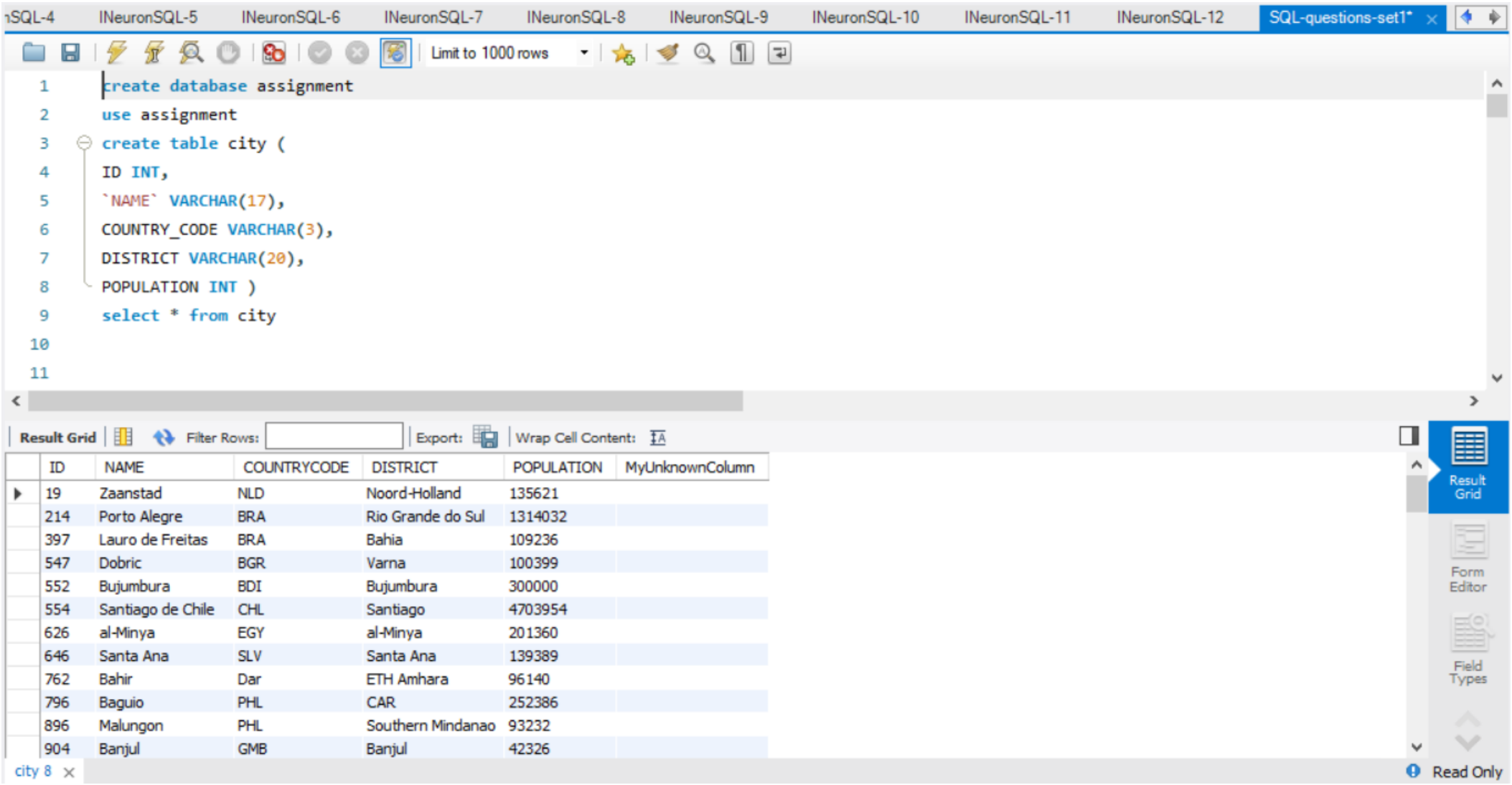
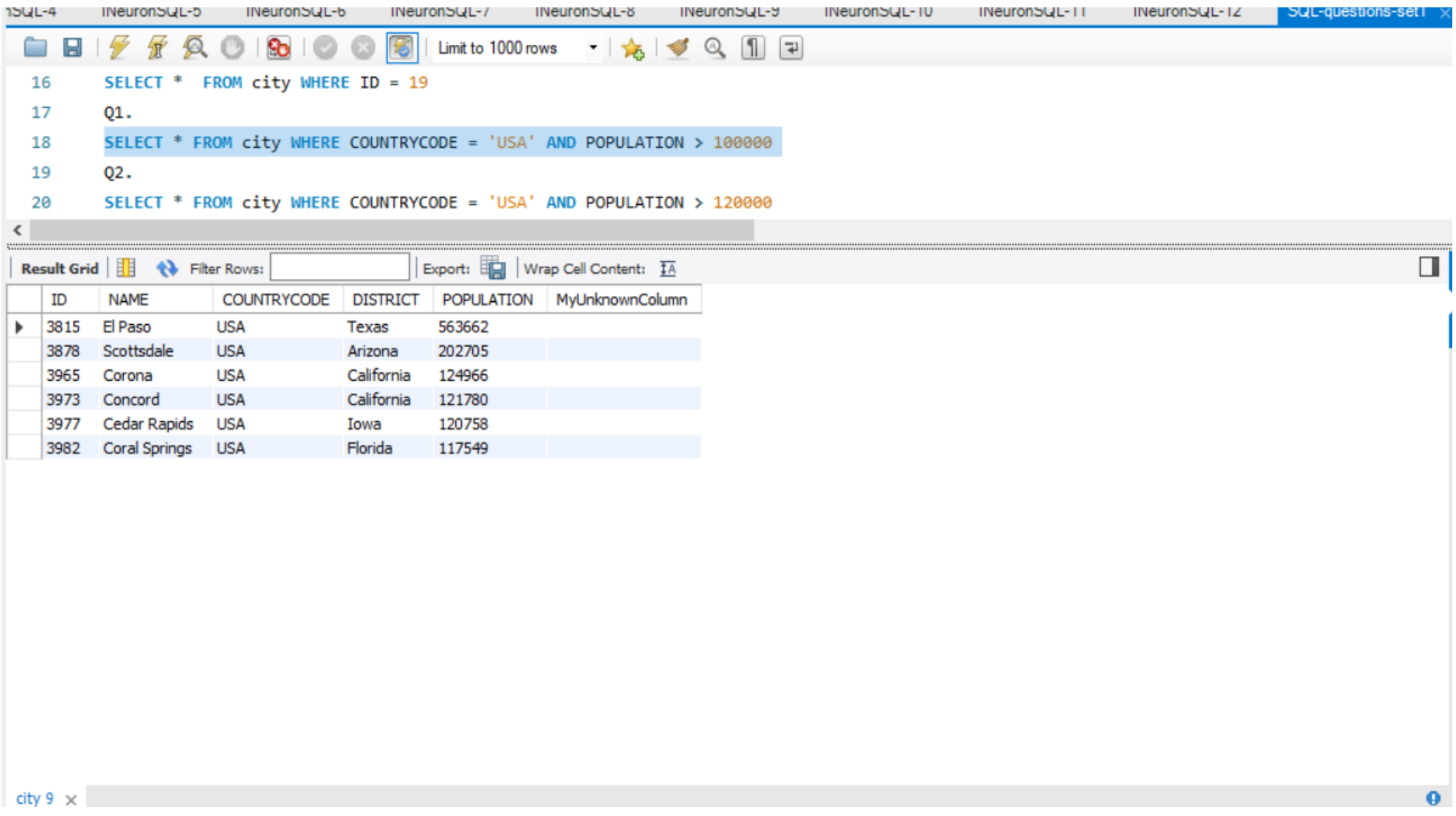


Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA.The CITY table is described as follows:



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

```
19    Q2.
20    SELECT * FROM city WHERE COUNTRYCODE = 'USA' AND POPULATION > 120000
21    Q4.
22    SELECT * FROM city WHERE ID = 1661
23    Q5.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	NAME	COUNTRYCODE	DISTRICT	POPULATION	MyUnknownColumn
3815	El Paso	USA	Texas	563662	
3878	Scottsdale	USA	Arizona	202705	
3965	Corona	USA	California	124966	
3973	Concord	USA	California	121780	
3977	Cedar Rapids	USA	Iowa	120758	

Q3. Query all columns (attributes) for every row in the CITY table. The CITY table is described as follows:

```
23    Q5.
24    SELECT * FROM city WHERE COUNTRYCODE = 'JPN'
25    Q3.
26    SELECT * FROM station
27
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Id	City	State	Lat_N	Long_W
794	Kissee Mills	MO	139	73
824	Loma Mar	CA	48	130
603	Sandy Hook	CT	72	148
478	Tipton	IN	33	97
619	Arlington	CO	75	92
711	Turner	AR	50	101
839	Slidell	LA	85	151
411	Negreet	LA	98	105
588	Glencoe	KY	46	136
665	Chelsea	IA	98	59
342	Chignik Lag...	AK	103	153
733	Pelahatchie	MS	38	28
441	Hanna City	IL	50	136
811	Dorrance	KS	102	121
698	Albany	CA	49	80
325	Monument	KS	70	141
414	Manchester	MD	73	37
113	Prescott	IA	39	65
971	Graettinger	IA	94	150

station 11

Q4. Query all columns for a city in CITY with the ID 1661.The CITY table is described as follows:

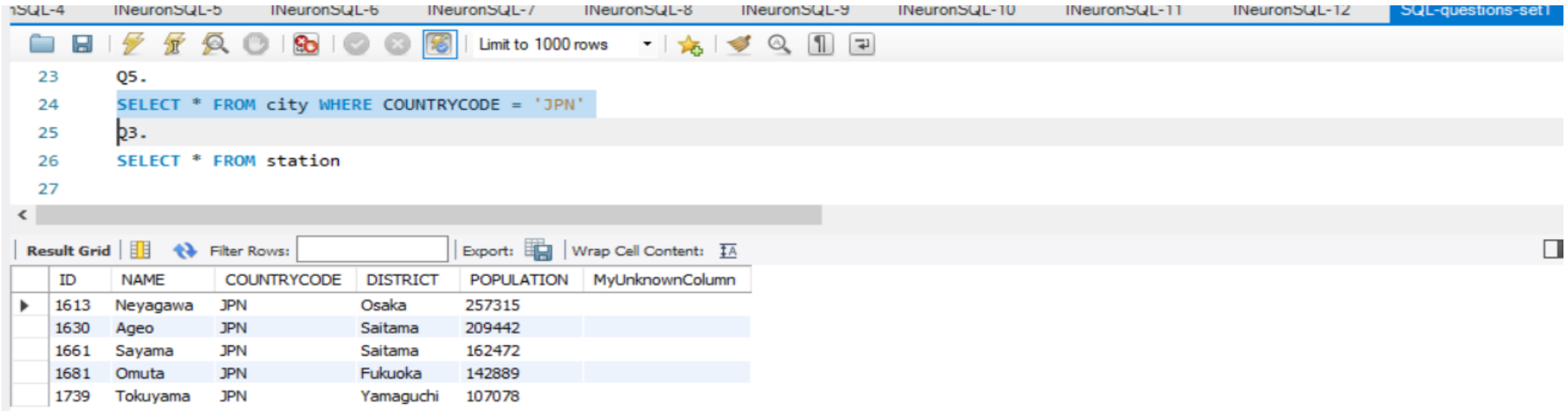
INeuronSQL-4 INeuronSQL-5 INeuronSQL-6 INeuronSQL-7 INeuronSQL-8 INeuronSQL-9 INeuronSQL-10 INeuronSQL-11 INeuronSQL-12 SQL-questions-set1

```
20    SELECT * FROM city WHERE COUNTRYCODE = 'USA' AND POPULATION > 120000
21    Q4.
22    SELECT * FROM city WHERE ID = 1661
23    Q5.
24    SELECT * FROM city WHERE COUNTRYCODE = 'JPN'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

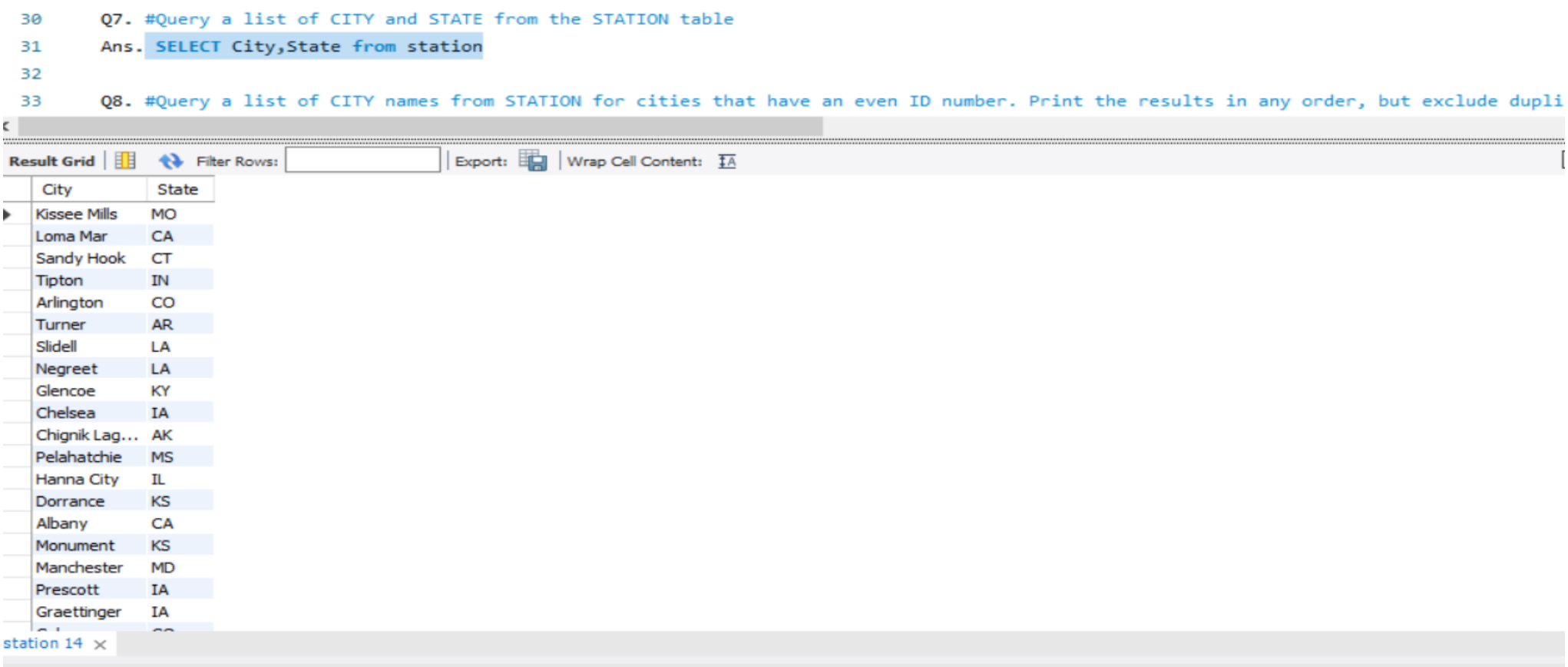
ID	NAME	COUNTRYCODE	DISTRICT	POPULATION	MyUnknownColumn
1661	Sayama	JPN	Saitama	162472	

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



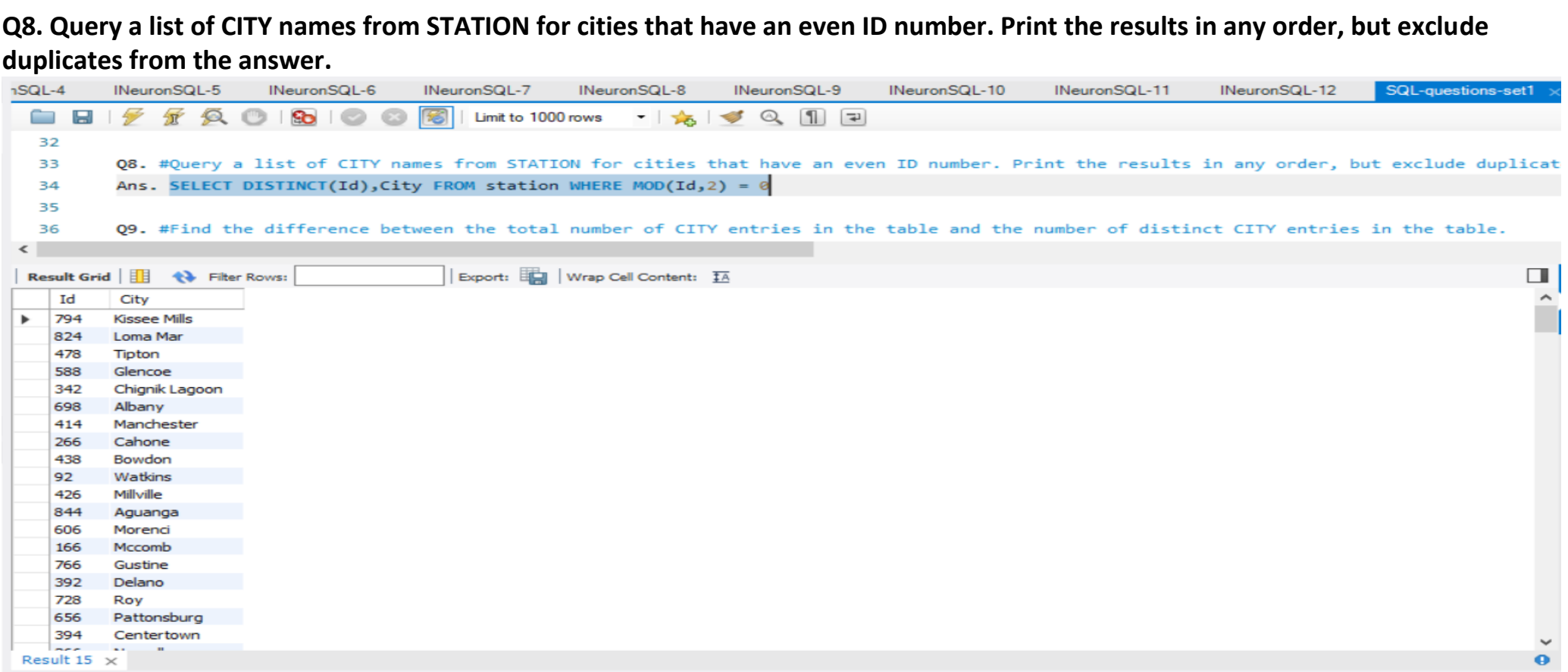
ID	NAME	COUNTRYCODE	DISTRICT	POPULATION	MyUnknownColumn
1613	Neyagawa	JPN	Osaka	257315	
1630	Ageo	JPN	Saitama	209442	
1661	Sayama	JPN	Saitama	162472	
1681	Omuta	JPN	Fukuoka	142889	
1739	Tokuyama	JPN	Yamaguchi	107078	

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



City	State
Kissee Mills	MO
Loma Mar	CA
Sandy Hook	CT
Tipton	IN
Arlington	CO
Turner	AR
Slidell	LA
Negreet	LA
Glencoe	KY
Chelsea	IA
Chignik Lag...	AK
Pelahatchie	MS
Hanna City	IL
Dorrance	KS
Albany	CA
Monument	KS
Manchester	MD
Prescott	IA
Graettinger	IA

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.



Id	City
794	Kissee Mills
824	Loma Mar
478	Tipton
588	Glencoe
342	Chignik Lagoon
698	Albany
414	Manchester
266	Cahone
438	Bowdon
92	Watkins
426	Millville
844	Aguanga
606	Morenci
166	Mccomb
766	Gustine
392	Delano
728	Roy
656	Pattonsburg
394	Centertown

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

```
35
36    Q9. #Find the difference between the total number of CITY entries in the table and the number of di
37    Ans. SELECT (COUNT(City) - COUNT(DISTINCT(City))) as difference from station
38
39    Q10. #Query the two cities in STATION with the shortest and longest CITY names, as well as their re
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

difference
13

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.

INeuronSQL-4 | INeuronSQL-5 | INeuronSQL-6 | INeuronSQL-7 | INeuronSQL-8 | INeuronSQL-9 | INeuronSQL-10 | INeuronSQL-11 | INeuronSQL-12

Limit to 1000 rows

```
38
39    Q10. #Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.:
40    Ans. SELECT City,LENGTH(City) as Length FROM station WHERE City like '___' ORDER BY City
41         SELECT City,LENGTH(City) as Length FROM station WHERE City like '_________________'ORDER BY City
42
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

City	Length
Amo	3
Lee	3
Roy	3

INeuronSQL-4 | INeuronSQL-5 | INeuronSQL-6 | INeuronSQL-7 | INeuronSQL-8 | INeuronSQL-9 | INeuronSQL-10 | INeuronSQL-11 | INeuronSQL-12 | SQL-questions-

Limit to 1000 rows

```
38
39    Q10. #Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number o
40    Ans. SELECT City,LENGTH(City) as Length FROM station WHERE City like '___' ORDER BY City
41         SELECT City,LENGTH(City) as Length FROM station WHERE City like '_________________'ORDER BY City
42
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

City	Length
West Baden Springs	18
White Horse Beach	18

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.

```
2
3     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.
4     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%')
5     ORDER BY City
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

City
Acme
Addison
Agency
Aguanga
Alanson
Alba
Albany
Albion
Algonac
Aliso Viejo
Allerton
Alpine
Alton
Amazonia
Amo
Anderso...
Andover
Anthony
Archie

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

Limit to 1000 rows

```
47    Q12. #Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.
48    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')
49
50    Q13. #Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.
51    Ans. SELECT DISTINCT(City) FROM station WHERE (City NOT like 'a%' AND City NOT like 'e%' AND City NOT like 'i%' AND City NOT like 'o%'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

City
Glencoe
Chelsea
Pelahatchie
Dorrance
Cahone
Upperco
Waipahu
Millville
Aguanga
Morenci
South El M...
Gustine
Delano
Westphalia
Saint Elmo
Raymondville
Barrigada
Hesperia
Wickliffe

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

```
50  Q13. #Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.
51  SELECT DISTINCT(City) FROM station
52  WHERE (City NOT like 'a%' AND City NOT like 'e%' AND City NOT like 'i%' AND City NOT like 'o%' AND City NOT like 'u%')
53  ORDER BY City
54
```

City
Baileyville
Bainbridge
Baker
Baldwin
Barrigada
Bass Harbor
Baton Rouge
Bayville
Beaufort
Beaver Island
Bellevue
Benedict
Bennington
Bentonville
Berryton
Bertha
Beverly
Biggsville
Bison

Result 22

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

```
54
55  Q14. #Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.
56  SELECT DISTINCT(City) FROM station
57  WHERE (City NOT like '%a' AND City NOT like '%e' AND City NOT like '%i' AND City NOT like '%o' AND City NOT like '%u')
58
```

City
Kissee Mills
Loma Mar
Sandy Hook
Tipton
Arlington
Turner
Slidell
Negreet
Chignik Lagoon
Hanna City
Albany
Monument
Manchester
Prescott
Graettinger
Sturgis
Highwood
Bowdon
Tyler

Result 23

Read Only

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.

```
57  WHERE (City NOT like '%a' AND City NOT like '%e' AND City NOT like '%i' AND City NOT like '%o' AND City NOT like '%u')
58
59  Q15.SELECT DISTINCT(City) FROM station
60  WHERE (City NOT like 'a%' AND City NOT like 'e%' AND City NOT like 'i%' AND City NOT like 'o%' AND City NOT like 'u%') AND
61  (City NOT like '%a' AND City NOT like '%e' AND City NOT like '%i' AND City NOT like '%o' AND City NOT like '%u')
62
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

City
Kissee Mills
Loma Mar
Sandy Hook
Tipton
Turner
Slidell
Negreet
Chignik Lagoon
Hanna City
Monument
Manchester
Prescott
Graettinger
Sturgis
Highwood
Bowdon
Tyler
Watkins

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.

```
Q17.
CREATE TABLE Product(
product_id int not null,
product_name varchar(40),
unit_price int,
primary key (product_id))

INSERT INTO Product values (1 , 'S8', 1000),
(2,'G4', 800),
(3,'iPhone', 1400)
select * from Product
```

```
75  CREATE TABLE Sales (
76  seller_id int,
77  product_id int,
78  buyer_id int,
79  sale_date date,
80  quantity int,
81  price int,
82  foreign key(product_id) references Product(product_id) )
83  insert into Sales values(1,1,1,'2019-01-21',2,2000)
84  insert into Sales values(1,2,2,'2019-02-17',1,800),
85  (2,2,3,'2019-06-02',1,800),
86  (3,3,4,'2019-05-13',2,2800)
87  SELECT * FROM Sales
88
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Result Grid | Form Editor | Read Only

Sales 26

```sql
insert into Sales values(1,2,2,'2019-02-17',1,800),
(2,2,3,'2019-06-02',1,800),
(3,3,4,'2019-05-13',2,2800)
SELECT * FROM Sales

WITH Q1 AS (SELECT * FROM Sales WHERE sale_date BETWEEN '2019-01-01' AND '2019-03-31')
SELECT P.product_id,P.product_name,Q.sale_date FROM Product P INNER JOIN Q1 Q ON P.product_id = Q.product_id

SELECT product_id,MONTH(sale_date) AS Month FROM Sales WHERE MONTH(sale_date) IN (1,2) AND MONTH(sale_date) NOT IN (5,6)
AND sale_date NOT IN BETWEEN '2019-01-04' AND '2019-31-12'


Q21.
CREATE TABLE Employee(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_id	product_name	sale_date
1	S8	2019-01-21
2	G4	2019-02-17

Q21. Write an SQL query to find the team size of each of the employees.

```sql
Q21.
CREATE TABLE Employee(
employee_id int,
team_id int,
primary key (employee_id)
)
INSERT INTO Employee VALUES (1,8),
(2,8),
(3,8),
(4,7),
(5,9),
(6,9)
SELECT * FROM Employee
WITH team_CTE AS (SELECT team_id,count(employee_id) as total FROM Employee GROUP BY team_id)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9
NULL	NULL

```sql
INSERT INTO Employee VALUES (1,8),
(2,8),
(3,8),
(4,7),
(5,9),
(6,9)
SELECT * FROM Employee
WITH team_CTE AS (SELECT team_id,count(employee_id) as total FROM Employee GROUP BY team_id)
SELECT E.employee_id,T.total AS team_size FROM Employee E INNER JOIN team_CTE T ON E.team_id = T.team_id

Q22.
CREATE TABLE Countries(
country_id int,
country_name varchar(30),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	team_size
1	3
2	3
3	3
4	1
5	2
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.**

```
Q22.
CREATE TABLE Countries(
country_id int,
country_name varchar(30),
PRIMARY KEY (country_id)
)
INSERT INTO Countries VALUES (2,'USA'),
(3,'Australia'),
(7,'Peru'),
(5,'China'),
(8,'Morocco'),
(9,'Spain')
SELECT * FROM Countries
```

country_id	country_name
2	USA
3	Australia
5	China
7	Peru
8	Morocco
9	Spain
NULL	NULL

```
CREATE TABLE Weather(
country_id int,
weather_state int,
`day` date,
PRIMARY KEY (country_id,`day`)
)
INSERT INTO Weather VALUES (2,15,'2019-11-01'),
(2,12,'2019-10-28'),
(2,12,'2019-10-27'),
(3,-2,'2019-11-10'),
(3,0,'2019-11-11'),
(3,3,'2019-11-12'),
(5,16,'2019-11-07'),
(5,18,'2019-11-09'),
(5,21,'2019-11-23'),
```

```
SELECT * FROM Weather
SELECT * FROM Countries

DELIMITER &&
CREATE FUNCTION Type_weather1(weather_state int)
```

country_id	weather_state	day
2	12	2019-10-27
2	12	2019-10-28
2	15	2019-11-01
3	-2	2019-11-10
3	0	2019-11-11
3	3	2019-11-12
5	16	2019-11-07
5	18	2019-11-09
5	21	2019-11-23
7	25	2019-11-28
7	22	2019-12-01
7	20	2019-12-02
8	25	2019-11-05
8	27	2019-11-15
8	31	2019-11-25
9	7	2019-10-23
9	3	2019-12-23
NULL	NULL	NULL

```sql
DELIMITER &&
CREATE FUNCTION Type_weather1(weather_state int)
RETURNS VARCHAR(30)
DETERMINISTIC
BEGIN
DECLARE weather VARCHAR(30);
if weather_state <= 15 then
SET weather = 'Cold';
elseif weather_state >= 25 then
SET weather = 'Hot';
else
SET weather = 'Warm';
END if;
RETURN weather;
END &&;
```

```sql
WITH Country_Weather_Check AS  (SELECT country_id,Type_weather1(weather_state) AS Weather_type FROM Weather
WHERE day BETWEEN '2019-11-01' AND '2019-11-30' GROUP BY country_id)
SELECT C.country_name , CWC.Weather_type
FROM Countries C INNER JOIN Country_Weather_Check CWC ON C.country_id = CWC.country_id
```

Result Grid

country_name	Weather_type
USA	Cold
Australia	Cold
China	Warm
Peru	Hot
Morocco	Hot

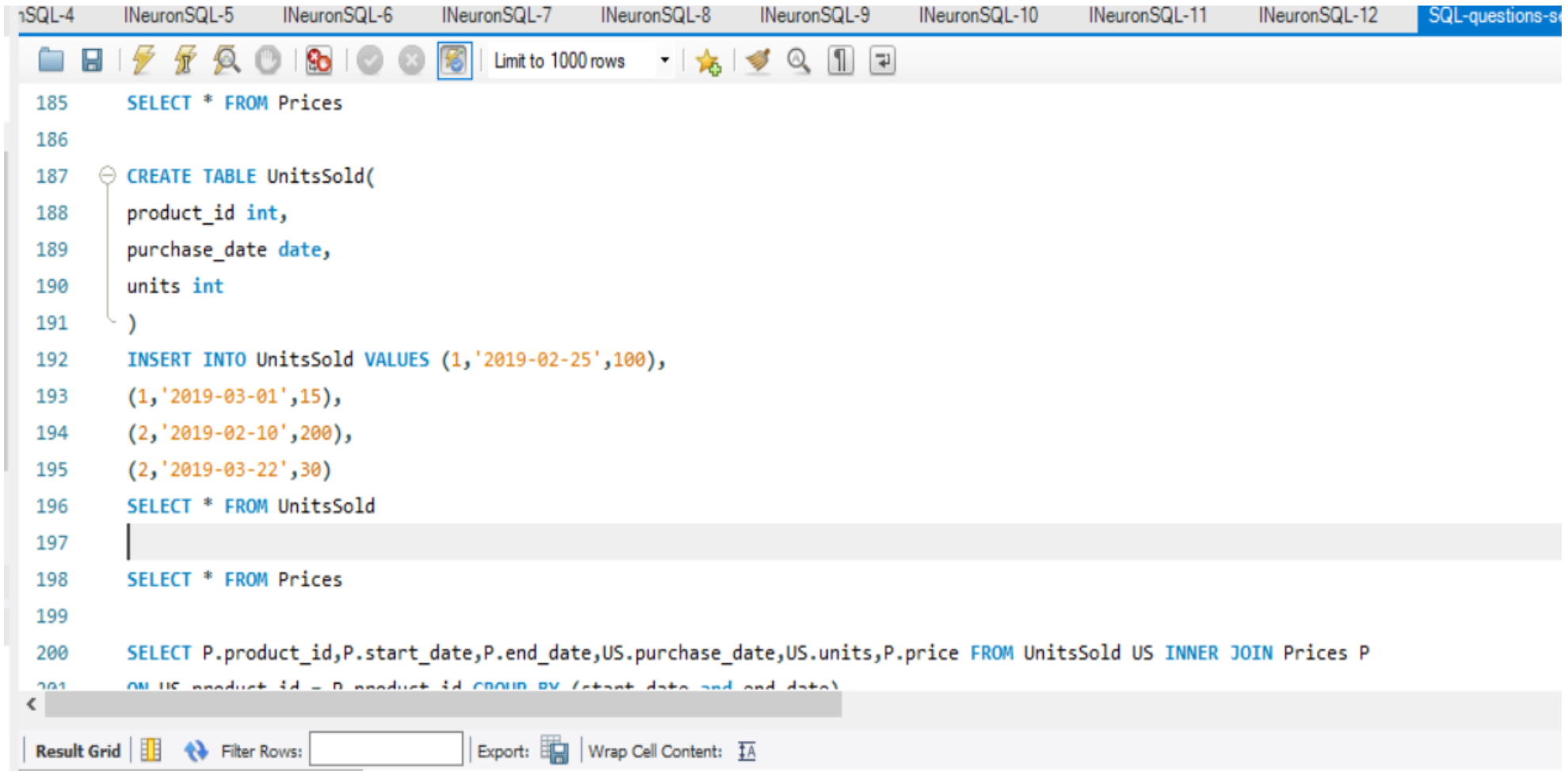
Q23. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places.

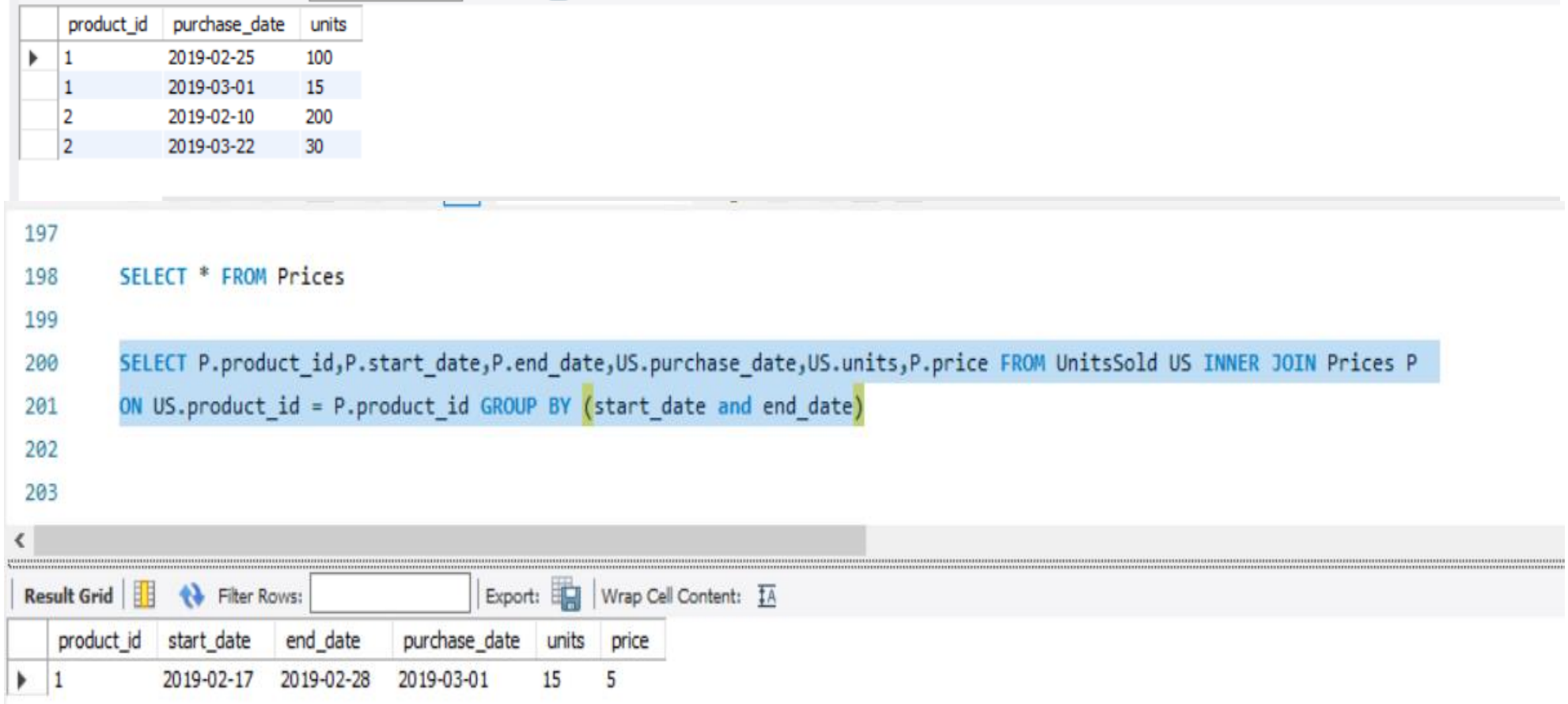
```sql
Q23.
CREATE TABLE Prices(
product_id int,
start_date date,
end_date date,
price int,
PRIMARY KEY (product_id,start_date,end_date)
)
INSERT INTO Prices VALUES (1,'2019-02-17','2019-02-28',5),
(1,'2019-03-01','2019-03-22',20),
(2,'2019-02-01','2019-02-20',15),
(2,'2019-02-21','2019-03-31',30)
SELECT * FROM Prices

CREATE TABLE UnitsSold(
product_id int,
```

Result Grid

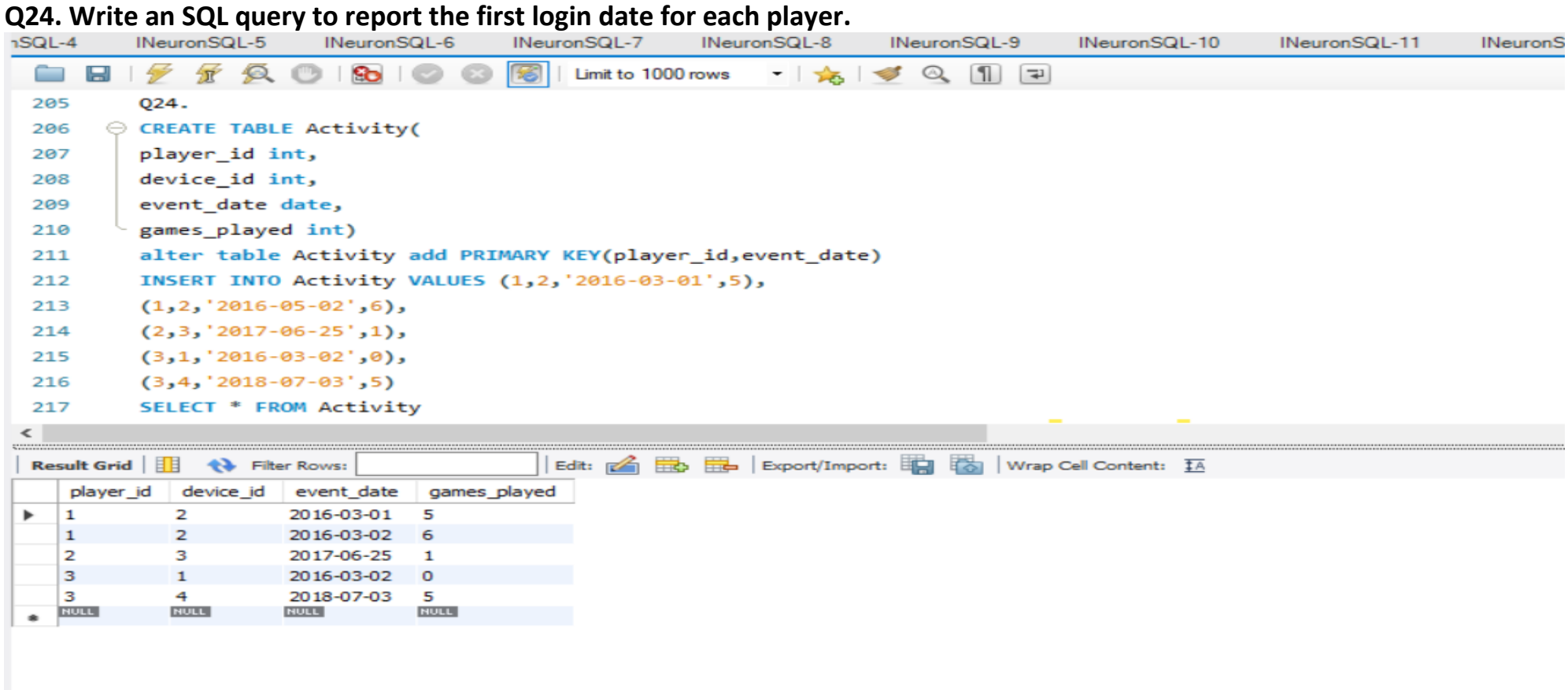
product_id	start_date	end_date	price
1	2019-02-17	2019-02-28	5
1	2019-03-01	2019-03-22	20
2	2019-02-01	2019-02-20	15
2	2019-02-21	2019-03-31	30
NULL	NULL	NULL	NULL

Prices 34





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



```
217     SELECT * FROM Activity
218     SELECT player_id,MIN(event_date) as first_login from Activity GROUP BY (player_id)
219
220
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

Q25. Write an SQL query to report the device that is first logged in for each player.

```
220
221     Q25.
222     SELECT player_id,MIN(event_date) as first_login,device_id from Activity GROUP BY (player_id)
223
224     Q26.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

player_id	first_login	device_id
1	2016-03-01	2
2	2017-06-25	3
3	2016-03-02	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.

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- assignment
  - Tables
  - Views
  - Stored Procedures
  - Functions
- dress_data
- ineuron
- ineuron_partition
- ineuron_triggers
- key_prim
- online_retail
- operation
- pivote
- sakila
- sales
- sys

Administration Schemas

Information

No object selected

Object Info Session

SQL-4 INeuronSQL-5 INeuronSQL-6 INeuronSQL-7 INeuronSQL-8 INeuronSQL-9 INeuronSQL-10 INeuronSQL-11 INeuronSQL-12 SQL-questions-set1*

Limit to 1000 rows

```
223
224     Q26.
225   CREATE TABLE IF NOT EXISTS Products(
226   product_id int,
227   product_name varchar(50),
228   product_category varchar(50))
229   ALTER TABLE Products add PRIMARY KEY(product_id)
230   INSERT INTO Products VALUES(1,'Leetcode Solutions','Book'),
231   (2,'Jewels of Stringology','Book'),
232   (3,'HP','Laptop'),
233   (4,'Lenovo','Laptop'),
234   (5,'Leetcode Kit','T-shirt')
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	product_name	product_category
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt
NULL	NULL	NULL

Result Grid | Form Editor | Field Types

Products 40 | Apply | Revert

SQLAdditions | Jump to

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

Context Help | Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
49	23:07:28	SELECT player_id,MIN(event_date) as first_login from Activity GROUP BY (player_id) LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
50	23:07:59	SELECT player_id,MIN(event_date) as first_login,device_id from Activity GROUP BY (player_id) LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
51	23:08:58	SELECT *FROM Products LIMIT 0, 1000	5 row(s) returned	0.032 sec / 0.000 sec

Type here to search

28°C Cloudy ENG 11:09 PM 12/18/2022

```
240     unit int)
241     INSERT INTO Orders VALUES(1,'2020-02-05',60),
242     (1,'2020-02-10',70),
243     (2,'2020-01-18',30),
244     (2,'2020-02-11',80),
245     (3,'2020-02-17',2),
246     (3,'2020-02-24',3),
247     (4,'2020-03-01',20),
248     (4,'2020-03-04',30),
249     (4,'2020-03-04',60),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_id	order_date	unit
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

```
255
256     WITH Orders_tmp AS (SELECT product_id,SUM(unit)as unit FROM Orders WHERE order_date BETWEEN '2020-02-01' AND '2020-02-28'
257     GROUP BY (product_id) ORDER BY unit desc limit 2)
258     SELECT P.product_name,OT.unit FROM Products P INNER JOIN Orders_tmp OT ON P.product_id = OT.product_id
259
260     Q27.
261     CREATE TABLE Users(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Q27. Write an SQL query to find the users who have valid emails.

Limit to 1000 rows

```
260     Q27.
261     CREATE TABLE Users(
262     user_id int,
263     `name` varchar(50),
264     mail varchar(50),
265     PRIMARY KEY (user_id))
266     INSERT INTO Users VALUES (1,'Winston','winston@leetcode.com'),
267     (2,'Jonathan', 'jonathanisgreat'),
268     (3,'Annabelle','bella-@leetcode.com'),
269     (4, 'Sally', 'sally.come@leetcode.com'),
270     (5, 'Marwan','quarz#2020@leetcode.com'),
271     (6, 'David','david69@gmail.com'),
272     (7, 'Shapiro','.shapo@leetcode.com')
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

user_id	name	mail
1	Winston	winston@leetcode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella-@leetcode.com
4	Sally	sally.come@leetcode.com
5	Marwan	quarz#2020@leetcode.com
6	David	david69@gmail.com
7	Shapiro	.shapo@leetcode.com
NULL	NULL	NULL

Users 43

Q28. 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.

```
275
276   Q28.
277   CREATE TABLE IF NOT EXISTS Customers(
278   customer_id int,
279   `name` varchar(50),
280   country varchar(50),
281   PRIMARY KEY(customer_id)
282   )
283   INSERT INTO Customers VALUES (1,'Winston', 'USA'),
284   (2,'Jonathan', 'Peru'),
285   (3,'Moustafa', 'Egypt')
286   SELECT * FROM Customers
287
```

Result Grid

customer_id	name	country
1	Winston	USA
2	Jonathan	Peru
3	Moustafa	Egypt
NULL	NULL	NULL

```
287
288   CREATE TABLE IF NOT EXISTS Product28(
289   product_id int,
290   `description` varchar(50),
291   price int,
292   PRIMARY KEY (product_id))
293   INSERT INTO Product28 VALUES (10,'LC-Phone',300),
294   (20,'LC T-shirt',10),
295   (30,'LC Book',45),
296   (40,'LC Keychain',2)
297   SELECT * FROM Product28
298
299   CREATE TABLE IF NOT EXISTS Orders28(
```

Result Grid

product_id	description	price
10	LC-Phone	300
20	LC T-shirt	10
30	LC Book	45
40	LC Keychain	2
NULL	NULL	NULL

```
297   SELECT * FROM Product28
298
299   CREATE TABLE IF NOT EXISTS Orders28(
300   order_id int,
301   customer_id int,
302   product_id int,
303   order_date date,
304   quantity int,
305   PRIMARY KEY (order_id))
306   INSERT INTO Orders28 VALUES (1,1,10,'2020-06-10',1),
307   (2,1,20,'2020-07-01',1),
308   (3,1,30,'2020-07-08',2),
309   (4,2,10,'2020-06-15',2),
```

Result Grid

order_id	customer_id	product_id	order_date	quantity
1	1	10	2020-06-10	1
2	1	20	2020-07-01	1
3	1	30	2020-07-08	2
4	2	10	2020-06-15	2
5	2	40	2020-07-01	10
6	3	20	2020-06-24	2
7	3	30	2020-06-25	2
9	3	30	2020-05-08	3
NULL	NULL	NULL	NULL	NULL

```sql
DELIMITER $$
CREATE FUNCTION Total_price(quantity INT, price INT)
RETURNS INT
DETERMINISTIC
BEGIN
DECLARE final_price INT;
SET final_price = (quantity*price);
RETURN final_price;
END $$
```

```sql
WITH Customer_order_product AS(WITH Customer_orders AS (WITH Order28_tmp AS (SELECT * FROM Orders28 WHERE order_date BETWEEN '2020-06-01' AND '
SELECT OT.customer_id,C.name, C.country, OT.order_date, OT.product_id, OT.quantity, OT.order_id
FROM customers C INNER JOIN Order28_tmp OT ON C.customer_id = OT.customer_id)
SELECT CO.customer_id,CO.name, CO.country, CO.order_date, CO.product_id, CO.quantity,P.price,P.description, CO.order_id
FROM Product28 P INNER JOIN Customer_orders CO ON P.product_id = CO.product_id)
SELECT customer_id,name,order_date,month(order_date) as Month,SUM(Total_price(quantity,price)) as final_price FROM Customer_order_product
WHERE name ='winston' GROUP BY month(order_date)

DELIMITER $$
CREATE FUNCTION Total_price(quantity INT, price INT)
RETURNS INT
DETERMINISTIC
```

customer_id	name	order_date	Month	final_price
1	Winston	2020-06-10	6	300
1	Winston	2020-07-01	7	100

Q29. 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

```sql
Q29.
CREATE TABLE TV_Program(
program_date DATE,
content_id INT,
`channel` VARCHAR(50) )
ALTER TABLE TV_Program ADD PRIMARY KEY(program_date,content_id)
ALTER TABLE TV_Program MODIFY program_date DATETIME
TRUNCATE TABLE TV_Program
INSERT INTO TV_Program 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'),
```

program_date	content_id	channel
2020-05-11 12:00:00	2	LC-Channel
2020-05-12 12:00:00	3	LC-Channel
2020-05-13 14:00:00	4	Disney Ch
2020-06-10 08:00:00	1	LC-Channel
2020-06-18 14:00:00	4	Disney Ch
2020-07-15 16:00:00	5	Disney Ch
NULL	NULL	NULL

Limit to 1000 rows

```
355  CREATE TABLE IF NOT EXISTS Content(
356  content_id INT,
357  title VARCHAR(50),
358  kids_content ENUM('Y','N'),
359  content_type VARCHAR(50),
360  PRIMARY KEY (content_id))
361  INSERT INTO Content VALUES (1, 'Leetcode Movie', 'N', 'Movies'),
362  (2, 'Alg. for Kids', 'Y', 'Series'),
363  (3, 'Database Sols', 'N', 'Series'),
364  (4, 'Aladdin', 'Y', 'Movies'),
365  (5, 'Cinderella', 'Y', 'Movies')
366  SELECT * FROM Content
367  SELECT * FROM TV_Program
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

content_id	title	kids_content	content_type
1	Leetcode Movie	N	Movies
2	Alg. for Kids	Y	Series
3	Database Sols	N	Series
4	Aladdin	Y	Movies
5	Cinderella	Y	Movies
NULL	NULL	NULL	NULL

```
369  CREATE VIEW TV_Program_Content AS SELECT * FROM TV_Program WHERE program_date BETWEEN '2020-06-01' AND '2020-06-30';
370  SELECT * FROM TV_Program_Content
371
372  CREATE VIEW Content_tmp AS SELECT * FROM Content WHERE kids_content = 'Y' AND content_type = 'Movies';
373  SELECT * FROM Content_tmp
374
375  SELECT * FROM TV_Program_Content TPC INNER JOIN Content_tmp CT ON TPC.content_id = CT.content_id
376
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

program_date	content_id	channel
2020-06-10 08:00:00	1	LC-Channel
2020-06-18 14:00:00	4	Disney Ch

```
371
372  CREATE VIEW Content_tmp AS SELECT * FROM Content WHERE kids_content = 'Y' AND content_type = 'Movies';
373  SELECT * FROM Content_tmp
374
375  SELECT * FROM TV_Program_Content TPC INNER JOIN Content_tmp CT ON TPC.content_id = CT.content_id
376
377  Q30.
378  CREATE TABLE NPV (
379  id int,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

content_id	title	kids_content	content_type
4	Aladdin	Y	Movies
5	Cinderella	Y	Movies

```
374
375  SELECT * FROM TV_Program_Content TPC INNER JOIN Content_tmp CT ON TPC.content_id = CT.content_id
376
377  Q30.
378  CREATE TABLE NPV (
379  id int,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

program_date	content_id	channel	content_id	title	kids_content	content_type
2020-06-18 14:00:00	4	Disney Ch	4	Aladdin	Y	Movies

Q30. Write an SQL query to find the npv of each query of the Queries table

```
377  Q30.
378  CREATE TABLE NPV (
379  id int,
380  `year` int,
381  npv int,
382  PRIMARY KEY (id,year) )
383  INSERT INTO NPV VALUES (1, 2018, 100),
384  (7, 2020, 30),
385  (13, 2019, 40),
386  (1, 2019, 113),
387  (2, 2008, 121),
388  (3, 2009, 12),
389  (11, 2020, 99),
390  (7, 2019, 0)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	year	npv
1	2018	100
1	2019	113
2	2008	121
3	2009	12
7	2019	0
7	2020	30
11	2020	99
13	2019	40
NULL	NULL	NULL

nSQL-4 | INeuronSQL-5 | INeuronSQL-6 | INeuronSQL-7 | INeuronSQL-8 | INeuronSQL-9 | INeuronSQL-10 | INeuronSQL-11 | INeuronSQL-12 | SQL-questions-set1*

```
393  CREATE TABLE Queries(
394  id int,
395  `year` int,
396  PRIMARY KEY(id,year))
397  INSERT INTO Queries VALUES (1, 2019),
398  (2, 2008),
399  (3, 2009),
400  (7, 2018),
401  (7, 2019),
402  (7, 2020),
403  (13, 2019)
404  SELECT * FROM Queries
405  SELECT * FROM NPV
406
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019
NULL	NULL

```
406
407  SELECT * FROM NPV WHERE year in (SELECT year FROM Queries) ORDER BY id
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	year	npv
1	2018	100
1	2019	113
2	2008	121
3	2009	12
7	2019	0
7	2020	30
11	2020	99
13	2019	40
NULL	NULL	NULL

Q32. 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.

```
408
409     Q32.
410     CREATE TABLE Employees(
411     d INT,
412     `name` VARCHAR(50) )
413     ALTER TABLE Employees RENAME COLUMN d TO id
414     ALTER TABLE Employees ADD PRIMARY KEY(id)
415     INSERT INTO Employees VALUES (1, 'Alice'),
416     (7, 'Bob'),
417     (11, 'Meir'),
418     (90, 'Winston'),
419     (3, 'Jonathan')
420     SELECT * FROM Employees
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name
1	Alice
3	Jonathan
7	Bob
11	Meir
90	Winston
NULL	NULL

```
420     SELECT * FROM Employees
421
422     CREATE TABLE EmployeeUNI(
423     id INT,
424     unique_id INT,
425     PRIMARY KEY (id,unique_id) )
426     INSERT INTO EmployeeUNI VALUES (3, 1),
427     (11, 2),
428     (90, 3)
429     SELECT * FROM EmployeeUNI
430
431     SELECT EU.unique_id,E.name FROM Employees E LEFT JOIN EmployeeUNI EU ON E.id = EU.id
432
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	unique_id
3	1
11	2
90	3
NULL	NULL

```
429     SELECT * FROM EmployeeUNI
430
431     SELECT EU.unique_id,E.name FROM Employees E LEFT JOIN EmployeeUNI EU ON E.id = EU.id
432
433     Q33.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

unique_id	name
NULL	Alice
1	Jonathan
NULL	Bob
2	Meir
3	Winston

Q33.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.

```
Q33.
CREATE TABLE IF NOT EXISTS Users33(
id INT,
`name` VARCHAR(50),
PRIMARY KEY(id) )
INSERT INTO Users33 VALUES (1, 'Alice'),
(2, 'Bob'),
(3, 'Alex'),
(4, 'Donald'),
(7, 'Lee'),
(13,'Jonathan'),
(19, 'Elvis')
```

id	name
1	Alice
2	Bob
3	Alex
4	Donald
7	Lee
13	Jonathan
19	Elvis

```
CREATE TABLE IF NOT EXISTS Rides(
id INT,
user_id INT,
distance INT,
PRIMARY KEY (id))
INSERT INTO Rides VALUES (1, 1, 120),
(2, 2, 317),
(3, 3, 222),
(4, 7, 100),
(5, 13, 312),
(6, 19, 50),
```

id	user_id	distance
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

```
WITH Rides_CTE AS (SELECT user_id,SUM(distance) AS travelled_distance FROM Rides GROUP BY user_id)
SELECT U.name,R.travelled_distance FROM Users33 U LEFT JOIN Rides_CTE R ON R.user_id =  U.id ORDER BY (travelled_distance)DESC

Q35.
CREATE TABLE Movies(
movie_id INT,
title VARCHAR(50),
PRIMARY KEY (movie_id) )
```

name	travelled_distance
Lee	450
Elvis	450
Bob	317
Jonathan	312
Alex	222
Alice	120
Donald	NULL

Q35. 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.

```
466
467     Q35.
468     CREATE TABLE Movies(
469     movie_id INT,
470     title VARCHAR(50),
471     PRIMARY KEY (movie_id) )
472     INSERT INTO Movies VALUES (1, 'Avengers'),
473     (2, 'Frozen 2'),
474     (3, 'Joker')
475     SELECT * FROM Movies
476     INSERT INTO Movies VALUES (4, 'James')
477     DELETE FROM Movies WHERE movie_id = 4
478
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

movie_id	title
1	Avengers
2	Frozen 2
3	Joker
NULL	NULL

SQL-4 INeuronSQL-5 INeuronSQL-6 INeuronSQL-7 INeuronSQL-8 INeuronSQL-9 INeuronSQL-10

Limit to 1000 rows

```
489     CREATE TABLE MovieRating(
490     movie_id INT,
491     user_id INT,
492     rating INT,
493     created_at DATE,
494     PRIMARY KEY (movie_id,user_id) )
495     INSERT INTO  MovieRating VALUES (1, 1, 3, '2020-01-12'),
496     (1, 2, 4, '2020-02-11'),
497     (1, 3, 2, '2020-02-12'),
498     (1, 4, 1, '2020-01-01'),
499     (2, 1, 5, '2020-02-17'),
500     (2, 2, 2, '2020-02-01'),
501     (2, 3, 2, '2020-03-01'),
502     (3, 1, 3, '2020-02-22'),
503     (3, 2, 4, '2020-02-25')
504     SELECT * FROM MovieRating
505
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

movie_id	user_id	rating	created_at
1	1	3	2020-01-12
1	2	4	2020-02-11
1	3	2	2020-02-12
1	4	1	2020-01-01
2	1	5	2020-02-17
2	2	2	2020-02-01
2	3	2	2020-03-01
3	1	3	2020-02-22
3	2	4	2020-02-25
NULL	NULL	NULL	NULL

```
477     DELETE FROM Movies WHERE movie_id = 4
478
479     CREATE TABLE IF NOT EXISTS Users35(
480     user_id INT,
481     `name` VARCHAR(50),
482     PRIMARY KEY (user_id) )
483     INSERT INTO Users35 VALUES (1, 'Daniel'),
484     (2, 'Monica'),
485     (3, 'Maria'),
486     (4, 'James')
487     SELECT * FROM Users35
488
489     CREATE TABLE MovieRating(
```

Result Grid | Filter Rows: | Edit: | Export/Import:

user_id	name
1	Daniel
2	Monica
3	Maria
4	James
NULL	NULL

```sql
#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.
WITH TotalMovie AS (SELECT user_id,COUNT(movie_id) as TotalMovies_watched FROM MovieRating GROUP BY user_id)
SELECT U.user_id,U.name,TM.TotalMovies_watched FROM Users35 U LEFT JOIN TotalMovie TM ON U.user_id = TM.user_id ORDER BY name ASC LIMIT 1

#Find the movie name with the highest average rating in February 2020. In case of a tie, return
#the lexicographically smaller movie name.
WITH HighAvg AS (SELECT movie_id,AVG(rating) as Avg_Rating FROM MovieRating WHERE created_at BETWEEN '2020-02-01' AND '2020-02-28'
GROUP BY (movie_id) ORDER BY Avg_Rating DESC)
SELECT M.movie_id,M.title,HA.Avg_Rating FROM Movies M LEFT JOIN HighAvg HA ON M.movie_id = HA.movie_id ORDER BY Avg_Rating DESC LIMIT 1
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	name	TotalMovies_watched
1	Daniel	3


```sql
#Find the movie name with the highest average rating in February 2020. In case of a tie, return
#the lexicographically smaller movie name.
WITH HighAvg AS (SELECT movie_id,AVG(rating) as Avg_Rating FROM MovieRating WHERE created_at BETWEEN '2020-02-01' AND '2020-02-28'
GROUP BY (movie_id) ORDER BY Avg_Rating DESC)
SELECT M.movie_id,M.title,HA.Avg_Rating FROM Movies M LEFT JOIN HighAvg HA ON M.movie_id = HA.movie_id ORDER BY Avg_Rating DESC LIMIT 1
```


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

movie_id	title	Avg_Rating
2	Frozen 2	3.5000

Q38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.

```sql
SELECT M.movie_id,M.title,HA.Avg_Rating FROM Movies M LEFT JOIN HighAvg HA ON M.movie_id = HA.movie_id ORDER BY Avg_Rating DESC

Q38.
CREATE TABLE Departments(
id INT,
`name` VARCHAR(100),
PRIMARY KEY (id) )
INSERT INTO Departments VALUES (1, 'Electrical Engineering'),
(7, 'Computer Engineering'),
(13, 'Business Administration')
SELECT * FROM Departments
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name
1	Electrical Engineering
7	Computer Engineering
13	Business Administration
NULL	NULL

```sql

CREATE TABLE IF NOT EXISTS Students (
id INT,
`name` VARCHAR(50),
department_id INT,
PRIMARY KEY (id) )
INSERT INTO Students VALUES (23, 'Alice', 1),
(1, 'Bob', 7),
(5, 'Jennifer', 13),
(2, 'John', 14),
(4, 'Jasmine', 77),
(3, 'Steve', 74),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	department_id
1	Bob	7
2	John	14
3	Steve	74
4	Jasmine	77
5	Jennifer	13
6	Luis	1
7	Daiana	33
8	Jonathan	7
11	Madelynn	1
23	Alice	1
NULL	NULL	NULL

```sql
(6, 'Luis', 1),
(8, 'Jonathan', 7),
(7, 'Daiana', 33),
(11, 'Madelynn', 1)
SELECT * FROM Students
SELECT * FROM Departments

SELECT S.id,D.name as department,S.name FROM Departments D RIGHT JOIN Students S ON D.id = S.department_id WHERE D.name is null

Q39.
CREATE TABLE IF NOT EXISTS Calls(
from id INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	department	name
2	NULL	John
3	NULL	Steve
4	NULL	Jasmine
7	NULL	Daiana

Q39. 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.

```sql
Q39.
CREATE TABLE IF NOT EXISTS Calls(
from_id INT,
to_id INT,
duration INT)
INSERT INTO Calls VALUES (1, 2, 59),
(2, 1, 11),
(1, 3, 20),
(3, 4, 100),
(3, 4, 200),
(3, 4, 200),
(4, 3, 499)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

from_id	to_id	duration
1	2	59
2	1	11
1	3	20
3	4	100
3	4	200
3	4	200
4	3	499

```sql
SELECT * FROM Calls

SELECT from_id AS person1,to_id AS person2,COUNT(duration)AS call_count,SUM(duration) AS total_duration
FROM Calls WHERE (from_id = 1 AND to_id = 2) OR (from_id = 2 AND to_id = 1)
UNION
SELECT from_id AS person1,to_id AS person2,COUNT(duration)AS call_count,SUM(duration) AS total_duration
FROM Calls WHERE (from_id = 1 AND to_id = 3) OR (from_id = 3 AND to_id = 1)
UNION
SELECT from_id AS person1,to_id AS person2,COUNT(duration)AS call_count,SUM(duration) AS total_duration
FROM Calls WHERE (from_id = 3 AND to_id = 4) OR (from_id = 4 AND to_id = 3)

041.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

person1	person2	call_count	total_duration
1	2	2	70
1	3	1	20
3	4	4	999

Q41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse.

```sql
Q41.
CREATE TABLE IF NOT EXISTS Warehouse(
`name` VARCHAR(50),
product_id INT,
units INT,
PRIMARY KEY (name,product_id) )
INSERT INTO Warehouse VALUES ('LCHouse1', 1, 1),
('LCHouse1', 2, 10),
('LCHouse1', 3, 5),
('LCHouse2', 1, 2),
('LCHouse2', 2, 2),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

name	product_id	units
LCHouse1	1	1
LCHouse1	2	10
LCHouse1	3	5
LCHouse2	1	2
LCHouse2	2	2
LCHouse3	4	1
NULL	NULL	NULL

```sql
CREATE TABLE IF NOT EXISTS Products41(
product_id INT,
product_name VARCHAR(50),
Width INT,
`Length` INT,
Height INT,
PRIMARY KEY(product_id) )
INSERT INTO Products41 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)
SELECT * FROM Products41
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	product_name	Width	Length	Height
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
NULL	NULL	NULL	NULL	NULL

```sql
WITH Volume AS ( WITH Warehouse_Products AS (SELECT W.name,W.product_id,P.product_name,P.Width,P.Length,P.Height,W.units FROM Warehouse W
LEFT JOIN Products41 P
ON W.product_id = P.product_id)
SELECT *, Width*Length*Height*units AS volume FROM Warehouse_Products)
SELECT name AS warehouse_name,SUM(volume) AS volume FROM Volume GROUP BY name

Q42.
CREATE TABLE IF NOT EXISTS Sales42(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

warehouse_name	volume
LCHouse1	12250
LCHouse2	20250
LCHouse3	800

Q42. 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.

```sql
Q42.
CREATE TABLE IF NOT EXISTS Sales42(
sale_date DATE,
fruit ENUM ('apples','oranges'),
sold_num INT,
PRIMARY KEY (sale_date,fruit) )
INSERT INTO Sales42 VALUES ('2020-05-01', 'apples', 10),
('2020-05-01', 'oranges', 8),
('2020-05-02', 'apples', 15),
('2020-05-02', 'oranges', 15),
('2020-05-03', 'apples', 20),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

sale_date	fruit	sold_num
2020-05-01	apples	10
2020-05-01	oranges	8
2020-05-02	apples	15
2020-05-02	oranges	15
2020-05-03	apples	20
2020-05-03	oranges	0
2020-05-04	apples	15
2020-05-04	oranges	16
NULL	NULL	NULL

```sql
CREATE VIEW apples AS (SELECT *,sold_num as apple_sold FROM Sales42 WHERE fruit = 'apples')
SELECT * FROM apples
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sale_date	fruit	sold_num	apple_sold
2020-05-01	apples	10	10
2020-05-02	apples	15	15
2020-05-03	apples	20	20
2020-05-04	apples	15	15

```
622
623  CREATE VIEW oranges AS (SELECT *,sold_num as orange_sold FROM Sales42 WHERE fruit = 'oranges')
624  SELECT * FROM oranges
625
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sale_date	fruit	sold_num	orange_sold
2020-05-01	oranges	8	8
2020-05-02	oranges	15	15
2020-05-03	oranges	0	0
2020-05-04	oranges	16	16

Limit to 1000 rows

```
624  SELECT * FROM oranges
625  WITH Fruit_CTE AS (SELECT A.sale_date,A.fruit as apple_fruit,A.apple_sold,O.fruit as orange_fruit,O.orange_sold FROM
626  apples A INNER JOIN oranges O
627  ON A.sale_date = O.sale_date)
628  SELECT sale_date, apple_sold-orange_sold as diff FROM Fruit_CTE
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sale_date	diff
2020-05-01	2
2020-05-02	0
2020-05-03	20
2020-05-04	-1

Q43. 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.

iSQL-4 | INeuronSQL-5 | INeuronSQL-6 | INeuronSQL-7 | INeuronSQL-8 | INeuronSQL-9 | INeuronSQL-10 | INeuronSQL-11 | INeuronSQL-12 | SQL-questions-set1

Limit to 1000 rows

```
630  Q43.
631  SET SQL_SAFE_UPDATES = 0
632  SELECT * FROM Activity
633  UPDATE Activity SET event_date = ('2016-03-02') WHERE games_played = 6
634
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5
NULL	NULL	NULL	NULL

```
634
635  CREATE VIEW player1 AS (SELECT *,Month(event_date) as Month,Day(event_date) as Day FROM Activity WHERE Day(event_date) IN (1,2) LIMIT 2)
636  SELECT * FROM player1
637
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

player_id	device_id	event_date	games_played	Month	Day
1	2	2016-03-01	5	3	1
1	2	2016-03-02	6	3	2

```sql
CREATE VIEW TotalCount AS (SELECT player_id, count(DISTINCT (player_id)) AS total_players FROM Activity)
SELECT * FROM TotalCount
```

player_id	total_players
1	3

```sql
SELECT * FROM TotalCount

WITH fraction AS (SELECT P.player_id,TC.total_players FROM player1 P INNER JOIN TotalCount TC ON P.player_id = TC.player_id)
SELECT player_id/total_players AS fraction FROM fraction LIMIT 1
```

fraction
0.3333

Q44. Write an SQL query to report the managers with at least five direct reports.Return the result table in any order.

```sql
Q44.
CREATE TABLE IF NOT EXISTS Employee44(
id INT,
`name` VARCHAR(50),
department ENUM('A','B'),
manager_id INT,
PRIMARY KEY(id)
)
INSERT INTO Employee44 (id,name,department) VALUES (101, 'John', 'A'),
INSERT INTO Employee44 VALUES (102, 'Dan', 'A', 101),
(103, 'James', 'A', 101),
(104, 'Amy', 'A', 101),
(105, 'Anne', 'A', 101),
(106, 'Ron', 'B', 101)
SELECT * FROM Employee44
```

id	name	department	manager_id
101	John	A	NULL
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

```sql
SELECT name FROM Employee44 WHERE manager_id IS NULL

Q45.
```

name
John

Q45. 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 name FROM Employee44 WHERE manager_id IS NULL

Q45.
CREATE TABLE IF NOT EXISTS Student(
student_id INT,
student_name VARCHAR(30),
gender ENUM('M','F'),
dep_id INT,
PRIMARY KEY (student_id) )
INSERT INTO Student VALUES (1, 'Jack', 'M', 1),
(2, 'Jane', 'F', 1),
(3, 'Mark', 'M', 2)
SELECT * FROM Student
ALTER TABLE Student RENAME COLUMN dep_id to dept_id
```

student_id	student_name	gender	dept_id
1	Jack	M	1
2	Jane	F	1
3	Mark	M	2
NULL	NULL	NULL	NULL

```
CREATE TABLE IF NOT EXISTS Department(
dept_id INT,
dept_name VARCHAR(50),
PRIMARY KEY(dept_id) )
INSERT INTO Department VALUES (1, 'Engineering'),
(2, 'Science'),
(3, 'Law')
SELECT * FROM Department

WITH student_CTE AS (SELECT S.student_id,S.student_name,S.gender,D.dept_name FROM Student S RIGHT JOIN Department D ON S.dept_id = D.dept_id)
SELECT dept_name,count(student_id) AS Number_Students FROM student_CTE GROUP BY dept_name

Q46.
CREATE TABLE IF NOT EXISTS Customer(
customer_id INT,
```

dept_id	dept_name
1	Engineering
2	Science
3	Law
NULL	NULL

```
WITH student_CTE AS (SELECT S.student_id,S.student_name,S.gender,D.dept_name FROM Student S RIGHT JOIN Department D ON S.dept_id = D.dept_id)
SELECT dept_name,count(student_id) AS Number_Students FROM student_CTE GROUP BY dept_name
```

dept_name	Number_Students
Engineering	2
Science	1
Law	0

Q46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.

```
687   Q46.
688   CREATE TABLE IF NOT EXISTS Customer(
689   customer_id INT,
690   product_key INT)
691   INSERT INTO Customer VALUES (1, 5),
692   (2, 6),
693   (3, 5),
694   (3, 6),
695   (1, 6)
696   SELECT * FROM Customer
697
698   CREATE TABLE IF NOT EXISTS Product46(
699   product_key INT,
700   PRIMARY KEY (product_key) )
701   INSERT INTO Product46 VALUES (5),(6)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

```
697
698   CREATE TABLE IF NOT EXISTS Product46(
699   product_key INT,
700   PRIMARY KEY (product_key) )
701   INSERT INTO Product46 VALUES (5),(6)
702   SELECT * FROM Product46
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_key
5
6
NULL

```
703
704   SELECT customer_id FROM Customer WHERE product_key = 5 AND 6
705
706   Q47.
707   CREATE TABLE IF NOT EXISTS Project(
708   project_id INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id
1
3

Q47. 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.

```
705
706     Q47.
707     CREATE TABLE IF NOT EXISTS Project(
708     project_id INT,
709     employee_id INT,
710     PRIMARY KEY(project_id,employee_id) )
711     INSERT INTO Project VALUES (1, 1),
712     (1, 2),
713     (1, 3),
714     (2, 1),
715     (2, 4)
716     SELECT * FROM Project
717     CREATE TABLE IF NOT EXISTS Employee47(
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conte

project_id	employee_id
1	1
1	2
1	3
2	1
2	4
NULL	NULL

```
717     CREATE TABLE IF NOT EXISTS Employee47(
718     employee_id INT,
719     `name` VARCHAR(30),
720     experience_year INT,
721     PRIMARY KEY (employee_id) )
722     INSERT INTO Employee47 VALUES (1, 'Khaled', 3),
723     (2, 'Ali', 2),
724     (3, 'John', 3),
725     (4, 'Doe', 2)
726     SELECT * FROM Employee47
727
728     WITH Experience_employee AS (SELECT * FROM Employee47 WHERE experience_year = 3)
729     SELECT P.project_id,P.employee_id FROM Project P INNER JOIN Experience_employee E ON P.employee_id = E.employee_id
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	name	experience_year
1	Khaled	3
2	Ali	2
3	John	3
4	Doe	2
NULL	NULL	NULL

```
727
728     WITH Experience_employee AS (SELECT * FROM Employee47 WHERE experience_year = 3)
729     SELECT P.project_id,P.employee_id FROM Project P INNER JOIN Experience_employee E ON P.employee_id = E.employee_id
730
731     Q48.
732     # ORDER TABLE HAS NOT GIVEN IN PDF!
733     CREATE TABLE IF NOT EXISTS Books (
734     book_id INT,
735     `name` VARCHAR(100),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

project_id	employee_id
1	1
1	3
2	1

Q49. 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.

```
739
740     Q49.
741   CREATE TABLE IF NOT EXISTS Enrollments(
742     student_id INT,
743     course_id INT,
744     grade INT,
745     PRIMARY KEY (student_id,course_id) )
746     INSERT INTO Enrollments VALUES (2, 2, 95),
747     (2, 3, 95),
748     (1, 1, 90),
749     (1, 2, 99),
750     (3, 1, 80),
751     (3, 2, 75),
```

Result Grid | Filter Rows: | Edit: | Export/Import:

student_id	course_id	grade
1	1	90
1	2	99
2	2	95
2	3	95
3	1	80
3	2	75
3	3	82
NULL	NULL	NULL

```
754
755     (SELECT * FROM Enrollments WHERE student_id =1 ORDER BY grade DESC LIMIT 1)
756     UNION ALL
757     (SELECT * FROM Enrollments WHERE student_id =2 ORDER BY grade DESC LIMIT 1)
758     UNION ALL
759     (SELECT * FROM Enrollments WHERE student_id =3 ORDER BY grade DESC LIMIT 1)
760
761     Q50.
762     #CONFUSION
763
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

Q51. Write an SQL query to report the name, population, and area of the big countries. Return the result table in any order.

```
766     Q51.
767   CREATE TABLE IF NOT EXISTS World (
768     `name` VARCHAR(50),
769     continent VARCHAR(50),
770     `area` INT,
771     population INT,
772     gdp INT,
773     PRIMARY KEY (name) )
774     ALTER TABLE World MODIFY gdp BIGINT
775     INSERT INTO World VALUES ('Afghanistan', 'Asia', 652230, 25500100, 20343000000),
776     ('Albania', 'Europe', 28748, 2831741, 12960000000),
777     ('Algeria', 'Africa', 2381741, 37100000, 188681000000),
778     ('Andorra', 'Europe', 468, 78115, 3712000000),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

name	continent	area	population	gdp
Afghanistan	Asia	652230	25500100	20343000000
Albania	Europe	28748	2831741	12960000000
Algeria	Africa	2381741	37100000	188681000000
Andorra	Europe	468	78115	3712000000
Angola	Africa	1246700	20609294	100990000000
NULL	NULL	NULL	NULL	NULL

```
780     SELECT * FROM World
781     SELECT name,population,area FROM World ORDER BY population DESC LIMIT 2
782
783     Q52.
784     CREATE TABLE Customer52(
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows

name	population	area
Algeria	37100000	2381741
Afghanistan	25500100	652230
NULL	NULL	NULL

Q52. Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.

```
782
783     Q52.
784     CREATE TABLE Customer52(
785     id INT,
786     `name` VARCHAR(50),
787     referee_id INT,
788     PRIMARY KEY (id) )
789     INSERT INTO Customer52 VALUES (3, 'Alex', 2),
790     (5, 'Zack', 1),
791     (6, 'Mark', 2)
792     INSERT INTO Customer52 (id,name) VALUES (1, 'Will'),
793     (2, 'Jane'),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	referee_id
1	Will	NULL
2	Jane	NULL
3	Alex	2
4	Bill	NULL
5	Zack	1
6	Mark	2
NULL	NULL	NULL

```
796     SELECT name FROM Customer52 WHERE referee_id NOT IN (2) OR  referee_id IS NULL
797
798     Q53.
799     CREATE TABLE IF NOT EXISTS Customers53 (
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
Will
Jane
Bill
Zack

Q53. Write an SQL query to report all customers who never order anything.

```
798  Q53.
799  CREATE TABLE IF NOT EXISTS Customers53 (
800  id INT,
801  `name` VARCHAR(50),
802  PRIMARY KEY (id) )
803  INSERT INTO Customers53 VALUES (1, 'Joe'),
804  (2, 'Henry'),
805  (3, 'Sam'),
806  (4, 'Max')
807  SELECT * FROM Customers53
808
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name
1	Joe
2	Henry
3	Sam
4	Max
NULL	NULL

```
809  CREATE TABLE IF NOT EXISTS Orders53(
810  id INT,
811  customer_id INT,
812  PRIMARY KEY (id),
813  FOREIGN KEY (customer_id) REFERENCES Customers53(id) )
814  INSERT INTO Orders53 VALUES (1, 3),
815  (2, 1)
816  SELECT * FROM Orders53
817  SELECT C.name AS Customers FROM Customers53 C LEFT JOIN Orders53 O ON C.id = O.customer_id WHERE customer_id IS NULL
818
819  Q54.
820  SELECT * FROM Employee
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	customer_id
2	1
1	3
NULL	NULL

```
816  SELECT * FROM Orders53
817  SELECT C.name AS Customers FROM Customers53 C LEFT JOIN Orders53 O ON C.id = O.customer_id WHERE customer_id IS NULL
818
819  Q54.
820  SELECT * FROM Employee
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customers
Henry
Max

Q54. Write an SQL query to find the team size of each of the employees.

```
817  SELECT C.name AS Customers FROM Customers53 C LEFT JOIN Orders53 O ON C.id = O.customer_id WHERE customer_id IS NULL
818
819  Q54.
820  SELECT * FROM Employee
821  WITH team_CTE AS (SELECT team_id,count(employee_id) as total FROM Employee GROUP BY team_id)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9
NULL	NULL

```
820     SELECT * FROM Employee
821     WITH team_CTE AS (SELECT team_id,count(employee_id) as total FROM Employee GROUP BY team_id)
822     SELECT E.employee_id,T.total AS team_size FROM Employee E INNER JOIN team_CTE T ON E.team_id = T.team_id
823
824     Q55.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

Q55.Write an SQL query to find the countries where this company can invest.

nSQL-4 INeuronSQL-5 INeuronSQL-6 INeuronSQL-7 INeuronSQL-8 INeuronSQL-9 INeuronSQL-10 INeuronSQL-11 INeuronSQL-12 SQL-ques

Limit to 1000 rows

```
823
824     USE assignment
825     Q55.
826     CREATE TABLE IF NOT EXISTS Person (
827     id INT,
828     `name` VARCHAR(50),
829     phone_number VARCHAR(100),
830     PRIMARY KEY (id) )
831     INSERT INTO Person VALUES (3, 'Jonathan', '051-1234567'),
832     (12, 'Elvis', '051-7654321'),
833     (1, 'Moncef', '212-1234567'),
834     (2, 'Maroua', '212-6523651'),
835     (7, 'Meir', '972-1234567'),
836     (9, 'Rachel', '972-0011100')
837     SELECT * FROM Person
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	phone_number
1	Moncef	212-1234567
2	Maroua	212-6523651
3	Jonathan	051-1234567
7	Meir	972-1234567
9	Rachel	972-0011100
12	Elvis	051-7654321
NULL	NULL	NULL

Person 1

nSQL-4 INeuronSQL-5 INeuronSQL-6 INeuronSQL-7 INeuronSQL-8 INeuronSQL-9 INeuronSQL-10 INeuronSQL-11 INeuronSQL-12 SQL-

Limit to 1000 rows

```
838
839     CREATE TABLE IF NOT EXISTS Country(
840     `name` VARCHAR(50),
841     country_code INT,
842     PRIMARY KEY (country_code) )
843     INSERT INTO Country VALUES ('Peru', 51),
844     ('Israel', 972),
845     ('Morocco', 212),
846     ('Germany', 49),
847     ('Ethiopia', 251)
848     SELECT * FROM Country
849     ALTER TABLE Country RENAME COLUMN name TO Name
850     ALTER TABLE Country MODIFY country_code VARCHAR(50)
851     UPDATE Country SET country_code = '051' WHERE `name` = 'Peru'
852     SET SQL SAFE UPDATES = 0
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Name	country_code
Peru	051
Morocco	212
Ethiopia	251
Germany	49
Israel	972
NULL	NULL

Limit to 1000 rows

```
853
854  CREATE TABLE IF NOT EXISTS Calls55(
855  caller_id INT,
856  callee_id INT,
857  duration INT )
858  SELECT * FROM Calls55
859  INSERT INTO Calls55 VALUES (1, 9, 33),
860  (2, 9, 4),
861  (1, 2, 59),
862  (3, 12, 102),
863  (3, 12, 330),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

caller_id	callee_id	duration
1	9	33
2	9	4
1	2	59
3	12	102
3	12	330
12	3	5
7	9	13
7	1	3
9	7	1
1	7	7

```
896
897  SELECT (2*SUM(duration)/20) AS Global_call FROM Calls55
898
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Global_call
55.7000

```
881
882  (WITH Person_Country AS (WITH Person_CTE AS (SELECT  *,SUBSTR(phone_number,1,3) AS country_codes from person)
883  SELECT P.id,P.Name,P.phone_number,C.name AS Country FROM Person_CTE P INNER JOIN Country C ON P.country_codes = C.country_code)
884  SELECT PC.Country,(SUM(duration) + 102+330+5)/6 AS Avg_call FROM Person_Country PC INNER JOIN Calls55 C ON PC.id = C.Caller_id
885  WHERE caller_id  IN (3,12) OR callee_id IN (3,12))
886  UNION
887  (WITH Person_Country AS (WITH Person_CTE AS (SELECT  *,SUBSTR(phone_number,1,3) AS country_codes from person)
888  SELECT P.id,P.Name,P.phone_number,C.name AS Country FROM Person_CTE P INNER JOIN Country C ON P.country_codes = C.country_code)
889  SELECT PC.Country,(SUM(duration) + 59)/6 AS Avg_call FROM Person_Country PC INNER JOIN Calls55 C ON PC.id = C.Caller_id
890  WHERE caller_id  IN (1,2) OR callee_id IN (1,2))
891  UNION
892  (WITH Person_Country AS (WITH Person_CTE AS (SELECT  *,SUBSTR(phone_number,1,3) AS country_codes from person)
893  SELECT P.id,P.Name,P.phone_number,C.name AS Country FROM Person_CTE P INNER JOIN Country C ON P.country_codes = C.country_code)
894  SELECT PC.Country,(SUM(duration) + 13+1)/8 AS Avg_call FROM Person_Country PC INNER JOIN Calls55 C ON PC.id = C.Caller_id
895  WHERE caller_id  IN (7,9) OR callee_id IN (7,9))
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Country	Avg_call
Peru	145.6667
Morocco	27.5000
Morocco	9.3750

Q56. Write an SQL query to report the device that is first logged in for each player.

```
899  Q56.
900  SELECT * FROM Activity
901  SELECT player_id,device_id,MIN(event_date) AS First_log FROM Activity GROUP BY player_id
902
903  Q57.
```

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5
NULL	NULL	NULL	NULL


```
899  Q56.
900  SELECT * FROM Activity
901  SELECT player_id,device_id,MIN(event_date) AS First_log FROM Activity GROUP BY player_id
902
903  Q57.
```


player_id	device_id	First_log
1	2	2016-03-01
2	3	2017-06-25
3	1	2016-03-02

Q57. Write an SQL query to find the customer_number for the customer who has placed the largest number of orders. The test cases are generated so that exactly one customer will have placed more orders than any other customer.

```
902
903     Q57.
904     CREATE TABLE IF NOT EXISTS Orders57 (
905     order_number INT,
906     customer_number INT,
907     PRIMARY KEY (order_number) )
908     INSERT INTO Orders57 VALUES (1, 1),
909     (2, 2),
910     (3, 3),
911     (4, 3)
912     SELECT * FROM Orders57
913     SELECT customer_number,COUNT(order_number) AS Total_order FROM Orders57 GROUP BY customer_number
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

order_number	customer_number
1	1
2	2
3	3
4	3
NULL	NULL

```
903     Q57.
904     CREATE TABLE IF NOT EXISTS Orders57 (
905     order_number INT,
906     customer_number INT,
907     PRIMARY KEY (order_number) )
908     INSERT INTO Orders57 VALUES (1, 1),
909     (2, 2),
910     (3, 3),
911     (4, 3)
912     SELECT * FROM Orders57
913     SELECT customer_number,COUNT(order_number) AS Total_order FROM Orders57 GROUP BY customer_number
914
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_number	Total_order
1	1
2	1
3	2

Q58. Write an SQL query to report all the consecutive available seats in the cinema.Return the result table ordered by seat_id in ascending order.The test cases are generated so that more than two seats are consecutively available.

```
914
915     Q58.
916     USE assignment
917     CREATE TABLE IF NOT EXISTS Cinema(
918     seat_id INT NOT NULL AUTO_INCREMENT,
919     free BOOL,
920     PRIMARY KEY (seat_id) )
921     INSERT INTO Cinema (free) VALUES  (1),(0),(1),(1),(1)
922     SELECT * FROM Cinema
923     SELECT * FROM Cinema WHERE free = 1 limit 1,3
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

seat_id	free
1	1
2	0
3	1
4	1
5	1
NULL	NULL

```sql
SELECT customer_number,COUNT(order_number) AS Total_order FROM Orders57 GROUP BY customer_number

Q58.
USE assignment
CREATE TABLE IF NOT EXISTS Cinema(
seat_id INT NOT NULL AUTO_INCREMENT,
free BOOL,
PRIMARY KEY (seat_id) )
INSERT INTO Cinema (free) VALUES  (1),(0),(1),(1),(1)
SELECT * FROM Cinema
SELECT * FROM Cinema WHERE free = 1 limit 1,3

Q59.
```

seat_id	free
3	1
4	1
5	1
NULL	NULL

Q59. Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name "RED".

```sql
Q59.
CREATE TABLE IF NOT EXISTS SalesPerson (
sales_id INT,
`name` VARCHAR(50),
salary INT,
commission_rate INT,
hire_date DATE_FORMAT(DATE ,'%M/%D/%Y'),
PRIMARY KEY (sales_id) )
SELECT * FROM SalesPerson
ALTER TABLE SalesPerson MODIFY hire_date VARCHAR(50)
INSERT INTO SalesPerson (sales_id,name,salary,commission_rate,hire_date)  VALUES (2, 'Amy', 12000, 5, '5/1/2010'),
(3, 'Mark', 65000, 12, '12/25/2008'),
```

sales_id	name	salary	commission_rate	hire_date	hire_date_new
1	John	100000	6	4/1/2006	2006-04-01
2	Amy	12000	5	5/1/2010	2010-05-01
3	Mark	65000	12	12/25/2008	2008-12-25
4	Pam	25000	25	1/1/2005	2005-01-01
5	Alex	5000	10	2/3/2007	2007-02-03
NULL	NULL	NULL	NULL	NULL	NULL

SalesPerson 15

```sql
SET SQL_SAFE_UPDATES = 0
alter table SalesPerson add column hire_date_new date after hire_date
update SalesPerson set hire_date_new = str_to_date(hire_date, '%m/%d/%Y')
```

```sql

CREATE TABLE IF NOT EXISTS Company(
com_id INT,
`name` VARCHAR(50),
city VARCHAR(50),
PRIMARY KEY (com_id) )
INSERT INTO Company VALUES (1, 'RED', 'Boston'),
(2, 'ORANGE', 'New York'),
(3, 'YELLOW', 'Boston'),
(4, 'GREEN', 'Austin')
SELECT * FROM Company

CREATE TABLE IF NOT EXISTS Orders59 (
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

com_id	name	city
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin
NULL	NULL	NULL

nSQL-4 INeuronSQL-5 INeuronSQL-6 INeuronSQL-7 INeuronSQL-8 INeuronSQL-9 INeuronSQL-10 INeuronSQL-11 INeuronSQL-12

```sql
CREATE TABLE IF NOT EXISTS Orders59 (
order_id INT,
order_date VARCHAR(50),
com_id INT,
sales_id INT,
amount INT,
PRIMARY KEY (order_id),
FOREIGN KEY (com_id) REFERENCES Company(com_id),
FOREIGN KEY (sales_id) REFERENCES SalesPerson(sales_id) )
INSERT INTO Orders59 VALUES (1, '1/1/2014', 3, 4, 10000),
(2, '2/1/2014', 4, 5, 5000),
(3, '3/1/2014', 1, 1, 50000),
(4, '4/1/2014', 1, 4, 25000)
ALTER TABLE Orders59 ADD COLUMN order_date_new DATE AFTER order_date
UPDATE Orders59 SET order_date_new = str_to_date(order_date, '%m/%d/%Y')
SELECT * FROM Orders59
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

order_id	order_date	order_date_new	com_id	sales_id	amount
1	1/1/2014	2014-01-01	3	4	10000
2	2/1/2014	2014-02-01	4	5	5000
3	3/1/2014	2014-03-01	1	1	50000
4	4/1/2014	2014-04-01	1	4	25000
NULL	NULL	NULL	NULL	NULL	NULL

```sql

WITH Company_Order AS (SELECT C.com_id,C.name AS Name,O.order_id,O.sales_id FROM Company C LEFT JOIN Orders59 O ON C.com_id = O.com_id)
SELECT SP.name,CO.Name FROM SalesPerson SP LEFT JOIN Company_Order CO ON SP.sales_id = CO.sales_id
WHERE CO.Name IS NULL OR CO.Name NOT IN ('RED','YELLOW')

Q60.
CREATE TABLE IF NOT EXISTS Triangle(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	Name
Amy	NULL
Mark	NULL
Alex	GREEN

Q60. Write an SQL query to report for every three line segments whether they can form a triangle.

```
975
976     Q60.
977     CREATE TABLE IF NOT EXISTS Triangle(
978     x INT,
979     y INT,
980     z INT,
981     PRIMARY KEY (x,y,z) )
982     INSERT INTO Triangle VALUES (13,15,30),(10,20,15)
983     SELECT * FROM Triangle
984
985     Q61.
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

x	y	z
10	20	15
13	15	30
NULL	NULL	NULL

Q62. Write a SQL query for a report that provides the pairs (actor_id, director_id) where the actor has cooperated with the director at least three times.

Limit to 1000 rows

```
987
988     Q62.
989     CREATE TABLE IF NOT EXISTS ActorDirector (
990     actor_id INT,
991     director_id INT,
992     `timestamp` INT,
993     PRIMARY KEY (timestamp) )
994     INSERT INTO ActorDirector VALUES (1, 1, 0),
995     (1, 1, 1),
996     (1, 1, 2),
997     (1, 2, 3),
998     (1, 2, 4),
999     (2, 1, 5),
1000    (2, 1, 6)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6
NULL	NULL	NULL

```
1003
1004    (SELECT actor_id,director_id,count(director_id) AS Cooperated FROM ActorDirector WHERE actor_id = 1 and director_id = 2)
1005    UNION ALL
1006    (SELECT actor_id,director_id,count(director_id) AS Cooperated FROM ActorDirector WHERE actor_id = 2 and director_id = 1)
1007    UNION ALL
1008    (SELECT actor_id,director_id,count(director_id) AS Cooperated FROM ActorDirector WHERE actor_id = 1 and director_id = 1)
1009    UNION ALL
1010    (SELECT actor_id,director_id,count(director_id) AS Cooperated FROM ActorDirector WHERE actor_id = 2 and director_id = 2)
1011
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

actor_id	director_id	Cooperated
1	2	2
2	1	2
1	1	3
NULL	NULL	0

Q63. Write an SQL query that reports the product_name, year, and price for each sale_id in the Sales table.

```
1012
1013    Q63.
1014    CREATE TABLE IF NOT EXISTS Sales63(
1015    sale_id INT,
1016    product_id INT,
1017    `year` INT,
1018    quantity INT,
1019    price INT,
1020    PRIMARY KEY (sale_id,year) )
1021    ALTER TABLE Sales63 ADD FOREIGN KEY (product_id) REFERENCES Product63(product_id)
1022    INSERT INTO Sales63 VALUES (1, 100, 2008, 10, 5000),
1023    (2, 100, 2009, 12, 5000),
1024    (7, 200, 2011, 15, 9000)
1025    SELECT * FROM Sales63
1026
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000
NULL	NULL	NULL	NULL	NULL

```
26
27    CREATE TABLE IF NOT EXISTS Product63(
28    product_id INT,
29    product_name VARCHAR(50),
30    PRIMARY KEY (product_id) )
31    INSERT INTO Product63 VALUES (100, 'Nokia'),
32    (200, 'Apple'),
33    (300, 'Samsung')
34    SELECT * FROM Product63
35    SELECT * FROM Sales63
36
37    SELECT S.sale_id,S.product_id,P.product_name,S.year,S.pric
38
```

esult Grid | Filter Rows: | Edit: | Export/Import:

product_id	product_name
100	Nokia
200	Apple
300	Samsung
NULL	NULL

```
1036
1037    SELECT S.sale_id,S.product_id,P.product_name,S.year,S.price FROM Sales63 S LEFT JOIN Product63 P ON S.product_id = P.product_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sale_id	product_id	product_name	year	price
1	100	Nokia	2008	5000
2	100	Nokia	2009	5000
7	200	Apple	2011	9000

Q64. Write an SQL query that reports the average experience years of all the employees for each project, rounded to 2 digits.

```
1039    Q64.
1040    SELECT * FROM Project
1041    CREATE TABLE IF NOT EXISTS Employee64(
1042    employee_id INT,
1043    `name` VARCHAR(50),
1044    experience_years INT,
1045    PRIMARY KEY (employee_id) )
1046    INSERT INTO Employee64 VALUES (1, 'Khaled', 3),
1047    (2, 'Ali', 2),
1048    (3, 'John', 1),
1049    (4, 'Doe', 2)
1050
1051    (WITH Project_Employee AS (SELECT E.employee_id,E.name,P.project_id,E.experience_years FROM Project P
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4
NULL	NULL

```
1041    CREATE TABLE IF NOT EXISTS Employee64(
1042    employee_id INT,
1043    `name` VARCHAR(50),
1044    experience_years INT,
1045    PRIMARY KEY (employee_id) )
1046    INSERT INTO Employee64 VALUES (1, 'Khaled', 3),
1047    (2, 'Ali', 2),
1048    (3, 'John', 1),
1049    (4, 'Doe', 2)
1050    SELECT * FROM Employee64
1051
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wra

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2
NULL	NULL	NULL

```
1051
1052    (WITH Project_Employee AS (SELECT E.employee_id,E.name,P.project_id,E.experience_years FROM Project P
1053    INNER JOIN Employee64 E ON P.employee_id = E.employee_id)
1054    SELECT project_id,SUM(experience_years)/COUNT(project_id) as Avg_years FROM Project_Employee WHERE project_id = 1)
1055    UNION
1056    (WITH Project_Employee AS (SELECT E.employee_id,E.name,P.project_id,E.experience_years FROM Project P
1057    INNER JOIN Employee64 E ON P.employee_id = E.employee_id)
1058    SELECT project_id,SUM(experience_years)/COUNT(project_id) as Avg_years FROM Project_Employee WHERE project_id = 2)
1059
1060    Q65.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

project_id	Avg_years
1	2.0000
2	2.5000

Q65. Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all.

```
1059
1060    Q65.
1061    SELECT * FROM product
1062    SELECT * FROM sales
1063    SELECT seller_id,SUM(price)AS Total_price FROM sales GROUP BY seller_id ORDER BY Total_price DESC
1064
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400
NULL	NULL	NULL

```
1058    SELECT project_id,SUM(experience_years)/COUNT(project_id) as Avg_years FROM Project_Employee WHERE project_id = 2)
1059
1060    Q65.
1061    SELECT * FROM product
1062    SELECT * FROM sales
1063    SELECT seller_id,SUM(price)AS Total_price FROM sales GROUP BY seller_id ORDER BY Total_price DESC
1064
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

```
59
60      Q65.
61      SELECT * FROM product
62      SELECT * FROM sales
63      SELECT seller_id,SUM(price)AS Total_price FROM sales GROUP BY seller_id ORDER BY Total_price DESC
64
65      Q66
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

seller_id	Total_price
1	2800
3	2800
2	800

Q66. Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

```
1064
1065    Q66.
1066    WITH product_sales AS (SELECT P.product_id,P.product_name,P.unit_price,S.seller_id,S.buyer_id,S.sale_date,S.quantity,S.price
1067    FROM Product P INNER JOIN Sales S
1068    ON P.product_id = S.product_id)
1069    SELECT * FROM product_sales WHERE product_name = 'S8'
1070
1071    Q67
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_id	product_name	unit_price	seller_id	buyer_id	sale_date	quantity	price
1	S8	1000	1	1	2019-01-21	2	2000

Q67. Write an SQL query to compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average_amount should be rounded to two decimal places.

```
1070
1071    Q67.
1072    CREATE TABLE IF NOT EXISTS Customer67 (
1073    customer_id INT,
1074    `name` VARCHAR(50),
1075    visited_on DATE,
1076    amount INT,
1077    PRIMARY KEY (customer_id,visited_on) )
1078    INSERT INTO Customer67 VALUES (1, 'Jhon', '2019-01-01', 100),
1079    (2, 'Daniel', '2019-01-02', 110),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
1	Jhon	2019-01-10	130
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
3	Jade	2019-01-10	150
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140
7	Anna	2019-01-07	150
8	Maria	2019-01-08	80
9	Jaze	2019-01-09	110
NULL	NULL	NULL	NULL

Customer67 35 ×

```
1090
1091    WITH Avg_restaurant AS ((SELECT MAX(visited_on) AS visited_on ,SUM(amount) AS amount,ROUND(SUM(amount)/7,2) AS Avg_amount,
1092    DAY(visited_on) AS Visited_Day
1093    FROM Customer67 WHERE DAY(visited_on) BETWEEN 1 AND 7)
1094    UNION ALL
1095    (SELECT MAX(visited_on) AS visited_on ,SUM(amount) AS amount,ROUND(SUM(amount)/7,2) AS Avg_amount,DAY(visited_on) AS Visited_Day
1096    FROM Customer67 WHERE DAY(visited_on) BETWEEN 2 AND 8)
1097    UNION ALL
1098    (SELECT MAX(visited_on) AS visited_on ,SUM(amount) AS amount,ROUND(SUM(amount)/7,2) AS Avg_amount,DAY(visited_on) AS Visited_Day
1099    FROM Customer67 WHERE DAY(visited_on) BETWEEN 3 AND 9)
1100    UNION ALL
1101    (SELECT MAX(visited_on) AS visited_on ,SUM(amount) AS amount,ROUND(SUM(amount)/7,2) AS Avg_amount,DAY(visited_on) AS Visited_Day
1102    FROM Customer67 WHERE DAY(visited_on) BETWEEN 4 AND 10))
1103    SELECT visited_on,amount,Avg_amount FROM Avg_restaurant
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

visited_on	amount	Avg_amount
2019-01-07	860	122.86
2019-01-08	840	120.00
2019-01-09	840	120.00
2019-01-10	1000	142.86

Result 37 ×

Q68. Write an SQL query to find the total score for each gender on each day.

nSQL-4 | INeuronSQL-5 | INeuronSQL-6 | INeuronSQL-7 | INeuronSQL-8 | INeuronSQL-9 | INeuronSQL-10 | INeuronSQL-11 | INeuronSQL-12 | SQL-questions-set1*

Limit to 1000 rows

```
1105    Q68. USE assignment
1106    CREATE TABLE IF NOT EXISTS Scores(
1107    player_name VARCHAR(50),
1108    gender ENUM ('M','F'),
1109    `day` DATE,
1110    score_points INT,
1111    PRIMARY KEY (gender,day) )
1112    INSERT INTO Scores VALUES ('Aron', 'F', '2020-01-01', 17),
1113    ('Alice', 'F', '2020-01-07', 23),
1114    ('Bajrang', 'M', '2020-01-07', 7),
1115    ('Khali', 'M', '2019-12-25', 11),
1116    ('Slaman', 'M', '2019-12-30', 13),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

player_name	gender	day	score_points
Jose	M	2019-12-18	2
Khali	M	2019-12-25	11
Slaman	M	2019-12-30	13
Joe	M	2019-12-31	3
Bajrang	M	2020-01-07	7
Priyanka	F	2019-12-30	17
Priya	F	2019-12-31	23
Aron	F	2020-01-01	17
Alice	F	2020-01-07	23
NULL	NULL	NULL	NULL

```
1122
1123    SELECT gender, day, SUM(score_points) OVER(PARTITION BY gender ORDER BY day) as total FROM Scores
1124
1125    Q69.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

gender	day	total
M	2019-12-18	2
M	2019-12-25	13
M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80

Q70. Write an SQL query to find the number of times each student attended each exam.

Limit to 1000 rows

```
1129    Q70.
1130    CREATE TABLE IF NOT EXISTS Students70(
1131    student_id INT,
1132    student_name VARCHAR(50),
1133    PRIMARY KEY (student_id) )
1134    INSERT INTO Students70 VALUES (1, 'Alice'),
1135    (2, 'Bob'),
1136    (13, 'John'),
1137    (6, 'Alex')
1138    SELECT *FROM Students70
1139
1140    CREATE TABLE IF NOT EXISTS Subjects(
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

student_id	student_name
1	Alice
2	Bob
6	Alex
13	John
NULL	NULL

```sql
1139
1140  CREATE TABLE IF NOT EXISTS Subjects(
1141  subject_name VARCHAR(50),
1142  PRIMARY KEY(subject_name) )
1143  INSERT INTO Subjects VALUES ('Math'),
1144  ('Physics'),
1145  ('Programming')
1146  SELECT * FROM Subjects
1147
1148  CREATE TABLE IF NOT EXISTS Examinations (
1149  student_id INT,
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

subject_name
Math
Physics
Programming
NULL

```sql
1147
1148  CREATE TABLE IF NOT EXISTS Examinations (
1149  student_id INT,
1150  suject_name VARCHAR(50) )
1151  ALTER TABLE Examinations RENAME COLUMN suject_name TO subject_name
1152  INSERT INTO Examinations VALUES (1, 'Math'),
1153  (1, 'Physics'),
1154  (1, 'Programming'),
1155  (2, 'Programming'),
1156  (1, 'Physics'),
1157  (1, 'Math'),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	subject_name
1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

```sql
1164
1165  CREATE VIEW Student_Subject AS (SELECT * FROM Students70 S7 CROSS JOIN Subjects S)
1166  SELECT * FROM Student_Subject
1167  CREATE VIEW Student_Exam AS (SELECT student_id,subject_name,COUNT(*) AS attended_exam FROM Examinations GROUP BY 1,2 ORDER BY 1)
1168
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	student_name	subject_name
1	Alice	Programming
1	Alice	Physics
1	Alice	Math
2	Bob	Programming
2	Bob	Physics
2	Bob	Math
6	Alex	Programming
6	Alex	Physics
6	Alex	Math
13	John	Programming
13	John	Physics
13	John	Math

```sql
CREATE VIEW Student_Exam AS (SELECT student_id,subject_name,COUNT(*) AS attended_exam FROM Examinations GROUP BY 1,2 ORDER BY 1)
SELECT * FROM Student_Exam
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

student_id	subject_name	attended_exam
1	Math	3
1	Physics	2
1	Programming	1
2	Math	1
2	Programming	1
13	Math	1
13	Physics	1
13	Programming	1

Q71. Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company.

```sql
Q71.
CREATE TABLE IF NOT EXISTS Employees71(
employee_id INT,
employee_name VARCHAR(50),
manager_id INT,
PRIMARY KEY (employee_id) )
INSERT INTO Employees71 VALUES (1, 'Boss', 1),
(3, 'Alice', 3),
(2, 'Bob', 1),
(4, 'Daniel', 2)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cont

employee_id	employee_name	manager_id
1	Boss	1
2	Bob	1
3	Alice	3
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1
NULL	NULL	NULL

Q72. Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.

```
1185
1186    Q72.
1187    CREATE TABLE IF NOT EXISTS Transactions(
1188    id INT,
1189    country VARCHAR(50),
1190    state ENUM("approved", "declined"),
1191    amount INT,
1192    trans_date DATE,
1193    PRIMARY KEY (id) )
1194    INSERT INTO Transactions VALUES (121, 'US', 'approved', 1000, '2018-12-18'),
1195    (122, 'US', 'declined', 2000, '2018-12-19'),
1196    (123, 'US', 'approved', 2000, '2019-01-01'),
1197    (124, 'DE', 'approved', 2000, '2019-01-07')
1198    SELECT * FROM Transactions
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07
NULL	NULL	NULL	NULL	NULL

```
1199
1200    CREATE VIEW approved AS (SELECT SUBSTR(trans_date, 1,7) AS Month,country,COUNT(state) AS trans_count,SUM(amount) AS trans_total_amount,
1201    amount AS approved_total_amount FROM Transactions GROUP BY Month,country)
1202    SELECT * FROM approved
1203
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Month	country	trans_count	trans_total_amount	approved_total_amount
2018-12	US	2	3000	1000
2019-01	US	1	2000	2000
2019-01	DE	1	2000	2000

```
1206
1207    CREATE VIEW approved1 AS (SELECT country,trans_date,COUNT(state) AS approved_count FROM Transactions WHERE state = 'approved'
1208    GROUP BY country,trans_date)
1209    SELECT * FROM approved1
1210
1211    Q73.
1212    CREATE TABLE IF NOT EXISTS Actions(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

country	trans_date	approved_count
US	2018-12-18	1
US	2019-01-01	1
DE	2019-01-07	1

```
1203
1204    SELECT A.Month,A.country,A.trans_count,A1.approved_count,A.trans_total_amount,A.approved_total_amount
1205    FROM approved A INNER JOIN approved1 A1 ON A.country = A1.country GROUP BY country,trans_count
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Month	country	trans_count	approved_count	trans_total_amount	approved_total_amount
2018-12	US	2	1	3000	1000
2019-01	US	1	1	2000	2000
2019-01	DE	1	1	2000	2000

Q73. Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.

```
1210
1211    Q73.
1212    CREATE TABLE IF NOT EXISTS Actions(
1213    user_id INT,
1214    post_id INT,
1215    action_date DATE,
1216    `action` ENUM('view', 'like', 'reaction', 'comment', 'report', 'share'),
1217    extra VARCHAR(50) )
1218    INSERT INTO Actions VALUES (1, 1, '2019-07-01', 'view', 'null'),
1219    (1, 1, '2019-07-01', 'like', 'null'),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	post_id	action_date	action	extra
1	1	2019-07-01	view	null
1	1	2019-07-01	like	null
1	1	2019-07-01	share	null
2	2	2019-07-04	view	null
2	2	2019-07-04	report	spam
3	4	2019-07-04	view	null
3	4	2019-07-04	report	spam
4	3	2019-07-02	view	null
4	3	2019-07-02	report	spam
5	2	2019-07-03	view	null
5	2	2019-07-03	report	racism
5	5	2019-07-03	view	null
5	5	2019-07-03	report	racism

Actions 64 ×

```
1232
1233    CREATE TABLE IF NOT EXISTS Removals(
1234    post_id INT,
1235    remove_date DATE,
1236    PRIMARY KEY (post_id) )
1237    INSERT INTO Removals VALUES (2, '2019-07-20'),
1238    (3, '2019-07-18')
1239    SELECT * FROM Removals
1240        SELECT * FROM Actions WHERE extra = 'spam'
1241
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

post_id	remove_date
2	2019-07-20
3	2019-07-18
NULL	NULL

```
1240        SELECT * FROM Actions WHERE extra = 'spam'
1241
1242    Q76.
1243    CREATE TABLE IF NOT EXISTS Salaries(
1244    company_id INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	post_id	action_date	action	extra
2	2	2019-07-04	report	spam
3	4	2019-07-04	report	spam
4	3	2019-07-02	report	spam

Q76. Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the nearest integer.

```
1242    Q76.
1243    CREATE TABLE IF NOT EXISTS Salaries(
1244    company_id INT,
1245    employee_id INT,
1246    employee_name VARCHAR(50),
1247    salary INT,
1248    PRIMARY KEY (company_id,employee_id) )
1249    INSERT INTO Salaries VALUES (1, 1, 'Tony', 2000),
1250    (1, 2, 'Pronub', 21300),
1251    (1, 3, 'Tyrrox', 10800),
1252    (2, 1, 'Pam', 300),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

company_id	employee_id	employee_name	salary
1	1	Tony	2000
1	2	Pronub	21300
1	3	Tyrrox	10800
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	2	Ognjen	2200
3	7	Bocaben	100
3	13	Nyan Cat	3300
3	15	Morning Cat	7777
NULL	NULL	NULL	NULL

```
1259    SELECT * FROM Salaries
1260    (SELECT company_id,employee_id,employee_name,salary FROM Salaries WHERE salary < 1000)
1261    UNION ALL
1262    (SELECT company_id,employee_id,employee_name,ROUND(salary-(24/100)*salary ,0) AS Tax_cutoff FROM Salaries WHERE salary BETWEEN 1000 AND 10000)
1263    UNION ALL
1264    (SELECT  company_id,employee_id,employee_name,ROUND(salary-(49/100)*salary ,0) AS Tax_cutoff FROM Salaries WHERE salary >10000)
1265    ORDER BY company_id
1266
1267    Q77.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_id	employee_id	employee_name	salary
1	1	Tony	1520
1	2	Pronub	10863
1	3	Tyrrox	5508
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	100
3	2	Ognjen	1672
3	13	Nyan Cat	2508
3	15	Morning Cat	5911

Result Grid | Form Editor | Field Types

Result 68 x | Read Only

Q77. Write an SQL query to evaluate the boolean expressions in Expressions table.

```
1266
1267    Q77.
1268    CREATE TABLE IF NOT EXISTS `Variables`(
1269    `name` VARCHAR(50),
1270    `value` INT,
1271    PRIMARY KEY (name) )
1272    INSERT INTO variables VALUES ('x', 66),
1273    ('y', 77)
1274    SELECT * FROM variables
1275
1276    CREATE TABLE IF NOT EXISTS Expressions(
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

name	value
x	66
y	77
NULL	NULL

```
1275
1276    CREATE TABLE IF NOT EXISTS Expressions(
1277    left_operand VARCHAR(5),
1278    operator ENUM('<', '>', '='),
1279    right_operand VARCHAR(5),
1280    PRIMARY KEY (left_operand, operator, right_operand) )
1281    INSERT INTO Expressions VALUES ('x', '>', 'y'),
1282    ('x', '<', 'y'),
1283    ('x', '=', 'y'),
1284    ('y', '>', 'x'),
1285    ('y', '<', 'x'),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

left_operand	operator	right_operand
x	<	y
x	>	y
x	=	x
x	=	y
y	<	x
y	>	x
NULL	NULL	NULL

Q79. Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in alphabetical order.

```
1311    Q79.
1312    CREATE TABLE IF NOT EXISTS Employee79(
1313    employee_id INT,
1314    `name` VARCHAR(50),
1315    months INT,
1316    salary INT )
1317    INSERT INTO Employee79 VALUES (12228, 'Rose', 15, 1968),
1318    (33645, 'Angela', 1, 3443),
1319    (45692, 'Frank', 17, 1608),
1320    (56118, 'Patrick', 7, 1345),
1321    (59725, 'Lisa', 11, 2330),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

Employee79 72 ×

```
1328    SELECT * FROM Employee79 ORDER BY name
1329
1330    USE assignment
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	name	months	salary
33645	Angela	1	3443
78454	Bonnie	8	1771
45692	Frank	17	1608
99989	Joe	9	3573
74197	Kimberly	16	4372
59725	Lisa	11	2330
83565	Michael	6	2017
56118	Patrick	7	1345
12228	Rose	15	1968
98607	Todd	5	3396

Q80. Write a query to obtain the **year-on-year growth rate** for the total spend of each product for each year.

```
1332    Q80.
1333    CREATE TABLE IF NOT EXISTS  user_transactions(
1334    transaction_id INT,
1335    product_id INT,
1336    spend DECIMAL,
1337    transaction_date DATETIME )
1338    ALTER TABLE user_transactions MODIFY transaction_date VARCHAR(50)
1339    ALTER TABLE user_transactions MODIFY spend DECIMAL(50,2)
1340    INSERT INTO user_transactions VALUES (1341, 123424, 1500.60, '12/31/2019 12:00:00')
1341    INSERT INTO user_transactions VALUES (1423, 123424, 1000.20, '12/31/2020 12:00:00'),
1342    (1623, 123424, 1246.44, '12/31/2021 12:00:00'),
1343    (1322, 123424, 2145.32, '12/31/2022 12:00:00')
1344    SELECT * FROM user_transactions
1345    ALTER TABLE user_transactions ADD COLUMN transaction_date_new DATETIME
1346    SET SQL_SAFE_UPDATES = 0
1347    UPDATE user_transactions SET transaction_date_new = STR_TO_DATE(transaction_date, '%m/%d/%Y %H:%i:%s')
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	product_id	spend	transaction_date	transaction_date_new
1341	123424	1500.60	12/31/2019 12:00:00	2019-12-31 12:00:00
1423	123424	1000.20	12/31/2020 12:00:00	2020-12-31 12:00:00
1623	123424	1246.44	12/31/2021 12:00:00	2021-12-31 12:00:00
1322	123424	2145.32	12/31/2022 12:00:00	2022-12-31 12:00:00

```
1348
1349    SELECT SUBSTR(transaction_date_new, 1,4)AS Year,product_id,spend AS curr_year_spend,
1350    LAG(spend) OVER(ORDER BY transaction_date_new ) AS prev_year_spend,
1351    ROUND((LAG(spend) OVER(ORDER BY transaction_date_new )-spend),2) AS YOY_rate FROM user_transactions
1352
1353    Q81.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Year	product_id	curr_year_spend	prev_year_spend	YOY_rate
2019	123424	1500.60	NULL	NULL
2020	123424	1000.20	1500.60	500.40
2021	123424	1246.44	1000.20	-246.24
2022	123424	2145.32	1246.44	-898.88

Q81. Write a SQL query to find the number of prime and non-prime items that can be stored in the 500,000 square feet warehouse. Output the item type and number of items to be stocked.

```
1352
1353    Q81.
1354    CREATE TABLE IF NOT EXISTS inventory(
1355    item_id INT,
1356    item_type VARCHAR(50),
1357    item_category VARCHAR(50),
1358    square_footage DECIMAL(50,2) )
1359    INSERT INTO inventory VALUES (1374, 'prime_eligible', 'mini refrigerator', 68.00),
1360    (4245, 'not_prime standing', 'lamp', 26.40),
1361    (2452, 'prime_eligible', 'television', 85.00),
1362    (3255, 'not_prime side', 'table', 22.60),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

item_id	item_type	item_category	square_footage
1374	prime_eligible	mini refrigerator	68.00
4245	not_prime	standing lamp	26.40
2452	prime_eligible	television	85.00
3255	not_prime	side table	22.60
1672	prime_eligible	laptop	8.50

```
1371    SELECT item_type,COUNT(item_category) AS item_count,SUM(square_footage) AS Total_ FROM inventory GROUP BY item_type
1372
1373    Q82.
1374    CREATE TABLE user_actions(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

item_type	item_count	Total_
prime_eligible	3	161.50
not_prime	2	49.00

Q82. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

```
1373    Q82.
1374    CREATE TABLE user_actions(
1375    user_id INT,
1376    event_id INT,
1377    event_type VARCHAR(50),
1378    even_date VARCHAR(50) )
1379    INSERT INTO user_actions VALUES (445, 7765, 'sign-in', '05/31/2022 12:00:00'),
1380    (742, 6458, 'sign-in', '06/03/2022 12:00:00'),
1381    (445, 3634, 'like', '06/05/2022 12:00:00'),
1382    (742, 1374, 'comment', '06/05/2022 12:00:00'),
1383    (648, 3124, 'like', '06/18/2022 12:00:00')
1384    ALTER TABLE user_actions ADD COLUMN even_date_new DATETIME
1385    SET SQL_SAFE_UPDATES = 0
1386    UPDATE user_actions SET even_date_new = STR_TO_DATE(even_date, '%m/%d/%Y %H:%i:%s')
1387    SELECT * FROM user_actions
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	event_id	event_type	even_date	even_date_new
445	7765	sign-in	05/31/2022 12:00:00	2022-05-31 12:00:00
742	6458	sign-in	06/03/2022 12:00:00	2022-06-03 12:00:00
445	3634	like	06/05/2022 12:00:00	2022-06-05 12:00:00
742	1374	comment	06/05/2022 12:00:00	2022-06-05 12:00:00
648	3124	like	06/18/2022 12:00:00	2022-06-18 12:00:00

```
1388
1389    SELECT user_id,Month(even_date_new) AS Month,COUNT(user_id) AS Max_Active
1390    FROM user_actions WHERE Month(even_date_new) = 6 GROUP BY user_id
1391
1392    Q83.
1393
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	Month	Max_Active
742	6	2
445	6	1
648	6	1

Q83. **Write a query to report the median of searches made by a user. Round the median to one decimal point.**

```
1392    Q83.
1393    CREATE TABLE IF NOT EXISTS advertiser(
1394    user_id VARCHAR(50),
1395    `status` VARCHAR(50) )
1396    INSERT INTO advertiser VALUES ('bing', 'NEW'),
1397    ('yahoo', 'NEW'),
1398    ('alibaba', 'EXISTING')
1399    SELECT * FROM advertiser
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	status
bing	NEW
yahoo	NEW
alibaba	EXISTING

```
1401    CREATE TABLE daily_pay(
1402    user_id VARCHAR(50),
1403    paid DECIMAL(10,2) )
1404    INSERT INTO daily_pay VALUES ('yahoo', 45.00),
1405    ('alibaba', 100.00),
1406    ('target', 13.00)
1407    SELECT * FROM daily_pay
1408
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	paid
yahoo	45.00
alibaba	100.00
target	13.00

```
1411
1412    SELECT A.user_id,IF(DP.paid IS NULL ,'CHURN','EXISTING') AS new_status FROM advertiser A LEFT JOIN daily_pay DP ON A.user_id = DP.user_id
1413
1414
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	new_status
bing	CHURN
yahoo	EXISTING
alibaba	EXISTING

```
1408
1409    SELECT A.user_id,DP.paid,IF(DP.paid IS NULL ,'CHURN','EXISTING') AS new_status FROM advertiser A LEFT JOIN daily_pay DP
1410    ON A.user_id = DP.user_id
1411
1412    SELECT A.user_id,IF(DP.paid IS NULL ,'CHURN','EXISTING') AS new_status FROM advertiser A LEFT JOIN daily_pay DP ON A.user_id = DP.use
1413
1414
1415    Q85.
1416
1417
1418    Q86.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	paid	new_status
bing	NULL	CHURN
yahoo	45.00	EXISTING
alibaba	100.00	EXISTING

Q86.

```sql
Q86.
CREATE TABLE IF NOT EXISTS transactions86(
transaction_id INT,
merchant_id INT,
credit_card_id INT,
amount INT,
transaction_timestamp VARCHAR(50) )
INSERT INTO transactions86 VALUES (1, 101, 1, 100, '09/25/2022 12:00:00'),
(2, 101, 1, 100, '09/25/2022 12:08:00'),
(3, 101, 1, 100, '09/25/2022 12:28:00'),
(4, 102, 2, 300, '09/25/2022 12:00:00'),
(6, 102, 2, 400, '09/25/2022 14:00:00')
ALTER TABLE transactions86 ADD COLUMN transaction_timestamp_new DATETIME
UPDATE transactions86 SET transaction_timestamp_new = STR_TO_DATE(transaction_timestamp, '%m/%d/%Y %H:%i:%s')
SELECT * FROM transactions86
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	merchant_id	credit_card_id	amount	transaction_timestamp	transaction_timestamp_new
1	101	1	100	09/25/2022 12:00:00	2022-09-25 12:00:00
2	101	1	100	09/25/2022 12:08:00	2022-09-25 12:08:00
3	101	1	100	09/25/2022 12:28:00	2022-09-25 12:28:00
4	102	2	300	09/25/2022 12:00:00	2022-09-25 12:00:00
6	102	2	400	09/25/2022 14:00:00	2022-09-25 14:00:00

Q87. Write a query to find the bad experience rate in the first 14 days for new users who signed up in June 2022. Output the percentage of bad experience rounded to 2 decimal places

nSQL-4 | INeuronSQL-5 | INeuronSQL-6 | INeuronSQL-7 | INeuronSQL-8 | INeuronSQL-9 | INeuronSQL-10 | INeuronSQL-11 | INeuronSQL-12 | SQL-questions-set1*

Limit to 1000 rows

```sql
Q87.
CREATE TABLE IF NOT EXISTS orders87(
order_id INT,
customer_id INT,
trip_id INT,
`status` VARCHAR(50),
order_timestamp timestamp )
ALTER TABLE orders87 MODIFY order_timestamp VARCHAR(50)
INSERT INTO orders87 VALUES (727424, 8472, 100463, 'completed successfully', '06/05/2022 09:12:00')
INSERT INTO orders87 VALUES (242513, 2341, 100482, 'completed incorrectly', '06/05/2022 14:40:00'),
(141367, 1314, 100362, 'completed incorrectly', '06/07/2022 15:03:00'),
(582193, 5421, 100657, 'never_received', '07/07/2022 15:22:00'),
(253613, 1314, 100213, 'completed successfully', '06/12/2022 13:43:00')
SELECT * FROM orders87
CREATE TABLE IF NOT EXISTS trips(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	customer_id	trip_id	status	order_timestamp
727424	8472	100463	completed successfully	06/05/2022 09:12:00
242513	2341	100482	completed incorrectly	06/05/2022 14:40:00
141367	1314	100362	completed incorrectly	06/07/2022 15:03:00
582193	5421	100657	never_received	07/07/2022 15:22:00
253613	1314	100213	completed successfully	06/12/2022 13:43:00

```sql
dasher_id INT,
trip_id INT,
estimated_delivery_timestamp VARCHAR(50),
actual_delivery_timestamp VARCHAR(50) )
INSERT INTO trips VALUES (101, 100463, '06/05/2022 09:42:00', '06/05/2022 09:38:00'),
(102, 100482, '06/05/2022 15:10:00', '06/05/2022 15:46:00'),
(101, 100362, '06/07/2022 15:33:00', '06/07/2022 16:45:00'),
(102, 100657, '07/07/2022 15:52:00',  '-'),
(103, 100213, '06/12/2022 14:13:00', '06/12/2022 14:10:00')
SELECT * FROM trips
CREATE TABLE customers87(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dasher_id	trip_id	estimated_delivery_timestamp	actual_delivery_timestamp
101	100463	06/05/2022 09:42:00	06/05/2022 09:38:00
102	100482	06/05/2022 15:10:00	06/05/2022 15:46:00
101	100362	06/07/2022 15:33:00	06/07/2022 16:45:00
102	100657	07/07/2022 15:52:00	-
103	100213	06/12/2022 14:13:00	06/12/2022 14:10:00

```sql
CREATE TABLE customers87(
customer_id INT,
signup_timestamp VARCHAR(50) )

INSERT INTO customers87 VALUES (8472, '05/30/2022 00:00:00'),
(2341, '06/01/2022 00:00:00'),
(1314, '06/03/2022 00:00:00'),
(1435, '06/05/2022 00:00:00'),
(5421, '06/07/2022 00:00:00')
SELECT * FROM customers87
SELECT * FROM trips
SELECT * FROM orders87
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	signup_timestamp
8472	05/30/2022 00:00:00
2341	06/01/2022 00:00:00
1314	06/03/2022 00:00:00
1435	06/05/2022 00:00:00
5421	06/07/2022 00:00:00

```sql
SELECT * FROM orders87

WITH Detail_customer AS (WITH Customer_order AS (SELECT C.customer_id,C. signup_timestamp,O.order_id,O.status,O.order_timestamp,O.trip_id
FROM orders87 O INNER JOIN customers87 C ON O.customer_id = C.customer_id)
SELECT CO.customer_id,CO. signup_timestamp,CO.order_id,CO.status,CO.order_timestamp,T.estimated_delivery_timestamp,
T.actual_delivery_timestamp,CO.trip_id
FROM Customer_order CO INNER JOIN trips T ON CO.trip_id = T.trip_id)
SELECT * FROM Detail_customer

Q91.
CREATE TABLE Salary(
id INT,
employee_id INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	signup_timestamp	order_id	status	order_timestamp	estimated_delivery_timestamp	actual_delivery_timestamp	trip_id
8472	05/30/2022 00:00:00	727424	completed successfully	06/05/2022 09:12:00	06/05/2022 09:42:00	06/05/2022 09:38:00	100463
2341	06/01/2022 00:00:00	242513	completed incorrectly	06/05/2022 14:40:00	06/05/2022 15:10:00	06/05/2022 15:46:00	100482
1314	06/03/2022 00:00:00	141367	completed incorrectly	06/07/2022 15:03:00	06/07/2022 15:33:00	06/07/2022 16:45:00	100362
5421	06/07/2022 00:00:00	582193	never_received	07/07/2022 15:22:00	07/07/2022 15:52:00	-	100657
1314	06/03/2022 00:00:00	253613	completed successfully	06/12/2022 13:43:00	06/12/2022 14:13:00	06/12/2022 14:10:00	100213

Result 21

Q91. Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.

```
1489
1490    Q91.
1491    CREATE TABLE Salary(
1492    id INT,
1493    employee_id INT,
1494    amount INT,
1495    pay_date DATE,
1496    PRIMARY KEY (id) )
1497    INSERT INTO Salary VALUES (1, 1, 9000, '2017/03/31')
1498    INSERT INTO Salary VALUES (2, 2, 6000, '2017/03/31'),
1499    (3, 3, 10000, '2017/03/31'),
1500    (4, 1, 7000, '2017/02/28'),
1501    (5, 2, 6000, '2017/02/28'),
1502    (6, 3, 8000, '2017/02/28')
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	employee_id	amount	pay_date
1	1	9000	2017-03-31
2	2	6000	2017-03-31
3	3	10000	2017-03-31
4	1	7000	2017-02-28
5	2	6000	2017-02-28
6	3	8000	2017-02-28
NULL	NULL	NULL	NULL

```
1505    CREATE TABLE IF NOT EXISTS Employee91(
1506    employee_id INT,
1507    department_id INT,
1508    PRIMARY KEY(employee_id) )
1509    INSERT INTO Employee91 VALUES (1, 1),
1510    (2, 2),
1511    (3, 2)
1512    SELECT *FROM Employee91
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

employee_id	department_id
1	1
2	2
3	2
NULL	NULL

```
1513
1514    SELECT SUM(amount)/3 AS March_avgsal FROM Salary WHERE MONTH(pay_date) = 3
1515    SELECT SUM(amount)/3 AS March_avgsal FROM Salary WHERE MONTH(pay_date) = 2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

March_avgsal
8333.3333

```
1515    SELECT SUM(amount)/3 AS March_avgsal FROM Salary WHERE MONTH(pay_date) = 2
1516
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

March_avgsal
7000.0000

```
1517
1518  (WITH dep_emp AS (WITH March AS (SELECT * FROM Salary WHERE MONTH(pay_date) = 3)
1519  SELECT E.department_id,M.id,M.employee_id,M.amount,M.pay_date FROM March M INNER JOIN Employee91 E ON M.employee_id = E.employee_id)
1520  SELECT SUBSTR(pay_date,1,7) AS pay_month,department_id,IF(amount > 8333.3333, 'Higher','lower') AS Comparision FROM dep_emp
1521  GROUP BY department_id)
1522  UNION ALL
1523  (WITH dep_emp AS (WITH March AS (SELECT * FROM Salary WHERE MONTH(pay_date) = 2)
1524  SELECT E.department_id,M.id,M.employee_id,M.amount,M.pay_date FROM March M INNER JOIN Employee91 E ON M.employee_id = E.employee_id)
1525  SELECT SUBSTR(pay_date,1,7) AS pay_month,department_id,IF(amount > 7000.0000, 'Higher','same') AS Comparision FROM dep_emp
1526  GROUP BY department_id) ORDER BY department_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pay_month	department_id	Comparision
2017-03	1	Higher
2017-02	1	same
2017-03	2	lower
2017-02	2	same

Q92. Write an SQL query to report for each install date, the number of players that installed the game on that day, and the day one retention.

```
1528  Q92.
1529  CREATE TABLE IF NOT EXISTS Activity92(
1530  player_id INT,
1531  device_id INT,
1532  event_date DATE,
1533  games_played INT,
1534  PRIMARY KEY (player_id, event_date) )
1535  INSERT INTO Activity92 VALUES (1, 2, '2016-03-01', 5),
1536  (1, 2, '2016-03-02', 6),
1537  (2, 3, '2017-06-25', 1),
1538  (3, 1, '2016-03-01', 0),
1539  (3, 4, '2016-07-03', 5)
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-01	0
3	4	2016-07-03	5
NULL	NULL	NULL	NULL

```
1543
1544  SELECT event_date,COUNT(player_id) AS installs,player_id / COUNT(player_id) AS installs FROM Activity92 GROUP BY event_date
1545
1546  USE assignment
1547  Q93.
1548  CREATE TABLE IF NOT EXISTS Players(
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

event_date	installs	installs
2016-03-01	2	0.5000
2016-03-02	1	1.0000
2017-06-25	1	2.0000
2016-07-03	1	3.0000

Result 31

Q93. Write an SQL query to find the winner in each group.

```
1547    Q93.
1548    CREATE TABLE IF NOT EXISTS Players(
1549    player_id int,
1550    group_id int,
1551    PRIMARY KEY (player_id) )
1552    INSERT INTO Players VALUES (15, 1),
1553    (25, 1),
1554    (30, 1),
1555    (45, 1),
1556    (10, 2),
1557    (35, 2),
1558    (50, 2),
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

player_id	group_id
10	2
15	1
20	3
25	1
30	1
35	2
40	3
45	1
50	2
NULL	NULL

```
1563    CREATE TABLE IF NOT EXISTS Matches(
1564    match_id int,
1565    first_player int,
1566    second_player int,
1567    first_score int,
1568    second_score int,
1569    PRIMARY KEY (match_id) )
1570    INSERT INTO Matches VALUES (1, 15, 45, 3, 0),
1571    (2, 30, 25, 1, 2),
1572    (3, 30, 15, 2, 0),
1573    (4, 40, 20, 5, 2),
1574    (5, 35, 50, 1, 1)
1575    SELECT * FROM Matches
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

match_id	first_player	second_player	first_score	second_score
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
NULL	NULL	NULL	NULL	NULL

Q94. Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam.

```
1577    Q94.
1578    CREATE TABLE IF NOT EXISTS Student94(
1579    student_id int,
1580    student_name varchar(50),
1581    PRIMARY KEY(student_id) )
1582    INSERT INTO Student94 VALUES (1, 'Daniel'),
1583    (2, 'Jade'),
1584    (3, 'Stella'),
1585    (4, 'Jonathan'),
1586    (5, 'Will')
1587    SELECT * FROM Student94
1588
1589    CREATE TABLE IF NOT EXISTS Exam(
1590    exam id int,
```

student_id	student_name
1	Daniel
2	Jade
3	Stella
4	Jonathan
5	Will
NULL	NULL

```
1588
1589    CREATE TABLE IF NOT EXISTS Exam(
1590    exam_id int,
1591    student_id int,
1592    score int,
1593    PRIMARY KEY (exam_id, student_id) )
1594    INSERT INTO Exam VALUES (10, 1, 70),
1595    (10, 2, 80),
1596    (10, 3, 90),
1597    (20, 1, 80),
1598    (30, 1, 70),
```

exam_id	student_id	score
10	1	70
10	2	80
10	3	90
20	1	80
30	1	70
30	3	80
30	4	90
40	1	60
40	2	70
40	4	80
NULL	NULL	NULL

```
1606
1607    WITH avg_exam AS (SELECT student_id FROM Exam WHERE (exam_id,score) IN (SELECT exam_id,avg(score) FROM Exam  GROUP BY exam_id))
1608    SELECT S.student_id,S.student_name FROM avg_exam A INNER JOIN Student94 S ON A.student_id = S.student_id
1609
1610    Q96.
```

student_id	student_name
2	Jade
1	Daniel
3	Stella
2	Jade

Q97. Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

```
1612
1613    Q97.
1614    CREATE TABLE IF NOT EXISTS emails(
1615    email_id INT,
1616    user_id INT,
1617    signup_date VARCHAR(50) )
1618    INSERT INTO emails VALUES (125, 7771, '06/14/2022 00:00:00'),
1619    (236, 6950, '07/01/2022 00:00:00'),
1620    (433, 1052, '07/09/2022 00:00:00')
1621    ALTER TABLE emails ADD COLUMN signup_date_new DATETIME
1622    SET SQL_SAFE_UPDATES = 0
1623    UPDATE emails SET signup_date_new = STR_TO_DATE(signup_date, '%m/%d/%Y %H:%i:%s')
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_id	user_id	signup_date	signup_date_new
125	7771	06/14/2022 00:00:00	2022-06-14 00:00:00
236	6950	07/01/2022 00:00:00	2022-07-01 00:00:00
433	1052	07/09/2022 00:00:00	2022-07-09 00:00:00

```
1625
1626    CREATE TABLE IF NOT EXISTS texts(
1627    text_id INT,
1628    email_id INT,
1629    signup_action varchar(50) )
1630    INSERT INTO texts VALUES (6878, 125, 'Confirmed'),
1631    (6920, 236, 'Not Confirmed'),
1632    (6994, 236, 'Confirmed')
1633    SELECT * FROM texts
1634    SELECT * FROM emails
1635
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

text_id	email_id	signup_action
6878	125	Confirmed
6920	236	Not Confirmed
6994	236	Confirmed

```
1635
1636    SELECT E.email_id,E.signup_date_new,E.user_id,T.text_id,T.signup_action
1637    FROM emails E LEFT JOIN texts T ON E.email_id = T.email_id GROUP BY email_id,signup_action
1638
1639    Q99.
1640    CREATE TABLE IF NOT EXISTS activities(
1641    activity_id INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

email_id	signup_date_new	user_id	text_id	signup_action
125	2022-06-14 00:00:00	7771	6878	Confirmed
236	2022-07-01 00:00:00	6950	6994	Confirmed
236	2022-07-01 00:00:00	6950	6920	Not Confirmed
433	2022-07-09 00:00:00	1052	NULL	NULL

Q99. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

```
1639    Q99.
1640    CREATE TABLE IF NOT EXISTS activities(
1641    activity_id INT,
1642    user_id INT,
1643    activity_type ENUM ('send', 'open', 'chat'),
1644    time_spent float,
1645    activity_date VARCHAR(50) )
1646    ALTER TABLE activities MODIFY time_spent DECIMAL(30,2)
1647    SELECT * FROM activities
1648    INSERT INTO activities VALUES (7274, 123, 'open', 4.50, '06/22/2022 12:00:00')
1649    INSERT INTO activities VALUES (2425, 123, 'send', 3.50, '06/22/2022 12:00:00'),
1650    (1413, 456, 'send', 5.67, '06/23/2022 12:00:00'),
1651    (1414, 789, 'chat', 11.00, '06/25/2022 12:00:00'),
1652    (2536, 456, 'open', 3.00, '06/25/2022 12:00:00')
1653    ALTER TABLE activities ADD COLUMN activity_date_new DATETIME
1654    UPDATE activities SET activity_date_new = STR_TO_DATE(activity_date, '%m/%d/%Y %H:%i:%s')
```

activity_id	user_id	activity_type	time_spent	activity_date	activity_date_new
7274	123	open	4.50	06/22/2022 12:00:00	2022-06-22 12:00:00
2425	123	send	3.50	06/22/2022 12:00:00	2022-06-22 12:00:00
1413	456	send	5.67	06/23/2022 12:00:00	2022-06-23 12:00:00
1414	789	chat	11.00	06/25/2022 12:00:00	2022-06-25 12:00:00
2536	456	open	3.00	06/25/2022 12:00:00	2022-06-25 12:00:00

```
1655
1656    CREATE TABLE IF NOT EXISTS age_breakdown(
1657    user_id INT,
1658    age_bucket ENUM ('21-25', '26-30', '31-25') )
1659    ALTER TABLE age_breakdown MODIFY age_bucket VARCHAR(50)
1660    INSERT INTO age_breakdown VALUES (123, '31-35'),
1661    (456, '26-30'),
1662    (789, '21-25')
1663    SELECT * FROM age_breakdown
```

user_id	age_bucket
123	31-35
456	26-30
789	21-25

```
1664
1665    WITH CTE AS (SELECT AB.age_bucket,A.activity_type
1666    FROM activities A INNER JOIN age_breakdown AB ON A.user_id = AB.user_id)
1667    select *,
1668    case
1669    when age_bucket = '31-35' AND activity_type = 'open' then ROUND((4.50/(3.50+4.50))*100.0, 2)
1670    when age_bucket = '31-35' AND activity_type = 'send' then ROUND((3.50/(3.50+4.50))*100.0, 2)
1671    when age_bucket = '26-30' AND activity_type = 'open' then ROUND((3.00/(5.67+3.00))*100.0, 2)
1672    when age_bucket = '26-30' AND activity_type = 'send' then ROUND((5.67/(5.67+3.00))*100.0, 2)
1673    end as Perc
1674    from CTE
1675
```

age_bucket	activity_type	Perc
31-35	open	56.25
31-35	send	43.75
26-30	send	65.40
21-25	chat	NULL
26-30	open	34.60

Q100 . Write a query to return the IDs of these LinkedIn power creators in ascending order.

```
1675
1676    Q100.
1677    CREATE TABLE IF NOT EXISTS Personal_profiles(
1678    profile_id INT,
1679    `name` VARCHAR(50),
1680    followers INT )
1681    ALTER TABLE Personal_profiles MODIFY followers BIGINT
1682    ALTER TABLE Personal_profiles DROP followers
1683    ALTER TABLE Personal_profiles ADD COLUMN followers BIGINT
1684    TRUNCATE Personal_profiles
1685    INSERT INTO Personal_profiles VALUES (1, 'Nick Singh', 92000),
1686    (2, 'Zach Wilson', 199000),
1687    (3, 'Daliana Liu', 171000),
1688    (4, 'Ravit Jain', 107000),
1689    (5, 'Vin Vashishta', 139000),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

profile_id	name	followers
1	Nick Singh	92000
2	Zach Wilson	199000
3	Daliana Liu	171000
4	Ravit Jain	107000
5	Vin Vashishta	139000
6	Susan Wojcicki	39000

```
1693
1694    CREATE TABLE  IF NOT EXISTS employee_company(
1695    personal_profile_id INT,
1696    company_id INT )
1697    INSERT INTO employee_company VALUES (1, 4),
1698    (1, 9),
1699    (2, 2),
1700    (3, 1),
1701    (4, 3),
1702    (5, 6),
1703    (6, 5)
1704    SELECT * FROM employee_company
1705
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

personal_profile_id	company_id
1	4
1	9
2	2
3	1
4	3
5	6
6	5

```
1705
1706    CREATE TABLE IF NOT EXISTS company_pages(
1707    company_id INT,
1708    `name` VARCHAR(50),
1709    followers VARCHAR(50) )
1710    ALTER TABLE company_pages MODIFY followers BIGINT
1711    TRUNCATE company_pages
1712    INSERT INTO company_pages VALUES (1, 'The Data Science Podcast', 8000),
1713    (2, 'Airbnb', 700000),
1714    (3, 'The Ravit Show', 6000),
1715    (4, 'DataLemur', 200),
1716    (5, 'YouTube', 16000000),
1717    (6, 'DataScience.Vin', 4500),
1718    (9, 'Ace The Data Science Interview', 4479)
1719    SELECT * FROM company_pages
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_id	name	followers
1	The Data Science Podcast	8000
2	Airbnb	700000
3	The Ravit Show	6000
4	DataLemur	200
5	YouTube	16000000
6	DataScience.Vin	4500
9	Ace The Data Science Interview	4479

```sql
WITH CTE AS (SELECT EC.personal_profile_id,CP.name AS Company_name,CP.followers AS Company_followers FROM company_pages CP
INNER JOIN employee_company EC ON CP.company_id = EC.company_id)
SELECT C.personal_profile_id,PP.name,PP.followers AS Linkedin_followers,C.Company_name,C.Company_followers FROM
Personal_profiles PP INNER JOIN CTE C
ON PP.profile_id = C.personal_profile_id WHERE PP.followers  BETWEEN 90000 AND 200000 ORDER BY  personal_profile_id ASC LIMIT 1,5
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

personal_profile_id	name	Linkedin_followers	Company_name	Company_followers
1	Nick Singh	92000	Ace The Data Science Interview	4479
2	Zach Wilson	199000	Airbnb	700000
3	Daliana Liu	171000	The Data Science Podcast	8000
4	Ravit Jain	107000	The Ravit Show	6000
5	Vin Vashishta	139000	DataScience.Vin	4500

Q 101. Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Limit to 1000 rows

```sql
Q101.
CREATE TABLE IF NOT EXISTS UserActivity(
username VARCHAR(50),
activity VARCHAR(50),
startDate DATE,
endDate DATE )
SELECT * FROM UserActivity
INSERT INTO UserActivity VALUES ('Alice', 'Travel', '2020-02-12', '2020-02-20'),
('Alice', 'Dancing', '2020-02-21', '2020-02-23'),
('Alice', 'Travel', '2020-02-24', '2020-02-28'),
('Bob', 'Travel', '2020-02-11', '2020-02-18')
SELECT * FROM UserActivity
SELECT username,activity,MAX(startDate),endDate FROM UserActivity WHERE username ='Alice'  AND 'Bob'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

username	activity	startDate	endDate
Alice	Travel	2020-02-12	2020-02-20
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

Q103. Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

```sql
Q103.
CREATE TABLE IF NOT EXISTS STUDENTS103(
ID INT,
`Name` VARCHAR(50),
Marks INT )
INSERT INTO STUDENTS103 VALUES (1,'Ashley',81),
(2,'Samantha',75),
(4,'Julia',76),
(3,'Belvet',84)
SELECT * FROM STUDENTS103
SELECT * FROM STUDENTS103 WHERE Marks >75
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Name	Marks
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

```
1754    SELECT * FROM STUDENTS103 WHERE Marks >75
1755
1756    0104.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Name	Marks
1	Ashley	81
4	Julia	76
3	Belvet	84

Q104. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee_id.

```
1756    Q104.
1757    SELECT * FROM employee79
1758    SELECT name FROM employee79 WHERE salary > 2000 AND months < 10
1759
1760    Q105.
1761
1762    0106.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

```
1757    SELECT * FROM employee79
1758    SELECT name FROM employee79 WHERE salary > 2000 AND months < 10
1759
1760    Q105.
1761
1762    0106.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
Angela
Michael
Todd
Joe

Q106.

```
1761
1762    Q106.
1763    CREATE TABLE Employees106(
1764    ID INT,
1765    `Name` VARCHAR(50),
1766    Salary INT )
1767    INSERT INTO Employees106 VALUES (1,'Kristeen',1420),
1768    (2,'Ashley',2006),
1769    (3,'Julia',2210),
1770    (4,'Maria',3000)
1771    SELECT * FROM Employees106
1772    SELECT AVG(REPLACE(salary, 0, '')) AS ErrorSalary_Avg,ROUND(AVG(salary)) AS ActualSalary_Avg,
1773    ROUND(AVG(salary) - AVG(REPLACE(salary, 0, '')), 2) AS error_calculations
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Name	Salary
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

```
1772    SELECT AVG(REPLACE(salary, 0, '')) AS ErrorSalary_Avg,ROUND(AVG(salary)) AS ActualSalary_Avg,
1773    ROUND(AVG(salary) - AVG(REPLACE(salary, 0, '')), 2) AS error_calculations
1774    FROM Employees106
1775
1776    Q107.
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ErrorSalary_Avg	ActualSalary_Avg	error_calculations
98	2159	2061

Q107. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

```
1775
1776    Q107.
1777    SELECT * FROM employee79
1778
1779    SELECT *, salary * months AS Maximum_Earnings FROM employee79
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

```
1777    SELECT * FROM employee79
1778    SELECT *, salary * months AS Maximum_Earnings FROM employee79
1779
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	name	months	salary	Maximum_Earnings
12228	Rose	15	1968	29520
33645	Angela	1	3443	3443
45692	Frank	17	1608	27336
56118	Patrick	7	1345	9415
59725	Lisa	11	2330	25630
74197	Kimberly	16	4372	69952
78454	Bonnie	8	1771	14168
83565	Michael	6	2017	12102
98607	Todd	5	3396	16980
99989	Joe	9	3573	32157

Q108.

```
1779
1780    Q108.
1781    CREATE TABLE IF NOT EXISTS OCCUPATIONS(
1782    `Name` VARCHAR(50),
1783    Occupation VARCHAR(50) )
1784    INSERT INTO OCCUPATIONS VALUES ('Samantha','Doctor'),
1785    ('Julia', 'Actor'),
1786    ('Maria', 'Actor'),
1787    ('Meera', 'Singer'),
1788    ('Ashely', 'Professor'),
1789    ('Ketty', 'Professor'),
1790    ('Christeen', 'Professor'),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

```
1795    SELECT Name,SUBSTRING(Occupation, 1,1 ) AS First_letter  FROM OCCUPATIONS  ORDER BY Name
1796
1797    SELECT Occupation,COUNT(Occupation) AS occupation_count FROM OCCUPATIONS GROUP BY Occupation
1798    USE assignment
1799
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Name	First_letter
Ashely	P
Christeen	P
Jane	A
Jenny	D
Julia	A
Ketty	P
Maria	A
Meera	S
Priya	S
Samantha	D

```
796
797     SELECT Occupation,COUNT(Occupation) AS occupation_count FROM OCCUPATIONS GROUP BY Occupation
798     USE assignment
799
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Occupation	occupation_count
Doctor	2
Actor	3
Singer	2
Professor	3

Q144 . Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students get the same salary offer.

```
Q144.
CREATE TABLE IF NOT EXISTS Students144(
ID INT,
`Name` VARCHAR(50) )
INSERT INTO Students144 VALUES (1,'Ashley'),
(2,'Samantha'),
(3,'julia'),
(4,'Scarlet')
SELECT * FROM Students144

CREATE TABLE IF NOT EXISTS Friends(
ID INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Name
1	Ashley
2	Samantha
3	julia
4	Scarlet

```
CREATE TABLE IF NOT EXISTS Friends(
ID INT,
Friend_ID INT )
INSERT INTO Friends VALUES (1,2),
(2,3),
(3,4),
(4,1)
SELECT * FROM Friends

CREATE TABLE IF NOT EXISTS Packages (
ID INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Friend_ID
1	2
2	3
3	4
4	1

```
CREATE TABLE IF NOT EXISTS Packages (
ID INT,
Salary FLOAT)
INSERT INTO Packages VALUES (1, 15.20),
(2, 10.06),
(3, 11.55),
(4, 12.12)
SELECT * FROM Packages
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Salary
1	15.2
2	10.06
3	11.55
4	12.12

```
WITH CTE AS (SELECT F.ID,S.Name,P.Salary,F.Friend_ID FROM Friends F INNER JOIN Students144 S ON F.ID = S.ID
INNER JOIN Packages P ON F.ID = P.ID)
SELECT C.ID,C.Name,C.Salary,C.Friend_ID,S.Name AS Best_Friend,P.Salary AS BestiSalary FROM CTE C LEFT JOIN Students144 S
ON C.Friend_ID = S.ID
LEFT JOIN Packages P ON C.Friend_ID = P.ID WHERE C.Salary < P.Salary
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	Name	Salary	Friend_ID	Best_Friend	BestiSalary
4	Scarlet	12.12	1	Ashley	15.2
2	Samantha	10.06	3	julia	11.55
3	julia	11.55	4	Scarlet	12.12

Q145.

```
1845
1846    Q145.
1847    CREATE TABLE IF NOT EXISTS Hackers(
1848    hacker_id INT,
1849    `name` VARCHAR(50) )
1850    INSERT INTO Hackers VALUES (5580, 'Rose'),
1851    (8439, 'Angela'),
1852    (27205, 'Frank'),
1853    (52243, 'Patrick'),
1854    (52348, 'Lisa'),
1855    (57645, 'Kimberly'),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

hacker_id	name
5580	Rose
8439	Angela
27205	Frank
52243	Patrick
52348	Lisa
57645	Kimberly
77726	Bonnie
83082	Michael
86870	Todd
90411	Joe

```
1862
1863    CREATE TABLE IF NOT EXISTS Difficulty(
1864    difficulty_level INT,
1865    score INT )
1866    INSERT INTO Difficulty VALUES (1, 20),
1867    (2, 30),
1868    (3, 40),
1869    (4, 60),
1870    (5, 80),
1871    (6, 100),
1872    (7, 120)
1873    SELECT * FROM Difficulty
1874
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

difficulty_level	score
1	20
2	30
3	40
4	60
5	80
6	100
7	120

```
1875    CREATE TABLE IF NOT EXISTS Challenges(
1876    challenge_id INT,
1877    hacker_id INT,
1878    difficulty_level INT )
1879    INSERT INTO Challenges VALUES (4810, 77726, 4),
1880    (21089, 27205, 1),
1881    (36566, 5580, 7),
1882    (66730, 52243, 6),
1883    (71055, 52243, 2)
1884    SELECT * FROM Challenges
1885
1886    CREATE TABLE IF NOT EXISTS Submissions (
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

challenge_id	hacker_id	difficulty_level
4810	77726	4
21089	27205	1
36566	5580	7
66730	52243	6
71055	52243	2

```sql
CREATE TABLE IF NOT EXISTS Submissions (
submission_id INT,
hacker_id INT,
challenge_id INT,
score INT )
INSERT INTO Submissions VALUES (68628, 77726, 36566, 30),
(65300, 77726, 21089, 10),
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35

```sql
SELECT S.submission_id,H.name,S.hacker_id,S.challenge_id,S.score,D.difficulty_level FROM Submissions S
INNER JOIN Hackers H ON S.hacker_id = H.hacker_id
INNER JOIN Difficulty D ON S.Score = D.Score WHERE S.score = 100
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

submission_id	name	hacker_id	challenge_id	score	difficulty_level
97397	Joe	90411	66730	100	6

Q146.

```sql
Q146.
CREATE TABLE IF NOT EXISTS Projects(
Task_ID INT,
Start_Date DATE,
End_Date DATE )
INSERT INTO Projects VALUES (1, '2015-10-01', '2015-10-02'),
(2, '2015-10-02', '2015-10-03'),
(3, '2015-10-03', '2015-10-04'),
(4, '2015-10-13', '2015-10-14'),
(5, '2015-10-14', '2015-10-15'),
(6, '2015-10-28', '2015-10-29'),
(7, '2015-10-30', '2015-10-31')
SELECT * FROM Projects
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Task_ID	Start_Date	End_Date
1	2015-10-01	2015-10-02
2	2015-10-02	2015-10-03
3	2015-10-03	2015-10-04
4	2015-10-13	2015-10-14
5	2015-10-14	2015-10-15
6	2015-10-28	2015-10-29
7	2015-10-30	2015-10-31

Q148 .

```
1942    Q148.
1943    CREATE TABLE IF NOT EXISTS Payments(
1944    payer_id INT,
1945    recipient_id INT,
1946    amount INT )
1947    INSERT INTO Payments VALUES (101, 201, 30),
1948    (201, 101, 10),
1949    (101, 301, 20),
1950    (301, 101, 80),
1951    (201, 301, 70)
1952    SELECT * FROM Payments
1953    SELECT payer_id, COUNT(recipient_id) AS unique_relationships FROM Payments GROUP BY payer_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

payer_id	recipient_id	amount
101	201	30
201	101	10
101	301	20
301	101	80
201	301	70

```
1953    SELECT payer_id, COUNT(recipient_id) AS unique_relationships FROM Payments GROUP BY payer_id
1954
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

payer_id	unique_relationships
101	2
201	2
301	1

Q149.

```
1955    Q149.
1956    CREATE TABLE IF NOT EXISTS user_transactions149 (
1957    transaction_id INT,
1958    user_id INT,
1959    spend DECIMAL(5,2),
1960    transaction_date VARCHAR(50) )
1961    ALTER TABLE user_transactions149 ADD COLUMN transaction_date_new DATETIME
1962    INSERT INTO user_transactions149 VALUES (759274, 111, 49.50, '02/03/2022 00:00:00'),
1963    (850371, 111, 51.00, '03/15/2022 00:00:00'),
1964    (615348, 145, 36.30, '03/22/2022 00:00:00'),
1965    (137424, 156, 151.00, '04/04/2022 00:00:00'),
1966    (248475, 156, 87.00, '04/16/2022 00:00:00')
1967    SET SQL_SAFE_UPDATES = 0
1968    UPDATE user_transactions149 SET transaction_date_new = STR_TO_DATE(transaction_date, '%m/%d/%Y %H:%i:%s')
1969    SELECT * FROM user_transactions149
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	user_id	spend	transaction_date	transaction_date_new
759274	111	49.50	02/03/2022 00:00:00	2022-02-03 00:00:00
850371	111	51.00	03/15/2022 00:00:00	2022-03-15 00:00:00
615348	145	36.30	03/22/2022 00:00:00	2022-03-22 00:00:00
137424	156	151.00	04/04/2022 00:00:00	2022-04-04 00:00:00
248475	156	87.00	04/16/2022 00:00:00	2022-04-16 00:00:00

```
1970
1971    SELECT *,MIN(transaction_date_new) FROM user_transactions149 WHERE spend >=50 GROUP BY user_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	user_id	spend	transaction_date	transaction_date_new	MIN(transaction_date_new)
850371	111	51.00	03/15/2022 00:00:00	2022-03-15 00:00:00	2022-03-15 00:00:00
137424	156	151.00	04/04/2022 00:00:00	2022-04-04 00:00:00	2022-04-04 00:00:00

Q150.

```
1974    Q150.
1975    CREATE TABLE IF NOT EXISTS measurements(
1976    measurement_id INT,
1977    measurement_value DECIMAL(10,2),
1978    measurement_time VARCHAR(50) )
1979    INSERT INTO measurements VALUES (131233, 1109.51, '07/10/2022 09:00:00'),
1980    (135211, 1662.74, '07/10/2022 11:00:00'),
1981    (523542, 1246.24, '07/10/2022 13:15:00'),
1982    (143562, 1124.50, '07/11/2022 15:00:00'),
1983    (346462, 1234.14, '07/11/2022 16:45:00')
1984    SELECT * FROM measurements
1985    ALTER TABLE measurements ADD COLUMN measurement_time_new DATETIME
1986    UPDATE measurements SET measurement_time_new = STR_TO_DATE(measurement_time, '%m/%d/%Y %H:%i:%s')
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

measurement_id	measurement_value	measurement_time	measurement_time_new
131233	1109.51	07/10/2022 09:00:00	2022-07-10 09:00:00
135211	1662.74	07/10/2022 11:00:00	2022-07-10 11:00:00
523542	1246.24	07/10/2022 13:15:00	2022-07-10 13:15:00
143562	1124.50	07/11/2022 15:00:00	2022-07-11 15:00:00
346462	1234.14	07/11/2022 16:45:00	2022-07-11 16:45:00

```
1988
1989    (WITH CTE AS (SELECT measurement_id,measurement_value,measurement_time_new,DAY(measurement_time_new) AS DAY,
1990    row_number() OVER(ORDER BY measurement_time_new) AS ROW_NUMBERS FROM measurements WHERE DAY(measurement_time_new) = 10)
1991    SELECT measurement_time_new,measurement_value as even_sum FROM CTE WHERE MOD(ROW_NUMBERS ,2) = 0)
1992    UNION ALL
1993    (WITH CTE AS (SELECT measurement_id,measurement_value,measurement_time_new,DAY(measurement_time_new) AS DAY,
1994    row_number() OVER(ORDER BY measurement_time_new) AS ROW_NUMBERS FROM measurements WHERE DAY(measurement_time_new) = 11)
1995    SELECT measurement_time_new,measurement_value as even_sum FROM CTE WHERE MOD(ROW_NUMBERS ,2) = 0)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

measurement_time_new	even_sum
2022-07-10 11:00:00	1662.74
2022-07-11 16:45:00	1234.14

```
1996
1997    (WITH CTE AS (SELECT measurement_id,measurement_value,measurement_time_new,DAY(measurement_time_new) AS DAY,
1998    row_number() OVER(ORDER BY measurement_time_new) AS ROW_NUMBERS FROM measurements WHERE DAY(measurement_time_new) = 10)
1999    SELECT measurement_time_new,SUM(measurement_value) AS odd_sum FROM CTE WHERE MOD(ROW_NUMBERS ,2) != 0)
2000    UNION ALL
2001    (WITH CTE AS (SELECT measurement_id,measurement_value,measurement_time_new,DAY(measurement_time_new) AS DAY,
2002    row_number() OVER(ORDER BY measurement_time_new) AS ROW_NUMBERS FROM measurements WHERE DAY(measurement_time_new) = 11)
2003    SELECT measurement_time_new,measurement_value as odd_sum FROM CTE WHERE MOD(ROW_NUMBERS ,2) != 0)
2004
2005    Q153.
2006    CREATE TABLE IF NOT EXISTS ad_campaigns(
2007    campaign_id INT,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

measurement_time_new	odd_sum
2022-07-10 09:00:00	2355.75
2022-07-11 15:00:00	1124.50

Q153.

```
2005    Q153.
2006    CREATE TABLE IF NOT EXISTS ad_campaigns(
2007    campaign_id INT,
2008    spend INT,
2009    revenue FLOAT,
2010    advertiser_id INT )
2011    INSERT INTO ad_campaigns VALUES (1, 5000, 7500, 3),
2012    (2, 1000, 900, 1),
2013    (3, 3000, 12000, 2),
2014    (4, 500, 2000, 4),
2015    (5, 100, 400, 4)
2016    SELECT * FROM ad_campaigns
2017
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

campaign_id	spend	revenue	advertiser_id
1	5000	7500	3
2	1000	900	1
3	3000	12000	2
4	500	2000	4
5	100	400	4

```
2017
2018    SELECT advertiser_id, ROUND(revenue/spend, 2) AS ROAS FROM ad_campaigns GROUP BY advertiser_id ORDER BY advertiser_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

advertiser_id	ROAS
1	0.9
2	4
3	1.5
4	4

Q154.

```
2020    Q154.
2021    CREATE TABLE IF NOT EXISTS employee_pay (
2022    employee_id INT,
2023    salary INT,
2024    title VARCHAR(50) )
2025    INSERT INTO  employee_pay VALUES  (101, 80000, 'Data Analyst'),
2026    (102, 90000, 'Data Analyst'),
2027    (103, 100000, 'Data Analyst'),
2028    (104, 30000, 'Data Analyst')
2029    INSERT INTO employee_pay VALUES (105, 120000, 'Data Scientist'),
2030    (106, 100000, 'Data Scientist'),
2031    (107, 80000, 'Data Scientist'),
2032    (108, 310000, 'Data Scientist')
2033    SELECT * FROM employee_pay
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	salary	title
101	80000	Data Analyst
102	90000	Data Analyst
103	100000	Data Analyst
104	30000	Data Analyst
105	120000	Data Scientist
106	100000	Data Scientist
107	80000	Data Scientist
108	310000	Data Scientist

```
2034
2035    (SELECT * FROM employee_pay WHERE salary > (SELECT AVG(salary) AS AVG FROM employee_pay WHERE title = 'Data Scientist'))
2036    UNION
2037    (SELECT * FROM employee_pay WHERE salary < (SELECT AVG(salary) AS AVG FROM employee_pay WHERE title = 'Data Analyst'))
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_id	salary	title
108	310000	Data Scientist
104	30000	Data Analyst

Q156.

```
2039    Q156.
2040    CREATE TABLE IF NOT EXISTS purchases(
2041    user_id INT,
2042    product_id INT,
2043    quantity INT,
2044    purchase_date VARCHAR(50) )
2045    INSERT INTO purchases VALUES (536, 3223, 6, '01/11/2022 12:33:44'),
2046    (827, 3585, 35, '02/20/2022 14:05:26'),
2047    (536, 3223, 5, '03/02/2022 09:33:28'),
2048    (536, 1435, 10, '03/02/2022 08:40:00'),
2049    (827, 2452, 45, '04/09/2022 00:00:00')
2050    ALTER TABLE purchases ADD COLUMN purchase_date_new DATETIME
2051    SET SQL_SAFE_UPDATES = 0
2052    UPDATE purchases SET purchase_date_new = STR_TO_DATE(purchase_date, '%m/%d/%Y %H:%i:%s')
2053    SELECT * FROM purchases
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	product_id	quantity	purchase_date	purchase_date_new
536	3223	6	01/11/2022 12:33:44	2022-01-11 12:33:44
827	3585	35	02/20/2022 14:05:26	2022-02-20 14:05:26
536	3223	5	03/02/2022 09:33:28	2022-03-02 09:33:28
536	1435	10	03/02/2022 08:40:00	2022-03-02 08:40:00
827	2452	45	04/09/2022 00:00:00	2022-04-09 00:00:00

```
2054
2055    SELECT user_id,COUNT(DISTINCT(purchase_date_new)) AS repeat_purchasers,product_id FROM purchases GROUP BY user_id,product_id
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id	repeat_purchasers	product_id
536	1	1435
536	2	3223
827	1	2452
827	1	3585

Q157.

```
2058    Q157.
2059    CREATE TABLE IF NOT EXISTS transactions157(
2060    transaction_id INT,
2061    `type` VARCHAR(50),
2062    amount DECIMAL(5,2),
2063    transaction_date VARCHAR(50) )
2064    INSERT INTO transactions157 VALUES (19153, 'deposit', 65.90, '07/10/2022 10:00:00'),
2065    (53151, 'deposit', 178.55, '07/08/2022 10:00:00'),
2066    (29776, 'withdrawal', 25.90, '07/08/2022 10:00:00'),
2067    (16461, 'withdrawal', 45.99, '07/08/2022 10:00:00'),
2068    (77134, 'deposit', 32.60, '07/10/2022 10:00:00')
2069    ALTER TABLE transactions157 ADD COLUMN transaction_date_new DATETIME
2070    UPDATE transactions157 SET transaction_date_new = STR_TO_DATE(transaction_date, '%m/%d/%Y %H:%i:%s')
2071    SELECT * FROM transactions157
2072
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_id	type	amount	transaction_date	transaction_date_new
19153	deposit	65.90	07/10/2022 10:00:00	2022-07-10 10:00:00
53151	deposit	178.55	07/08/2022 10:00:00	2022-07-08 10:00:00
29776	withdrawal	25.90	07/08/2022 10:00:00	2022-07-08 10:00:00
16461	withdrawal	45.99	07/08/2022 10:00:00	2022-07-08 10:00:00
77134	deposit	32.60	07/10/2022 10:00:00	2022-07-10 10:00:00

```
2073    WITH CTE AS (SELECT *,DAY(transaction_date_new) AS DAY,
2074    CASE
2075    WHEN DAY(transaction_date_new) = 8 AND  type = 'deposit' THEN amount
2076    ELSE 'NULL' END AS '8_DAY_DEPOSIT',
2077    CASE
2078    WHEN DAY(transaction_date_new) = 8 AND  type = 'withdrawal' THEN amount
2079    ELSE 'NULL' END AS '8_DAY_WITHDRAWAL',
2080    CASE
2081    WHEN DAY(transaction_date_new) = 10 AND  type = 'deposit' THEN amount
2082    ELSE 'NULL' END AS '10_DAY_DEPOSIT',
2083    CASE
2084    WHEN DAY(transaction_date_new) = 10 AND  type = 'withdrawal' THEN amount
2085    ELSE 'NULL' END AS '10_DAY_WITHDRAWAL'
2086    FROM transactions157 )
2087    SELECT transaction_date_new, ROUND((8_DAY_DEPOSIT - SUM(8_DAY_WITHDRAWAL)) , 2) AS balance, SUM(10_DAY_DEPOSIT)
2088    FROM CTE
2089    GROUP BY transaction_date_new
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

transaction_date_new	balance	SUM(10_DAY_DEPOSIT)
2022-07-10 10:00:00	0	98.5
2022-07-08 10:00:00	106.66	0

Q158.

```
Q158.
CREATE TABLE IF NOT EXISTS product_spend(
category VARCHAR(50),
product VARCHAR(50),
user_id INT,
spend DECIMAL(3,2),
transaction_date VARCHAR(50) )
ALTER TABLE product_spend MODIFY spend DECIMAL(10,2)
INSERT INTO product_spend VALUES ('appliance', 'refrigerator', 165, 246.00, '12/26/2021 12:00:00'),
('appliance', 'refrigerator', 123, 299.99, '03/02/2022 12:00:00'),
('appliance', 'washing machine', 123, 219.80, '03/02/2022 12:00:00'),
('electronics', 'vacuum',  178, 152.00, '04/05/2022 12:00:00'),
('electronics', 'wireless headset', 156, 249.90, '07/08/2022 12:00:00'),
('electronics', 'vacuum', 145, 189.00, '07/15/2022 12:00:00')
SELECT * FROM product_spend
ALTER TABLE product_spend ADD COLUMN transaction_date_new DATETIME
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category	product	user_id	spend	transaction_date	transaction_date_new
appliance	refrigerator	165	246.00	12/26/2021 12:00:00	2021-12-26 12:00:00
appliance	refrigerator	123	299.99	03/02/2022 12:00:00	2022-03-02 12:00:00
appliance	washing machine	123	219.80	03/02/2022 12:00:00	2022-03-02 12:00:00
electronics	vacuum	178	152.00	04/05/2022 12:00:00	2022-04-05 12:00:00
electronics	wireless headset	156	249.90	07/08/2022 12:00:00	2022-07-08 12:00:00
electronics	vacuum	145	189.00	07/15/2022 12:00:00	2022-07-15 12:00:00

```
ALTER TABLE product_spend ADD COLUMN transaction_date_new DATETIME
SET SQL_SAFE_UPDATES = 0
UPDATE product_spend SET transaction_date_new = STR_TO_DATE(transaction_date, '%m/%d/%Y %H:%i:%s')

SELECT category,product,SUM(spend) AS total_spend,YEAR(transaction_date_new) AS year FROM product_spend
WHERE YEAR(transaction_date_new) = 2022 GROUP BY product
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category	product	total_spend	year
appliance	refrigerator	299.99	2022
appliance	washing machine	219.80	2022
electronics	vacuum	341.00	2022
electronics	wireless headset	249.90	2022