fairfield","USA","California",92256);
INSERT INTO CITY VALUES(4058,"Boulder","USA","Colorado",91238);
INSERT INTO CITY VALUES(4061,"Fall River","USA","Massachusetts",90555);
```

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

Solution:

SELECT * FROM CITY WHERE COUNTRYCODE = "USA" AND POPULATION > 100000;

Data Output:

```
SELECT * FROM CITY WHERE COUNTRYCODE = "USA" AND POPULATION > 100000
```

Input to filter result Free 1 Cost: 6ms

1 Total 5

	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	3878	Scottsdale	USA	Arizona	202705
2	3965	Corona	USA	California	124966
3	3973	Concord	USA	California	121780
4	3977	Cedar Rapids	USA	Iowa	120758
5	3982	Coral Springs	USA	Florida	117549

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

Solution:

SELECT NAME FROM CITY WHERE COUNTRYCODE = "USA" AND POPULATION > 120000;

Data Output:

```
# 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 COUNTRYCODE = "USA" AND POPULATION > 120000
```

Input to filter result Free 1 Cost: 8ms

1 Total 4

	NAME varchar
1	Scottsdale
2	Corona
3	Concord
4	Cedar Rapids

Q3. Query all columns (attributes) for every row in the CITY table.
Solution:
SELECT * FROM CITY;

Data Output:

```
# Q3. Query all columns (attributes) for every row in the CITY table.
SELECT * FROM CITY
```

Input to filter result Free Cost: 5ms

1 Total 9

	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	6	Rotterdam	NLD	Zuid-Holland	593321
2	3878	Scottsdale	USA	Arizona	202705
3	3965	Corona	USA	California	124966
4	3973	Concord	USA	California	121780
5	3977	Cedar Rapids	USA	Iowa	120758
6	3982	Coral Springs	USA	Florida	117549
7	4054	Fairfield	USA	California	92256
8	4058	Boulder	USA	Colorado	91238
9	4061	Fall River	USA	Massachusetts	90555

Q4. Query all columns for a city in CITY with the ID 1661(corrected to 4061).

Solution:
SELECT * FROM CITY WHERE ID = 4061;

Data Output:

```
# Q4. Query all columns for a city in CITY with the ID 1661(corrected to
4061).
SELECT * FROM CITY WHERE ID = 4061
```

Input to filter result Free Cost: 8ms

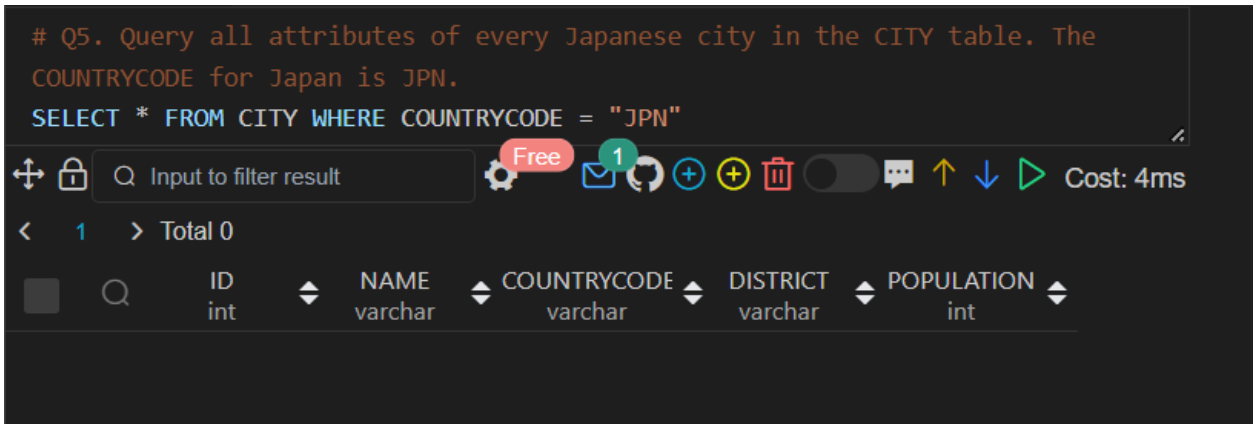
1 Total 1

	ID int	NAME varchar	COUNTRYCODE varchar	DISTRICT varchar	POPULATION int
1	4061	Fall River	USA	Massachusetts	90555

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

Solution:
SELECT * FROM CITY WHERE COUNTRYCODE = "JPN";

Data Output:



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

Solution:
SELECT NAME FROM CITY WHERE COUNTRYCODE = "JPN";

Data Output:

For Sample Dataset 2

```
-- CREATE TABLE STATION for sample dataset 2
CREATE TABLE STATION
(
   ID INT,
   CITY VARCHAR(21),
   STATE VARCHAR(2),
   LAT_N INT,
   LONG_W INT
);

-- Insert records into table STATION
INSERT INTO STATION VALUES (794,"Kissee Mills","MO",139,3);
INSERT INTO STATION VALUES (824,"Loma Mar","CA",48,130);
INSERT INTO STATION VALUES (603,"Sandy Hook","CT",72,148);
INSERT INTO STATION VALUES (478,"Tipton","IN",33,97);
INSERT INTO STATION VALUES (619,"Arlington","CO",75,92);
INSERT INTO STATION VALUES (711,"Turner","AR",50,101);
INSERT INTO STATION VALUES (839,"Slidell","LA",85,151);
INSERT INTO STATION VALUES (411,"Negreet","LA",98,105);
INSERT INTO STATION VALUES (588,"Glencoe","KY",46,136);
INSERT INTO STATION VALUES (665,"Chelsea","IA",98,59);
INSERT INTO STATION VALUES (342,"Chignik Lagoon","AK",103,153);
INSERT INTO STATION VALUES (733,"Pelahatchie","MS",38,28);
INSERT INTO STATION VALUES (441,"Hanna City","IL",50,136);
INSERT INTO STATION VALUES (811,"Dorrance","KS",102,121);
INSERT INTO STATION VALUES (698,"Albany","CA",49,80);
INSERT INTO STATION VALUES (325,"Monument","KS",70,141);
INSERT INTO STATION VALUES (414,"Manchester","MD",73,37);
INSERT INTO STATION VALUES (113,"Prescott","IA",39,65);
INSERT INTO STATION VALUES (971,"Graettinger","IA",94,150);
INSERT INTO STATION VALUES (266,"Cahone","CO",116,127);
```

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

Solution:

SELECT CITY, STATE FROM STATION;

Data Output:

```
# Q7. Query a list of CITY and STATE from the STATION table.
SELECT CITY, STATE FROM STATION
```

Input to filter result | Free | Cost: 5ms | 1 | Total 19

	CITY varchar	STATE varchar
1	Kissee Mills	MO
2	Loma Mar	CA
3	Sandy Hook	CT
4	Tipton	IN
5	Arlington	CO
6	Turner	AR
7	Slidell	LA
8	Glencoe	KY
9	Chelsea	IA
10	Chignik Lagoon	AK
11	Pelahatchie	MS
12	Hanna City	IL
13	Dorrance	KS
14	Albany	CA
15	Monument	KS
16	Manchester	MD
17	Prescott	IA
18	Graettinger	IA
19	Cahone	CO

-- 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.

Solution:

SELECT DISTINCT CITY FROM STATION WHERE ID % 2 = 0;

Data Output:

```
# 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 ID % 2 = 0
```

Input to filter result | Free | Cost: 2ms | 1 | Total 8

	CITY varchar
1	Kissee Mills
2	Loma Mar
3	Tipton
4	Glencoe
5	Chignik Lagoon
6	Albany
7	Manchester
8	Cahone

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

Solution:

SELECT COUNT(CITY) - COUNT(DISTINCT CITY) FROM STATION;

Data Output:

```
# 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
```

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

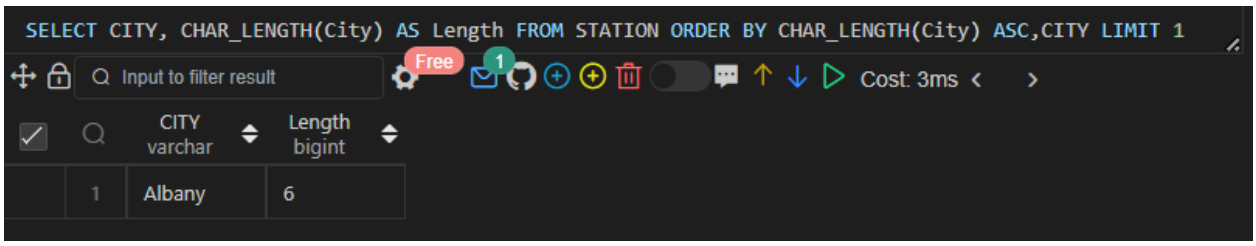
	COUNT(CITY) - COUNT(DISTINCT CITY) bigint
1	0

-- 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.

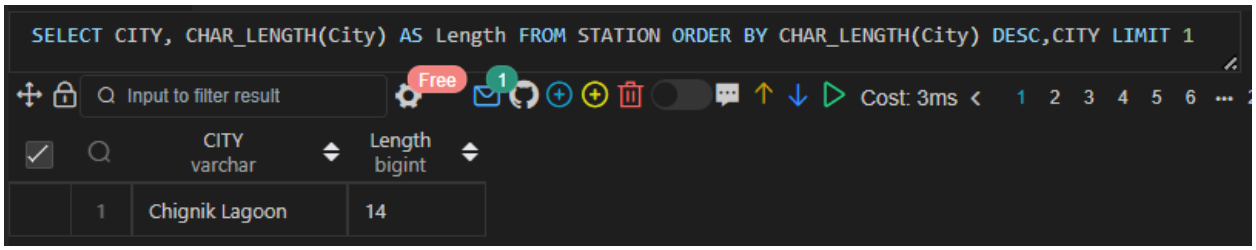
Solution:

SELECT CITY, CHAR_LENGTH(City) AS Length FROM STATION ORDER BY CHAR_LENGTH(City) ASC,CITY LIMIT 1;
SELECT CITY, CHAR_LENGTH(City) AS Length FROM STATION ORDER BY CHAR_LENGTH(City) DESC,CITY LIMIT 1;

Data Output:
1st query result:



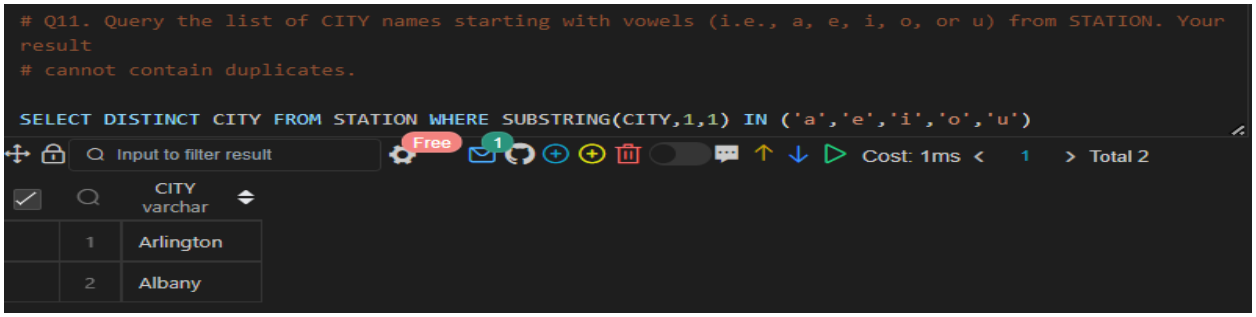
2nd query result:



-- 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.

Solution:
SELECT DISTINCT CITY FROM STATION WHERE SUBSTRING(CITY,1,1) IN ('a','e','i','o','u');

Data Output:



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

Solution:

SELECT DISTINCT CITY FROM STATION WHERE SUBSTRING(CITY,-1,1) IN ('a','e','i','o','u');

Data Output:

```
# 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 SUBSTRING(CITY,-1,1) IN ('a','e','i','o','u')
```

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

	CITY varchar
1	Glencoe
2	Chelsea
3	Pelahatchie
4	Dorrance
5	Cahone

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

Solution:

```
SELECT DISTINCT CITY FROM STATION WHERE SUBSTRING(CITY,1,1) NOT IN ('a','e','i','o','u');
```

Data Output:

```
# 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 SUBSTRING(CITY,1,1) NOT IN ('a','e','i','o','u')
```

Input to filter result | Free | 1 | Cost: 2ms | 1 | Total 18

	CITY varchar
1	Kissee Mills
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Turner
6	Slidell
7	Negreet
8	Glencoe
9	Chelsea
10	Chignik Lagoon
11	Pelahatchie
12	Hanna City
13	Dorrance
14	Monument
15	Manchester
16	Prescott
17	Graettinger
18	Cahone

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

Solution:

```
SELECT DISTINCT CITY FROM STATION WHERE SUBSTRING(CITY,-1,1) NOT IN ('a','e','i','o','u');
```

Data Output:

```
# 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 SUBSTRING(CITY,-1,1) NOT IN ('a','e','i','o','u')
```

Input to filter result | Free | 1 | Cost: 2ms | 1 | Total 15

	CITY varchar
1	Kissee Mills
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Arlington
6	Turner
7	Slidell
8	Negreet
9	Chignik Lagoon
10	Hanna City
11	Albany
12	Monument
13	Manchester
14	Prescott
15	Graettinger

-- 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.

Solution:

SELECT DISTINCT CITY FROM STATION WHERE SUBSTRING(CITY,-1,1) NOT IN ('a','e','i','o','u') OR SUBSTRING(CITY,1,1) NOT IN ('a','e','i','o','u');

Data Output:

```
# 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 SUBSTRING(CITY,-1,1) NOT IN ('a','e','i','o','u') OR
SUBSTRING(CITY,1,1) NOT IN ('a','e','i','o','u')
```

Input to filter result | Free | Cost: 3ms | 1 | Total 20

	CITY varchar
1	Kissee Mills
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Arlington
6	Turner
7	Slidell
8	Negreet
9	Glencoe
10	Chelsea
11	Chignik Lagoon
12	Pelahatchie
13	Hanna City
14	Dorrance
15	Albany
16	Monument
17	Manchester
18	Prescott
19	Graettinger
20	Cahone

-- 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.

Solution:

SELECT DISTINCT CITY FROM STATION WHERE SUBSTRING(CITY,-1,1) NOT IN ('a','e','i','o','u') AND SUBSTRING(CITY,1,1) NOT IN ('a','e','i','o','u');

Data Output:

```
# 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 SUBSTRING(CITY,-1,1) NOT IN ('a','e','i','o','u') AND
SUBSTRING(CITY,1,1) NOT IN ('a','e','i','o','u')
```

Input to filter result | Free | Cost: 4ms | 1 | Total 13

	CITY varchar
1	Kissee Mills
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Turner
6	Slidell
7	Negreet
8	Chignik Lagoon
9	Hanna City
10	Monument
11	Manchester
12	Prescott
13	Graettinger

Q.17 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.

Solution:

```
-- CREATE TABLE Product
CREATE TABLE Product
(
   product_id INT AUTO_INCREMENT PRIMARY KEY,
   product_name VARCHAR(10),
   unit_price INT
);

-- INSERT Records
INSERT INTO Product (product_name,unit_price) VALUES ('S8',1000);
INSERT INTO Product (product_name,unit_price) VALUES ('G4',800);
INSERT INTO Product (product_name,unit_price) VALUES ('iPhone',1400);

-- CREATE TABLE Sales
CREATE TABLE Sales
(
   seller_id INT,
   product_id INT,
   buyer_id INT,
   sale_date Date,
   quantity INT,
   price INT,
   FOREIGN KEY(product_id) REFERENCES Product(product_id)
);

-- INSERT Records
INSERT INTO Sales VALUES (1,1,1,'2019-01-21',2,2000);
INSERT INTO Sales VALUES (1,2,2,'2019-02-17',1,800);
INSERT INTO Sales VALUES (2,2,3,'2019-06-02',1,800);
INSERT INTO Sales VALUES (3,3,4,'2019-05-13',2,2800);

-- Final query
SELECT product_id, product_name FROM Product WHERE product_id IN (
   SELECT product_id FROM Sales GROUP BY product_id HAVING MIN(sale_date) >=
'2019-01-01' AND MAX(sale_date) <= '2019-03-31'
);
```

```
#
SELECT product_id, product_name FROM Product WHERE product_id IN (
    SELECT product_id FROM Sales GROUP BY product_id HAVING MIN(sale_date) >= '2019-01-01' AND
MAX(sale_date) <= '2019-03-31'

)
```

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

	product_id int	product_name varchar
1	1	S8

Q.18 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.

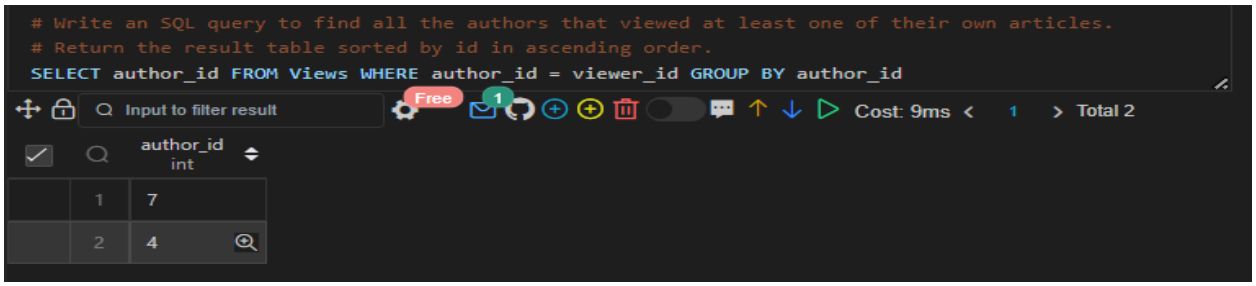
Solution:

```
-- CREATE TABLE Views
CREATE TABLE Views
(
   article_id INT,
   author_id INT,
   viewer_id INT,
   view_date Date
);

-- INSERT Records
INSERT INTO Views VALUES (1,3,5,'2019-08-01');
INSERT INTO Views VALUES (1,3,6,'2019-08-02');
INSERT INTO Views VALUES (2,7,7,'2019-08-01');
INSERT INTO Views VALUES (2,7,6,'2019-08-02');
INSERT INTO Views VALUES (4,7,1,'2019-07-22');
INSERT INTO Views VALUES (3,4,4,'2019-07-21');
INSERT INTO Views VALUES (3,4,4,'2019-07-21');
```

-- Final query:
SELECT author_id FROM Views WHERE author_id = viewer_id GROUP BY author_id;

Data Output;



Q.19 If the customer's preferred delivery date is the same as the order date, then the order is called immediately; otherwise, it is called scheduled. Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

Solution:
-- CREATE TABLE Delivery
CREATE TABLE Delivery
(
delivery_id INT PRIMARY KEY,
customer_id INT,
order_date DATE,
customer_pref_delivery_date DATE
);

-- INSERT Records
INSERT INTO Delivery (delivery_id,customer_id,order_date,customer_pref_delivery_date) VALUES(1,1,'2019-08-01','2019-08-02');
INSERT INTO Delivery (delivery_id,customer_id,order_date,customer_pref_delivery_date) VALUES(2,5,'2019-08-02','2019-08-02');
INSERT INTO Delivery (delivery_id,customer_id,order_date,customer_pref_delivery_date) VALUES(3,1,'2019-08-11','2019-08-11');
INSERT INTO Delivery (delivery_id,customer_id,order_date,customer_pref_delivery_date) VALUES(4,3,'2019-08-24','2019-08-26');
INSERT INTO Delivery (delivery_id,customer_id,order_date,customer_pref_delivery_date) VALUES(5,4,'2019-08-21','2019-08-22');
INSERT INTO Delivery (delivery_id,customer_id,order_date,customer_pref_delivery_date) VALUES(6,2,'2019-08-11','2019-08-13');

--Final Query
SELECT ROUND(COUNT(*)/(SELECT COUNT(*) FROM Delivery)*100,2) AS immediate_percentage FROM Delivery WHERE order_date = customer_pref_delivery_date;

Data Output:

```
SELECT ROUND(COUNT(*)/(SELECT COUNT(*) FROM Delivery)*100,2) AS immediate_percentage FROM Delivery
WHERE order_date = customer_pref_delivery_date
```

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

	immediate_percentage newdecimal
1	33.33

Q.20 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.
Explanation to calculate ctr:
for ad_id = 1, ctr = (2/(2+1)) * 100 = 66.67
for ad_id = 2, ctr = (1/(1+2)) * 100 = 33.33
for ad_id = 3, ctr = (1/(1+1)) * 100 = 50.00
for ad_id = 5, ctr = 0.00, Note that ad_id = 5 has no clicks or views.
Note that we do not care about Ignored Ads.

Solution:

```
-- CREATE TABLE Ads
CREATE TABLE Ads
(
   ad_id INT,
   user_id INT,
   `action` ENUM ('Clicked', 'Viewed', 'Ignored'),
   PRIMARY KEY (ad_id,user_id)
);

-- INSERT Records
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (1,1,'Clicked');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (2,2,'Clicked');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (3,3,'Viewed');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (5,5,'Ignored');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (1,7,'Ignored');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (2,7,'Viewed');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (3,5,'Clicked');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (1,4,'Viewed');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (2,11,'Viewed');
INSERT INTO Ads (ad_id,user_id,`action`) VALUES (1,2,'Clicked');
```

--Final query:

```
SELECT ad_id,ctr FROM (
SELECT ad_id, COALESCE(ROUND(SUM(`action` = 'Clicked') OVER(PARTITION BY ad_id)/
(SUM(`action` = 'Clicked') OVER(PARTITION BY ad_id) +
SUM(`action` = 'Viewed') OVER(PARTITION BY ad_id))*100,2),0) AS ctr FROM Ads) T GROUP
BY ad_id,ctr ORDER BY ctr DESC,ad_id ASC;
```

Data Output:

```
SELECT ad_id,ctr FROM (
SELECT ad_id, COALESCE(ROUND(SUM(`action` = 'Clicked') OVER(PARTITION BY ad_id)/
(SUM(`action` = 'Clicked') OVER(PARTITION BY ad_id) +
SUM(`action` = 'Viewed') OVER(PARTITION BY ad_id))*100,2),0) AS ctr FROM Ads) T GROUP BY ad_id,ctr
ORDER BY ctr DESC,ad_id ASC
```

Input to filter result | Free | Cost: 6ms | 1 | Total 4

	ad_id int	ctr newdecimal
1	1	66.67
2	3	50.00
3	2	33.33
4	5	0.00

Q.21 Write an SQL query to find the team size of each of the employees. Return result table in any order.

Solution:

```
-- CREATE TABLE Employee
CREATE TABLE Employee
(
   employee_id INT PRIMARY KEY,
   team_id INT
);

-- INSERT Records
INSERT INTO Employee VALUES(1,8);
INSERT INTO Employee VALUES(2,8);
INSERT INTO Employee VALUES(3,8);
INSERT INTO Employee VALUES(4,7);
INSERT INTO Employee VALUES(5,9);
INSERT INTO Employee VALUES(6,9);
```

--Final query:

```
SELECT employee_id, COUNT(*) OVER (PARTITION BY team_id) AS team_size FROM Employee;
```

Data Output:

```
SELECT employee_id, COUNT(*) OVER (PARTITION BY team_id) AS team_size FROM Employee
```

Input to filter result Free 1 Cost: 9ms 1 Total 6

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

Q.22 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.

Return result table in any order.

Solution:

```
-- CREATE TABLE Countries
CREATE TABLE Countries
(
   country_id INT PRIMARY KEY,
   country_name VARCHAR(20)
);

-- INSERT Records
INSERT INTO Countries VALUES(2,'USA');
INSERT INTO Countries VALUES(3,'Australia');
INSERT INTO Countries VALUES(7,'Peru');
INSERT INTO Countries VALUES(5,'China');
INSERT INTO Countries VALUES(8,'Morocco');
INSERT INTO Countries VALUES(9,'Spain');
```

```sql
-- CREATE TABLE Weather
CREATE TABLE Weather
(
   country_id INT,
   weather_state INT,
   day Date,
   PRIMARY KEY(country_id,day)
);

-- INSERT Records
INSERT INTO Weather VALUES(2,15,'2019-11-01');
INSERT INTO Weather VALUES(2,12,'2019-10-28');
INSERT INTO Weather VALUES(2,12,'2019-10-27');
INSERT INTO Weather VALUES(3,-2,'2019-11-10');
INSERT INTO Weather VALUES(3,0,'2019-11-11');
INSERT INTO Weather VALUES(3,3,'2019-11-12');
INSERT INTO Weather VALUES(5,16,'2019-11-07');
INSERT INTO Weather VALUES(5,18,'2019-11-09');
INSERT INTO Weather VALUES(5,21,'2019-11-23');
INSERT INTO Weather VALUES(7,25,'2019-11-28');
INSERT INTO Weather VALUES(7,22,'2019-12-01');
INSERT INTO Weather VALUES(7,20,'2019-12-02');
INSERT INTO Weather VALUES(8,25,'2019-11-05');
INSERT INTO Weather VALUES(8,27,'2019-11-15');
INSERT INTO Weather VALUES(8,31,'2019-11-25');
INSERT INTO Weather VALUES(9,7,'2019-10-23');
INSERT INTO Weather VALUES(9,3,'2019-12-23');

--Final query:
SELECT C.country_name,
CASE
WHEN AVG(W.weather_state) <= 15 THEN 'Cold'
WHEN AVG(W.weather_state) >= 25 THEN 'Hot'
ELSE 'Warm'
END AS weather_type
FROM Countries C INNER JOIN Weather W ON C.country_id = W.country_id AND
MONTHNAME(W.day) = 'November'
GROUP BY C.country_name;
```

Data Output:

```
SELECT C.country_name,
CASE
WHEN AVG(W.weather_state) <= 15 THEN 'Cold'
WHEN AVG(W.weather_state) >= 25 THEN 'Hot'
ELSE 'Warm'
END AS weather_type
FROM Countries C INNER JOIN Weather W ON C.country_id = W.country_id AND MONTHNAME(W.day) =
'November'
GROUP BY C.country_name
```

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

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

Q.23 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.

Solution:

```
-- CREATE TABLE Prices
CREATE TABLE Prices
(
   product_id INT,
   `start_date` DATE,
   end_date DATE,
   price INT,
   PRIMARY KEY(product_id,`start_date`,end_date)
);

-- INSERT Records
INSERT INTO Prices VALUES(1,'2019-02-17','2019-02-28',5);
INSERT INTO Prices VALUES(1,'2019-03-01','2019-03-22',20);
INSERT INTO Prices VALUES(2,'2019-02-01','2019-02-20',15);
INSERT INTO Prices VALUES(2,'2019-02-21','2019-03-31',30);
```

-- CREATE TABLE UnitsSold

```
CREATE TABLE UnitsSold
(
   product_id INT,
   purchase_date DATE,
   units INT
);
```

-- INSERT Records

```
INSERT INTO UnitsSold VALUES(1,'2019-02-25',100);
INSERT INTO UnitsSold VALUES(1,'2019-03-01',15);
INSERT INTO UnitsSold VALUES(2,'2019-02-10',200);
INSERT INTO UnitsSold VALUES(2,'2019-03-22',30);
```

--Final query:

```
SELECT P.product_id,
ROUND(SUM(P.price*U.units)/SUM(U.units),2) AS average_price
FROM Prices P INNER JOIN UnitsSold U ON P.product_id = U.product_id AND
U.purchase_date BETWEEN P.start_date AND P.end_date GROUP BY P.product_id;
```

Data Output:

```
SELECT P.product_id,
ROUND(SUM(P.price*U.units)/SUM(U.units),2) AS average_price
FROM Prices P INNER JOIN UnitsSold U ON P.product_id = U.product_id AND U.purchase_date BETWEEN
P.start_date AND P.end_date GROUP BY P.product_id
```

Input to filter result | Free | 1 | Cost: 3ms | 1 | Total 2

	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. Return the result table in any order.

Solution:

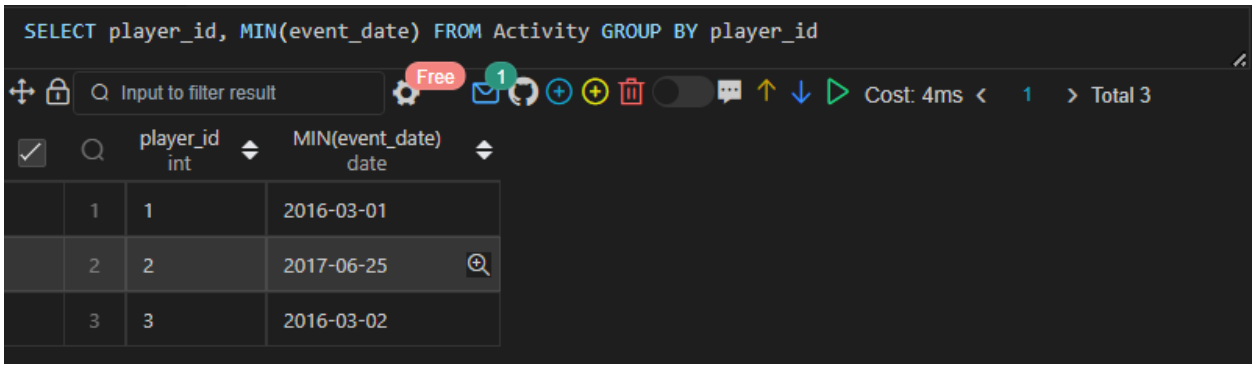
-- CREATE TABLE Activity

```
CREATE TABLE Activity
(
   player_id INT,
   device_id INT,
   event_date DATE,
   games_played INT,
   PRIMARY KEY (player_id,event_date)
);
```

-- Insert Records
```
INSERT INTO Activity VALUES(1,2,'2016-03-01',5);
INSERT INTO Activity VALUES(1,2,'2016-05-02',6);
INSERT INTO Activity VALUES(2,3,'2017-06-25',1);
INSERT INTO Activity VALUES(3,1,'2016-03-02',0);
INSERT INTO Activity VALUES(3,4,'2018-07-03',5);
```

--Final query:
```
SELECT player_id, MIN(event_date) FROM Activity GROUP BY player_id;
```

Data Output:



Q24. Write an SQL query to report the first login date for each player. Return the result table in any order.

Solution:
-- CREATE TABLE Activity
```
CREATE TABLE Activity
(
   player_id INT,
   device_id INT,
   event_date DATE,
   games_played INT,
   PRIMARY KEY (player_id,event_date)
);
```

-- Insert Records
```
INSERT INTO Activity VALUES(1,2,'2016-03-01',5);
INSERT INTO Activity VALUES(1,2,'2016-05-02',6);
INSERT INTO Activity VALUES(2,3,'2017-06-25',1);
INSERT INTO Activity VALUES(3,1,'2016-03-02',0);
INSERT INTO Activity VALUES(3,4,'2018-07-03',5);
```

--Final query:
SELECT player_id, MIN(event_date) FROM Activity GROUP BY player_id;

Data Output:

```
SELECT player_id, MIN(event_date) FROM Activity GROUP BY player_id
```

Input to filter result | Free | Cost: 8ms | 1 | Total 3

	player_id int	MIN(event_date) 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.

Solution:
SELECT player_id, device_id FROM Activity WHERE event_date IN (SELECT MIN(event_date) OVER (PARTITION BY player_id) FROM Activity);

Data Output:

```
SELECT player_id, device_id FROM Activity WHERE event_date IN (SELECT MIN(event_date) OVER (PARTITION
BY player_id) FROM Activity)
```

Input to filter result | Free | Cost: 2ms | 1 | Total 3

	player_id int	device_id int
1	1	2
2	2	3
3	3	1

Q26. 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.
Solution:
--CREATE TABLE Products

```
CREATE TABLE Products
(
    product_id INT PRIMARY KEY,
    product_name VARCHAR(50),
    product_category VARCHAR(20)

);
```

```
--INSERT Records
INSERT INTO Products VALUES(1,'Leetcode Solutions','Book');
INSERT INTO Products VALUES(2,'Jewels of Stringology','Book');
INSERT INTO Products VALUES(3,'HP','Laptop');
INSERT INTO Products VALUES(4,'Lenovo','Laptop');
INSERT INTO Products VALUES(5,'Leetcode Kit','T-shirt');

-- CREATE TABLE Orders
CREATE TABLE Orders
(
   product_id INT,
   order_date DATE,
   unit INT,
   FOREIGN KEY (product_id) REFERENCES Products(product_id)
);

--Insert Records
INSERT INTO Orders VALUES(1,'2020-02-05',60);
INSERT INTO Orders VALUES(1,'2020-02-10',70);
INSERT INTO Orders VALUES(2,'2020-01-18',30);
INSERT INTO Orders VALUES(2,'2020-02-11',80);
INSERT INTO Orders VALUES(3,'2020-02-17',2);
INSERT INTO Orders VALUES(3,'2020-02-24',3);
INSERT INTO Orders VALUES(4,'2020-03-01',20);
INSERT INTO Orders VALUES(4,'2020-03-04',30);
INSERT INTO Orders VALUES(4,'2020-03-04',60);
INSERT INTO Orders VALUES(5,'2020-02-25',50);
INSERT INTO Orders VALUES(5,'2020-02-27',50);
INSERT INTO Orders VALUES(5,'2020-03-01',50);

--Final query:
SELECT P.product_name,SUM(O.unit) AS unit FROM Products P INNER JOIN Orders O
ON P.product_id = O.product_id AND MONTHNAME(O.order_date) = 'February' GROUP BY
P.product_name;
```

Data Output:

```
SELECT P.product_name,SUM(O.unit) AS unit FROM Products P INNER JOIN Orders O ON P.product_id =
O.product_id AND MONTHNAME(O.order_date) = 'February' GROUP BY P.product_name
```

Input to filter result | Free | 1 | Cost: 6ms | 1 | Total 4

	product_name varchar	unit newdecimal
1	Leetcode Solutions	130
2	Jewels of Stringology	80
3	HP	5
4	Leetcode Kit	100

Q27. 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'.

Return the result table in any order.

Solution:

```
--CREATE TABLE Users
CREATE TABLE Users
(
   user_id INT PRIMARY KEY,
   `name` VARCHAR(15),
   mail VARCHAR(30)
);

-- INSERT Records
INSERT INTO Users VALUES(1,'Winston','winston@leetcode.com');
INSERT INTO Users VALUES(2,'Jonathan','jonathanisgreat');
INSERT INTO Users VALUES(3,'Annabelle','bella-@leetcode.com');
INSERT INTO Users VALUES(4,'Sally','sally.come@leetcode.com');
INSERT INTO Users VALUES(5,'Marwan','quarz#2020@leetcode.com');
INSERT INTO Users VALUES(6,'David','david69@gmail.com');
INSERT INTO Users VALUES(7,'Shapiro','.shapo@leetcode.com');


--Final query:
SELECT * FROM Users WHERE mail NOT LIKE '.%' AND mail NOT LIKE '%#%' AND mail
LIKE '%@leetcode.com';
```

Data Output:

```
SELECT * FROM Users WHERE mail NOT LIKE '.%' AND mail NOT LIKE '%#%' AND mail LIKE '%@leetcode.com'
```

Input to filter result Free 1 Cost: 7ms 1 Total 3

	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.co

Q.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.

Solution:

```
-- CREATE TABLE Customers
CREATE TABLE Customers
(
   customer_id INT PRIMARY KEY,
   `name` VARCHAR(15),
   country VARCHAR(10)

);

-- Insert Records
INSERT INTO Customers VALUES(1,'Winston','USA');
INSERT INTO Customers VALUES(2,'Jonathan','Peru');
INSERT INTO Customers VALUES(3,'Moustafa','Egypt');

-- CREATE TABLE Product
CREATE TABLE Product
(
   product_id INT PRIMARY KEY,
   `description` VARCHAR(15),
   price INT
);

-- INSERT Records
INSERT INTO Product VALUES(10,'LC Phone',300);
INSERT INTO Product VALUES(20,'LC T-Shirt',10);
INSERT INTO Product VALUES(30,'LC Book',45);
INSERT INTO Product VALUES(40,'LC Keychain',2);
```

```sql
-- CREATE TABLE Orders
CREATE TABLE Orders
(
   order_id INT PRIMARY KEY,
   customer_id INT,
   product_id INT,
   order_date DATE,
   quantity INT
);

-- Insert Records
INSERT INTO Orders VALUES(1,1,10,'2020-06-10',1);
INSERT INTO Orders VALUES(2,1,20,'2020-07-01',1);
INSERT INTO Orders VALUES(3,1,30,'2020-07-08',2);
INSERT INTO Orders VALUES(4,2,10,'2020-06-15',2);
INSERT INTO Orders VALUES(5,2,40,'2020-07-01',10);
INSERT INTO Orders VALUES(6,3,20,'2020-06-24',2);
INSERT INTO Orders VALUES(7,3,30,'2020-06-25',2);
INSERT INTO Orders VALUES(9,3,30,'2020-05-08',3);

--Final query:

WITH CTE AS (
   SELECT O.customer_id FROM Orders O INNER JOIN Product P
ON O.product_id = P.product_id WHERE MONTHNAME(order_date) IN ('June','July') AND
(O.quantity*P.price) > 100
)
SELECT C.customer_id,`name` FROM Customers C INNER JOIN CTE Ct ON C.customer_id =
Ct.customer_id;
```

Data Output:

```sql
WITH CTE AS (
    SELECT O.customer_id FROM Orders O INNER JOIN Product P
ON O.product_id = P.product_id WHERE MONTHNAME(order_date) IN ('June','July') AND
(O.quantity*P.price) > 100
)
SELECT C.customer_id,`name` FROM Customers C INNER JOIN CTE Ct ON C.customer_id = Ct.customer_id
```

Input to filter result | Free | 1 | Cost: 4ms | 1 | Total 2

	customer_id int	name varchar
1	1	Winston
2	2	Jonathan

Q.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.

Solution:

```
--CREATE TABLE TVProgram
CREATE TABLE TVProgram
(
   program_date DATE,
   content_id INT,
   channel VARCHAR(20),
   PRIMARY KEY(program_date,content_id)
);

--INSERT Records
INSERT INTO TVProgram VALUES('2020-06-10 08:00',1,'LC-Channel');
INSERT INTO TVProgram VALUES('2020-05-11 12:00',2,'LC-Channel');
INSERT INTO TVProgram VALUES('2020-05-12 12:00',3,'LC-Channel');
INSERT INTO TVProgram VALUES('2020-05-13 14:00',4,'Disney Ch');
INSERT INTO TVProgram VALUES('2020-06-18 14:00',4,'Disney Ch');
INSERT INTO TVProgram VALUES('2020-07-15 16:00',5,'Disney Ch');

-- CREATE TABLE Content
CREATE TABLE Content
(
   content_id INT PRIMARY KEY,
   title VARCHAR(25),
   kids_content ENUM('Y','N'),
   content_type VARCHAR(20)
);

-- INSERT Records
INSERT INTO Content VALUES(1,'Leetcode Movie','N','Movies');
INSERT INTO Content VALUES(2,'Alg. for Kids','Y','Series');
INSERT INTO Content VALUES(3,'Database Sols','N','Series');
INSERT INTO Content VALUES(4,'Aladdin','Y','Movies');
INSERT INTO Content VALUES(5,'Cinderella','Y','Movies');

--Final query:

SELECT title FROM Content WHERE kids_content = 'Y' AND content_id IN (SELECT
content_id FROM TVProgram WHERE program_date like '2020%' AND
MONTHNAME(program_date) = 'June');
```

Data Output:

```
SELECT title FROM Content WHERE kids_content = 'Y' AND content_id IN (SELECT content_id FROM
TVProgram WHERE program_date like '2020%' AND MONTHNAME(program_date) = 'June')
```

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

	title varchar
1	Aladdin

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

Solution:

```
-- CREATE TABLE NPV
CREATE TABLE NPV
(
   id INT,
   year INT,
   npv INT,
   PRIMARY KEY(id,year)
);

--Insert Records
INSERT INTO NPV VALUES(1,2018,100);
INSERT INTO NPV VALUES(7,2020,30);
INSERT INTO NPV VALUES(13,2019,40);
INSERT INTO NPV VALUES(1,2019,113);
INSERT INTO NPV VALUES(2,2008,121);
INSERT INTO NPV VALUES(3,2009,12);
INSERT INTO NPV VALUES(11,2020,99);
INSERT INTO NPV VALUES(7,2019,0);

-- CREATE TABLE Queries
CREATE TABLE Queries
(
   id INT,
   year INT,
   PRIMARY KEY(id,year)
);

-- Insert Records
INSERT INTO Queries VALUES(1,2019);
INSERT INTO Queries VALUES(2,2008);
```

```
INSERT INTO Queries VALUES(3,2009);
INSERT INTO Queries VALUES(7,2018);
INSERT INTO Queries VALUES(7,2019);
INSERT INTO Queries VALUES(7,2020);
INSERT INTO Queries VALUES(13,2019);
```

--Final query:

```
SELECT Q.*,COALESCE(N.npv,0) AS NPV FROM NPV N RIGHT OUTER JOIN Queries Q ON
N.id = Q.id AND N.year=Q.year;
```

Data Output:

```
SELECT Q.*,COALESCE(N.npv,0) AS NPV FROM NPV N RIGHT OUTER JOIN Queries Q ON N.id = Q.id AND
N.year=Q.year
```

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

Q.31 is a duplicate of Q.30.

Q.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. Return the result table in any order.

Solution:

-- CREATE TABLE Employees

```
CREATE TABLE Employees
(
   id INT PRIMARY KEY,
   `name` VARCHAR(15)
);
```

-- Insert Records

```
INSERT INTO Employees VALUES(1,'Alice');
INSERT INTO Employees VALUES(7,'Bob');
```

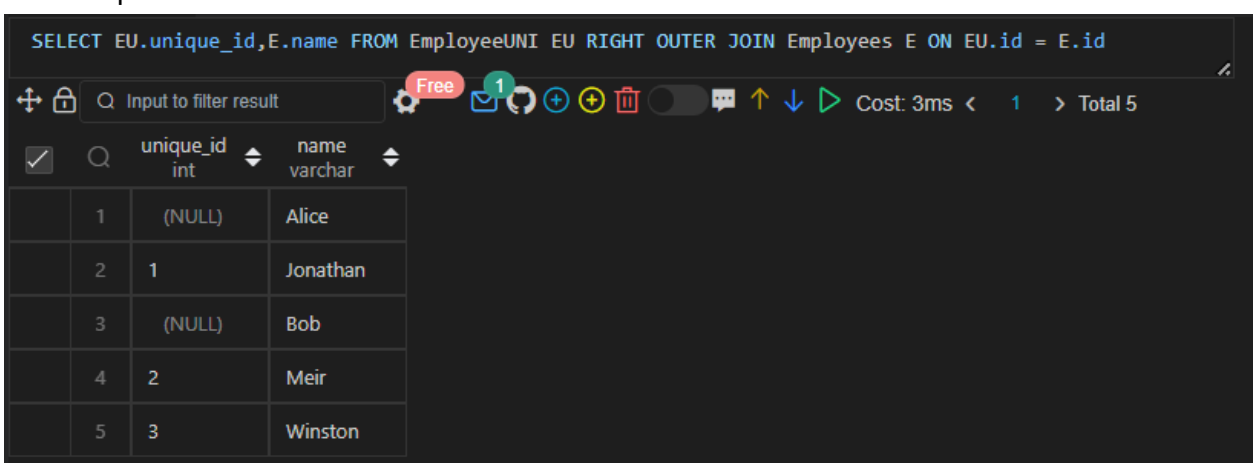
```
INSERT INTO Employees VALUES(11,'Meir');
INSERT INTO Employees VALUES(90,'Winston');
INSERT INTO Employees VALUES(3,'Jonathan');

-- CREATE TABLE EmployeeUNI
CREATE TABLE EmployeeUNI
(
   id INT,
   unique_id INT,
   PRIMARY KEY (id,unique_id)
);

-- INSERT Records
INSERT INTO EmployeeUNI VALUES(3,1);
INSERT INTO EmployeeUNI VALUES(11,2);
INSERT INTO EmployeeUNI VALUES(90,3);

--Final query:
SELECT EU.unique_id,E.name FROM EmployeeUNI EU RIGHT OUTER JOIN Employees E
ON EU.id = E.id;
```

Data Output:



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

Solution:

```
-- Create Table Users
CREATE TABLE Users
(
   id INT PRIMARY KEY,
   `name` VARCHAR(20)
);

-- Insert Records
INSERT INTO Users VALUES(1,'Alice');
INSERT INTO Users VALUES(2,'Bob');
INSERT INTO Users VALUES(3,'Alex');
INSERT INTO Users VALUES(4,'Donald');
INSERT INTO Users VALUES(7,'Lee');
INSERT INTO Users VALUES(13,'Jonathan');
INSERT INTO Users VALUES(19,'Elvis');

-- Create Table Rides
CREATE TABLE Rides
(
   id INT PRIMARY KEY,
   user_id INT,
   distance INT
);

-- Insert Records
INSERT INTO Rides VALUES(1,1,120);
INSERT INTO Rides VALUES(2,2,317);
INSERT INTO Rides VALUES(3,3,222);
INSERT INTO Rides VALUES(4,7,100);
INSERT INTO Rides VALUES(5,13,312);
INSERT INTO Rides VALUES(6,19,50);
INSERT INTO Rides VALUES(7,7,120);
INSERT INTO Rides VALUES(8,19,400);
INSERT INTO Rides VALUES(9,7,230);
```

--Final query:

```
SELECT U.name,COALESCE(SUM(R.distance),0) AS travelled_distance FROM Users U LEFT OUTER JOIN Rides R ON U.id = R.user_id
GROUP BY U.name ORDER BY SUM(R.distance) DESC, U.name ASC;
```

Data Output:

```
SELECT U.name,COALESCE(SUM(R.distance),0) AS travelled_distance FROM Users U LEFT OUTER JOIN Rides R
ON U.id = R.user_id GROUP BY U.name ORDER BY SUM(R.distance) DESC, U.name ASC
```

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

Q.34 is a duplicate of Q.26.

Q.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.

Solution:

```
-- Create Table Movies
CREATE TABLE Movies
(
   movie_id INT PRIMARY KEY,
   title VARCHAR(15)
);

-- Insert Records
INSERT INTO Movies VALUES(1,'Avengers');
INSERT INTO Movies VALUES(2,'Frozen 2');
INSERT INTO Movies VALUES(3,'Joker');
```

```sql
-- CREATE TABLE Users
CREATE TABLE Users
(
   user_id INT PRIMARY KEY,
   `name` VARCHAR(15)
);

-- Insert Records
INSERT INTO Users VALUES(1,'Daniel');
INSERT INTO Users VALUES(2,'Monica');
INSERT INTO Users VALUES(3,'Maria');
INSERT INTO Users VALUES(4,'James');

-- CREATE TABLE MovieRating
CREATE TABLE MovieRating
(
   movie_id INT,
   user_id INT,
   rating INT,
   created_at DATE,
   PRIMARY KEY(movie_id,user_id)
);

-- INSERT Records
INSERT INTO MovieRating VALUES(1,1,3,'2020-01-12');
INSERT INTO MovieRating VALUES(1,2,4,'2020-02-11');
INSERT INTO MovieRating VALUES(1,3,2,'2020-02-12');
INSERT INTO MovieRating VALUES(1,4,1,'2020-01-01');
INSERT INTO MovieRating VALUES(2,1,5,'2020-02-17');
INSERT INTO MovieRating VALUES(2,2,2,'2020-02-01');
INSERT INTO MovieRating VALUES(2,3,2,'2020-03-01');
INSERT INTO MovieRating VALUES(3,1,3,'2020-02-22');
INSERT INTO MovieRating VALUES(3,2,4,'2020-02-25');
```

--Final queries:

SELECT `name` FROM (
SELECT U.name, Count(*) FROM Users U INNER JOIN MovieRating MR ON U.user_id = MR.user_id GROUP BY U.name ORDER BY COUNT(*) DESC, U.name)t LIMIT 1;

Data Output:

```
SELECT `name` FROM (
SELECT U.name, Count(*) FROM Users U INNER JOIN MovieRating MR ON U.user_id = MR.user_id GROUP BY
U.name ORDER BY COUNT(*) DESC, U.name)t LIMIT 1
```

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

	name varchar
1	Daniel

SELECT title FROM (
SELECT M.title, AVG(rating) FROM MovieRating MR INNER JOIN Movies M ON M.movie_id = MR.movie_id WHERE MONTHNAME(created_at) = 'February'
GROUP BY M.title ORDER BY AVG(rating) DESC, M.title ASC)m LIMIT 1;

Data Output:

```
SELECT title FROM (
SELECT M.title, AVG(rating) FROM MovieRating MR INNER JOIN Movies M ON M.movie_id = MR.movie_id WHERE
MONTHNAME(created_at) = 'February'
GROUP BY M.title ORDER BY AVG(rating) DESC, M.title ASC)m LIMIT 1
```

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

	title varchar
1	Frozen 2

Q.36 is a duplicate of Q.33

Q.37 is a duplicate of Q.32

Q.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.

Solution:

```
-- CREATE TABLE Departments
CREATE TABLE Departments
(
   id INT PRIMARY KEY,
   `name` VARCHAR(50)
);

-- INSERT Records
INSERT INTO Departments VALUES(1,'Electrical Engineering');
INSERT INTO Departments VALUES(7,'Computer Engineering');
INSERT INTO Departments VALUES(13,'Business Administration');

-- CREATE TABLE Students
CREATE TABLE Students
(
   id INT PRIMARY KEY,
   `name` VARCHAR(20),
   department_id INT
);

-- INSERT Records
INSERT INTO Students VALUES(23,'Alice',1);
INSERT INTO Students VALUES(1,'Bob',7);
INSERT INTO Students VALUES(5,'Jennifer',13);
INSERT INTO Students VALUES(2,'John',14);
INSERT INTO Students VALUES(4,'Jasmine',77);
INSERT INTO Students VALUES(3,'Steve',74);
INSERT INTO Students VALUES(6,'Luis',1);
INSERT INTO Students VALUES(8,'Jonathan',7);
INSERT INTO Students VALUES(7,'Daiana',33);
INSERT INTO Students VALUES(11,'Madelynn',1);

--Final query:

SELECT id, `name` FROM Students WHERE department_id NOT IN (SELECT id FROM
Departments);
```

Data Output:

```
SELECT id, `name` FROM Students WHERE department_id NOT IN (SELECT id FROM Departments)
```

Input to filter result Free 1 Cost: 2ms 1 Total 4

	id int	name varchar
1	2	John
2	3	Steve
3	4	Jasmine
4	7	Daiana

Q.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.

Solution:

```
-- CREATE TABLE Calls
CREATE TABLE Calls
(
   from_id INT,
   to_id INT,
   duration INT
);

-- INSERT Records
INSERT INTO Calls VALUES(1,2,59);
INSERT INTO Calls VALUES(2,1,11);
INSERT INTO Calls VALUES(1,3,20);
INSERT INTO Calls VALUES(3,4,100);
INSERT INTO Calls VALUES(3,4,200);
INSERT INTO Calls VALUES(3,4,200);
INSERT INTO Calls VALUES(4,3,499);
```

--Final query:

```
WITH CTE AS(
SELECT person1,person2,duration FROM(
SELECT from_id AS person1, to_id AS person2, duration FROM Calls
UNION ALL
SELECT to_id AS person1, from_id AS person2, duration FROM Calls)temp WHERE person1 < person2)
SELECT person1, person2, COUNT(*) AS call_count, SUM(duration) AS total_duration FROM CTE GROUP BY person1, person2;
```

Data Output:

```
WITH CTE AS(
SELECT person1,person2,duration FROM(
SELECT from_id AS person1, to_id AS person2, duration FROM Calls
UNION ALL
SELECT to_id AS person1, from_id AS person2, duration FROM Calls)temp WHERE person1 < person2)
SELECT person1, person2, COUNT(*) AS call_count, SUM(duration) AS total_duration FROM CTE GROUP BY
person1, person2
```

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

	person1 int	person2 int	call_count bigint	total_duration newdecimal
1	1	2	2	70
2	1	3	1	20
3	3	4	4	999

Q.40 is a duplicate of Q.23.

Q.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.

Solution:

```
-- CREATE TABLE Warehouse
CREATE TABLE Warehouse
(
   `name` VARCHAR(10),
   product_id INT,
   units INT,
   PRIMARY KEY(`name`, product_id)
);
```

--Insert Records

```
INSERT INTO Warehouse VALUES('LCHouse1',1,1);
INSERT INTO Warehouse VALUES('LCHouse1',2,10);
INSERT INTO Warehouse VALUES('LCHouse1',3,5);
INSERT INTO Warehouse VALUES('LCHouse2',1,2);
INSERT INTO Warehouse VALUES('LCHouse2',2,2);
INSERT INTO Warehouse VALUES('LCHouse3',4,1);
```

-- CREATE TABLE Products

```
CREATE TABLE Products
(
   product_id INT PRIMARY KEY,
   product_name VARCHAR(20),
   Width INT,
   Length INT,
   Height INT
);
```

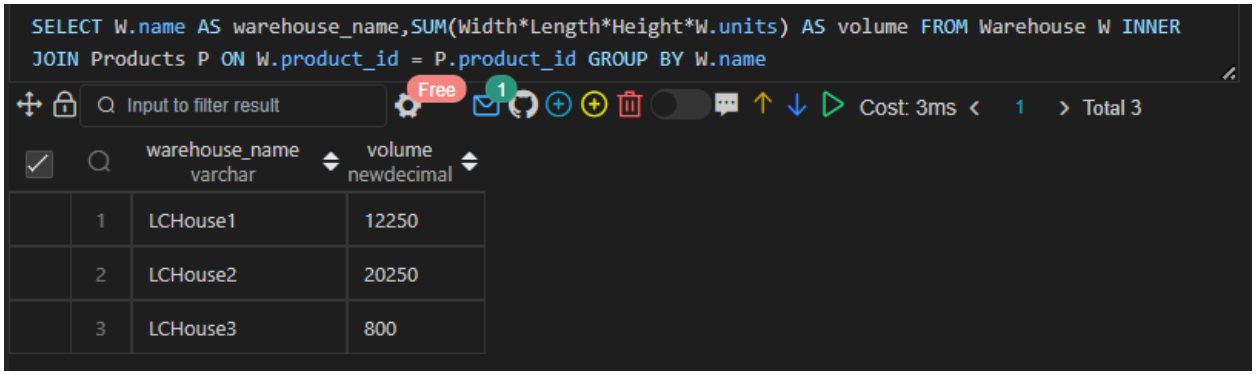
-- INSERT Records

```
INSERT INTO Products VALUES(1,'LC-TV',5,50,40);
INSERT INTO Products VALUES(2,'LC-KeyChain',5,5,5);
INSERT INTO Products VALUES(3,'LC-Phone',2,10,10);
INSERT INTO Products VALUES(4,'LC-T-Shirt',4,10,20);
```

--Final query:

```
SELECT W.name AS warehouse_name,SUM(Width*Length*Height*W.units) AS volume FROM
Warehouse W INNER JOIN Products P ON W.product_id = P.product_id GROUP BY W.name;
```

Data Output:



Q.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.

Solution:

```
-- CREATE Table Sales
CREATE TABLE Sales
(
   sale_date DATE,
   fruit ENUM('apples','oranges'),
   sold_num INT,
   PRIMARY KEY(sale_date,fruit)
);

-- INSERT Records
INSERT INTO Sales VALUES('2020-05-01','apples',10);
INSERT INTO Sales VALUES('2020-05-01','oranges',8);
INSERT INTO Sales VALUES('2020-05-02','apples',15);
INSERT INTO Sales VALUES('2020-05-02','oranges',15);
INSERT INTO Sales VALUES('2020-05-03','apples',20);
INSERT INTO Sales VALUES('2020-05-03','oranges',0);
INSERT INTO Sales VALUES('2020-05-04','apples',15);
INSERT INTO Sales VALUES('2020-05-04','oranges',16);

--Final query:

SELECT sale_date, SUM(IF(fruit='apples',sold_num,-sold_num)) AS diff FROM Sales GROUP BY sale_date;
```

Data Output:

```
SELECT sale_date, SUM(IF(fruit='apples',sold_num,-sold_num)) AS diff FROM Sales GROUP BY sale_date
```

Input to filter result | Free | 1 | Cost: 6ms | 1 | Total 4

	sale_date date	diff newdecimal
1	2020-05-01	2
2	2020-05-02	0
3	2020-05-03	20
4	2020-05-04	-1

Q.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.

Solution:

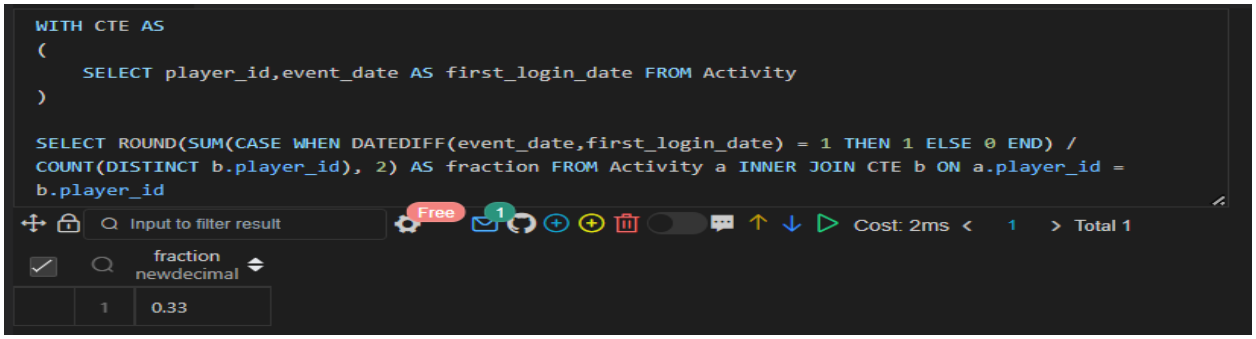
```
-- CREATE TABLE Activity
CREATE TABLE Activity
(
   player_id INT,
   device_id INT,
   event_date DATE,
   games_played INT,
   PRIMARY KEY (player_id,event_date)
);

-- Insert Records
INSERT INTO Activity VALUES(1,2,'2016-03-01',5);
INSERT INTO Activity VALUES(1,2,'2016-03-02',6);
INSERT INTO Activity VALUES(2,3,'2017-06-25',1);
INSERT INTO Activity VALUES(3,1,'2016-03-02',0);
INSERT INTO Activity VALUES(3,4,'2018-07-03',5);

--Final query:

WITH CTE AS
(
   SELECT player_id,event_date AS first_login_date FROM Activity
)
SELECT ROUND(SUM(CASE WHEN DATEDIFF(event_date,first_login_date) = 1 THEN 1
ELSE 0 END) / COUNT(DISTINCT b.player_id), 2) AS fraction
FROM Activity a INNER JOIN CTE b ON a.player_id = b.player_id;
```

Data Output;



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

Solution:

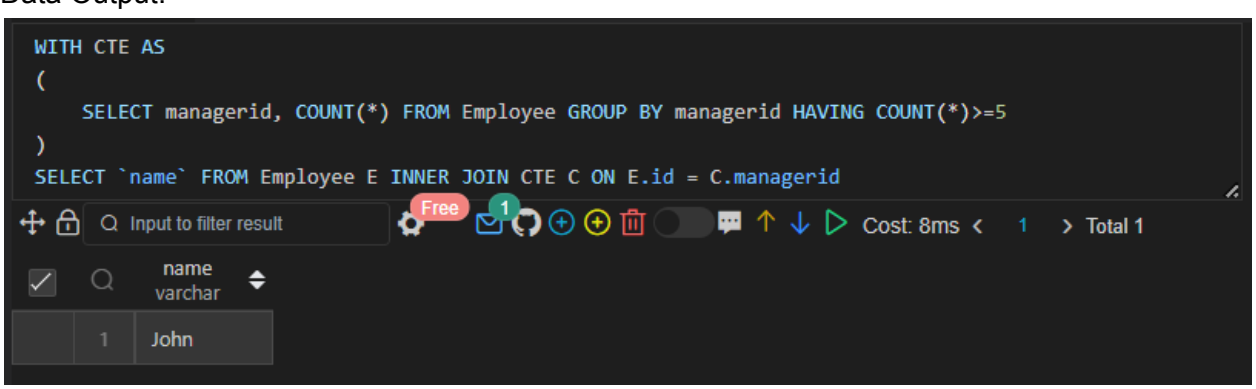
```
-- CREATE TABLE Employee
CREATE TABLE Employee
(
   id INT PRIMARY KEY,
   `name` VARCHAR(10),
   department VARCHAR(2),
   managerid INT
);

-- INSERT Records
INSERT INTO Employee VALUES(101,'John','A',NULL);
INSERT INTO Employee VALUES(102,'Dan','A',101);
INSERT INTO Employee VALUES(103,'James','A',101);
INSERT INTO Employee VALUES(104,'Amy','A',101);
INSERT INTO Employee VALUES(105,'Anne','A',101);
INSERT INTO Employee VALUES(106,'Ron','B',101);

--Final query:

WITH CTE AS
(
   SELECT managerid, COUNT(*) FROM Employee GROUP BY managerid HAVING
COUNT(*)>=5
)
SELECT `name` FROM Employee E INNER JOIN CTE C ON E.id = C.managerid ;
```

Data Output:



Q.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.

Solution:

```
-- CREATE TABLE Student
CREATE TABLE Student
(
   student_id INT PRIMARY KEY,
   student_name VARCHAR(10),
   gender VARCHAR(2),
   dept_id INT,
   FOREIGN KEY(dept_id) REFERENCES Department(dept_id)
);

-- INSERT Records
INSERT INTO Student VALUES(1,'Jack','M',1);
INSERT INTO Student VALUES(2,'Jane','F',1);
INSERT INTO Student VALUES(3,'Mark','M',2);

-- create table Department
CREATE TABLE Department
(
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(15)
);

-- Insert Records
INSERT INTO Department VALUES(1,'Engineering');
INSERT INTO Department VALUES(2,'Science');
INSERT INTO Department VALUES(3,'Law');

--Final query:

SELECT D.dept_name,COUNT(DISTINCT student_id) AS student_number FROM Department
D LEFT OUTER JOIN Student S ON D.dept_id = S.dept_id
GROUP BY D.dept_name ORDER BY COUNT(DISTINCT student_id) DESC, D.dept_name
ASC;
```

Data Output:

```
SELECT D.dept_name,COUNT(DISTINCT student_id) AS student_number FROM Department D LEFT OUTER JOIN
Student S ON D.dept_id = S.dept_id
GROUP BY D.dept_name ORDER BY COUNT(DISTINCT student_id) DESC, D.dept_name ASC
```

Input to filter result | Free | Cost: 4ms | 1 | Total 3

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

Q.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.

Solution:

```
-- CREATE TABLE Customer
CREATE TABLE Customer
(
   customer_id INT,
   product_key INT,
   FOREIGN KEY (product_key) REFERENCES Product(product_key)
);

-- INSERT Records
INSERT INTO Customer VALUES(1,5);
INSERT INTO Customer VALUES(2,6);
INSERT INTO Customer VALUES(3,5);
INSERT INTO Customer VALUES(3,6);
INSERT INTO Customer VALUES(1,6);

--Final query:
SELECT customer_id FROM Customer GROUP BY customer_id HAVING COUNT(DISTINCT
product_key) = (SELECT COUNT(*) FROM Product);
```

Data Output:

```
SELECT customer_id FROM Customer GROUP BY customer_id HAVING COUNT(DISTINCT product_key) = (SELECT
COUNT(*) FROM Product)
```

Input to filter result | Free | Cost: 6ms | 1 | Total 2

	customer_id int
1	1
2	3

Q.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.

Solution:

```
-- Create Tale Employee
CREATE TABLE Employee
(
   employee_id INT PRIMARY KEY,
   `name` VARCHAR(10),
   experience_years INT
);

-- Insert Records
INSERT INTO Employee VALUES(1,'Khaled',3);
INSERT INTO Employee VALUES(2,'Ali',2);
INSERT INTO Employee VALUES(3,'John',3);
INSERT INTO Employee VALUES(4,'Doe',2);

-- Create Table Project
CREATE TABLE Project
(
   project_id INT,
   employee_id INT,
   PRIMARY KEY (project_id,employee_id),
   FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);

-- Insert Records
INSERT INTO Project VALUES(1,1);
INSERT INTO Project VALUES(1,2);
INSERT INTO Project VALUES(1,3);
INSERT INTO Project VALUES(2,1);
INSERT INTO Project VALUES(2,4);

--Final query:

WITH CTE AS (
SELECT P.project_id,P.Employee_id, DENSE_RANK() OVER(PARTITION BY P.project_id
ORDER BY E.experience_years DESC) AS rn FROM Employee E INNER JOIN Project P
ON E.employee_id = P.employee_id)
SELECT project_id,employee_id FROM CTE WHERE rn = 1;
```

Data Output:

```
WITH CTE AS (
SELECT P.project_id,P.Employee_id, DENSE_RANK() OVER(PARTITION BY P.project_id ORDER BY
E.experience_years DESC) AS rn FROM Employee E INNER JOIN Project P
ON E.employee_id = P.employee_id)
SELECT project_id,employee_id FROM CTE WHERE rn = 1
```

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

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

Q.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.

Solution:

```
-- create table Books
CREATE TABLE Books
(
   book_id INT PRIMARY KEY,
   `name` VARCHAR(30),
   available_from DATE
);

-- insert records
INSERT INTO Books VALUES(1,'Kalila And Demna','2010-01-01');
INSERT INTO Books VALUES(2,'28 Letters','2012-05-12');
INSERT INTO Books VALUES(3,'The Hobbit','2019-06-10');
INSERT INTO Books VALUES(4,'13 Reasons Why','2019-06-01');
INSERT INTO Books VALUES(5,'The Hunger Games','2008-09-21');

-- create table Orders
CREATE TABLE Orders
(
   order_id INT PRIMARY KEY,
   book_id INT,
   quantity INT,
   dispatch_date DATE,
   FOREIGN KEY (book_id) REFERENCES Books(book_id)
);
```

-- Insert Records

```
INSERT INTO Orders VALUES (1,1,2,'2018-07-26');
INSERT INTO Orders VALUES (2,1,1,'2018-11-05');
INSERT INTO Orders VALUES (3,3,8,'2019-06-11');
INSERT INTO Orders VALUES (4,4,6,'2019-06-05');
INSERT INTO Orders VALUES (5,4,5,'2019-06-20');
INSERT INTO Orders VALUES (6,5,9,'2009-02-02');
INSERT INTO Orders VALUES (7,5,8,'2010-04-13');
```

--Final query:

```
SELECT book_id, `name` FROM Books WHERE book_id NOT IN (
SELECT book_id FROM Orders WHERE dispatch_date BETWEEN DATE_SUB('2019-06-23' ,
INTERVAL 1 YEAR) AND '2019-06-23'
GROUP BY book_id HAVING SUM(quantity)>=10) AND available_from <
DATE_SUB('2019-06-23', INTERVAL 1 MONTH);
```

Data Output:

```
SELECT book_id, `name` FROM Books WHERE book_id NOT IN (
SELECT book_id FROM Orders WHERE dispatch_date BETWEEN DATE_SUB('2019-06-23' , INTERVAL 1 YEAR) AND
'2019-06-23'
GROUP BY book_id HAVING SUM(quantity)>=10) AND available_from < DATE_SUB('2019-06-23', INTERVAL 1
MONTH)
```

Input to filter result | Free | 1 | Cost: 2ms < 1 > Total 3

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

Q.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.

Solution:

```
-- CREATE TABLE Enrollments
CREATE TABLE Enrollments
(
   student_id INT,
   course_id INT,
   grade INT,
   PRIMARY KEY (student_id, course_id)
);

-- Insert Records
INSERT INTO Enrollments VALUES(2,2,95);
INSERT INTO Enrollments VALUES(2,3,95);
INSERT INTO Enrollments VALUES(1,1,90);
INSERT INTO Enrollments VALUES(1,2,99);
INSERT INTO Enrollments VALUES(3,1,80);
INSERT INTO Enrollments VALUES(3,2,75);
INSERT INTO Enrollments VALUES(3,3,82);

-- Final query:

SELECT student_id,MIN(course_id) AS course_id,MAX(grade)AS grade FROM Enrollments
WHERE grade IN (
SELECT MAX(grade) FROM Enrollments GROUP BY student_id) GROUP BY student_id;
```

Data Output:

```
select student_id,min(course_id) as course_id,max(grade)as grade from Enrollments where grade IN (
select max(grade) from Enrollments group by student_id) GROUP BY student_id
```

Input to filter result | Free | 1 | Cost: 7ms | 1 | Total 3

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

Q.50 The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player_id wins. Write an SQL query to find the winner in each group.
Solution:

```
-- CREATE TABLE Players
CREATE TABLE Players
(
   player_id INT PRIMARY KEY,
   group_id INT
);

--insert Records
INSERT INTO Players VALUES(15,1);
INSERT INTO Players VALUES(25,1);
INSERT INTO Players VALUES(30,1);
INSERT INTO Players VALUES(45,1);
INSERT INTO Players VALUES(10,2);
INSERT INTO Players VALUES(35,2);
INSERT INTO Players VALUES(50,2);
INSERT INTO Players VALUES(20,3);
INSERT INTO Players VALUES(40,3);

-- CREATE TABLE Matches
CREATE TABLE Matches
(
   group_id       INT PRIMARY KEY,
   first_player   INT,
   second_player  INT,
   first_score    INT,
   second_score   INT
);

--INSERT Records
INSERT INTO Matches VALUES(1,15,45,3,0);
INSERT INTO Matches VALUES(2,30,25,1,2);
INSERT INTO Matches VALUES(3,30,15,2,0);
INSERT INTO Matches VALUES(4,40,20,5,2);
INSERT INTO Matches VALUES(5,35,50,1,1);
```

--Final query:

```
SELECT group_id,player_id FROM(
SELECT group_id,player_id,RANK() OVER (PARTITION BY group_id ORDER BY SUM(score)
DESC, player_id ASC) AS rnk  FROM(
SELECT group_id,P.player_id,SUM(first_score) AS score FROM Players P INNER JOIN
Matches M ON P.player_id = M.first_player
GROUP BY group_id,P.player_id
UNION ALL
SELECT group_id,P.player_id,SUM(second_score) AS score FROM Players P INNER JOIN
Matches M ON P.player_id = M.second_player
GROUP BY group_id,P.player_id)t GROUP BY group_id,player_id)t1 WHERE rnk = 1;
```

Data Output:

```
SELECT group_id,player_id FROM(
SELECT group_id,player_id,RANK() OVER (PARTITION BY group_id ORDER BY SUM(score) DESC, player_id ASC)
AS rnk  FROM(
SELECT group_id,P.player_id,SUM(first_score) AS score FROM Players P INNER JOIN Matches M ON
P.player_id = M.first_player
GROUP BY group_id,P.player_id
UNION ALL
SELECT group_id,P.player_id,SUM(second_score) AS score FROM Players P INNER JOIN Matches M ON
P.player_id = M.second_player
GROUP BY group_id,P.player_id)t GROUP BY group_id,player_id)t1 WHERE rnk = 1
```

Input to filter result Free 1 Cost: 4ms 1 Total 3

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