

City-Dataset: <https://docs.google.com/spreadsheets/d/1dk9kRwcMxi5USuJqxtITD05S-aOUD6fzNzVW41dcpqc/edit?usp=sharing>

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:

### CITY

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

Solution:

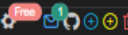
```
select * from city where countrycode = 'USA' and population > 100000;
```

```
orfig > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1:3306 > create-db-template.sql > ...
```

```
1 > Execute
2 create table city
3 (
4   id int,
5   name varchar(17),
6   countrycode varchar(3),
7   district varchar(20),
8   population int
9 );
10 > Execute
11 insert into city values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','NLD','Noord-Holland',135621), (214,'Porto Alegre','BRA','Rio Grande do Sul',1314032), (397,'Lauro de
12 547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDI','Bujumbura',300000), (554,'Santiago de Chile','CHL','Santiago',4783954), (626,'al-Minya','EGY','al-Minya',281360), (646,'San
13 762,'Bahir Dar','ETH','Amhara',96140), (796,'Baguio','PHL','CAR',252386), (896,'Malungon','PHL','Southern Mindanao',93232), (984,'Banjul','GMB','Banjul',42326), (924,'Villa Nueva','GT
14 998,'Waru','IDN','East Java',124300), (1155,'Latur','IND','Maharashtra',197408), (1222,'Tenali','IND','Andhra Pradesh',143726), (1235,'Tirunelveli','IND','Tamil Nadu',135825), (1256,'
15 1279,'Neyveli','IND','Tamil Nadu',118080), (1293,'Pallavaram','IND','Tamil Nadu',111866), (1350,'Dehli','IND','Bihar',94526), (1383,'Tabriz','IRN','East Azerbaizhan',1191043), (1385,'I
16 1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Romagna',89852), (1613,'Neyagawa','JPN','Osaka',257315), (1630,'Ageo','JPN','Saitama',209442), (1661,'S
17 1681,'Onuta','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107078), (1793,'Novi Sad','YUG','Vojvodina',179626), (1857,'Kelowna','CAN','British Columbia',89442), (1895,'H
18 1900,'Changchun','CHN','Jilin',281200), (1913,'Lanzhou','CHN','Gansu',1565000), (1947,'Changzhou','CHN','Jiangsu',530000), (2070,'Dezhou','CHN','Shandong',195485), (2081,'Heze','CHN'
19 2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',120000), (2193,'Lianyuan','CHN','Hunan',118858), (2227,'Xingcheng','CHN','Liaoning',102384), (2273,'Villavicencio','
20 2386,'Yongju','KOR','Kyongsangbuk',131097), (2387,'Chinhae','KOR','Kyongsangnam',125997), (2388,'Sangju','KOR','Kyongsangbuk',124116), (2406,'Heraklion','GRC','Crete',116178), (2440,
21 2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX','Veracruz',390058), (2602,'Ocosingo','MEX','Chiapas',171495), (2609,'M
22 2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600), (2706,'Tete','MOZ','Tete',101984), (2716,'Sittwe (Akyab)','MMR','Rakhine',137600), (2922,'Caroli
23 2972,'Malabo','GNQ','Bioko',40000), (3073,'Essen','DEU','Nordrhein-Westfalen',599515), (3169,'Apia','WSM','Upolu',35900), (3198,'Dakar','SEN','Cap-Vert',785071), (3253,'Hama','SYR','H
24 3353,'Sousse','TUN','Sousse',145900), (3377,'Kahranammaras','TUR','Kahranammaras',245772), (3430,'Odesa','UKR','Odesa',1011000), (3581,'St Petersburg','RUS','Pietari',4694000), (3770,
25 3878,'Scottsdale','USA','Arizona',202705), (3965,'Corona','USA','California',124966), (3973,'Concord','USA','California',121780), (3977,'Cedar Rapids','USA','Iowa',120758), (3982,'Cor
26 4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',90555);
27 > Execute
28 select * from city where countrycode = 'USA' and population > 100000;
```

city X

```
select * from city where countrycode = 'USA' and population > 100000
```

Q Input to filter result  Cost: 5ms < 1 > Total 6

	id int	name varchar	countrycode varchar	district varchar	population int
1	3815	El Paso	USA	Texas	563662
2	3878	Scottsdale	USA	Arizona	202705
3	3965	Corona	USA	California	124966
4	3973	Concord	USA	California	121780
5	3977	Cedar Rapids	USA	Iowa	120758
6	3982	Coral Springs	USA	Florida	117549

**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:

### CITY

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

Solution:

```
select name from city where countrycode = 'USA' and population > 120000;
```

The screenshot shows a database IDE with the following content:

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 @ 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
> Execute
3 create table city
4 ( id int,
5  name varchar(17),
6  countrycode varchar(3),
7  district varchar(20),
8  population int
9 );
> Execute
10 insert into city values (6,'Rotterdam','NLD', 'Zuid-Holland',593321), (19,'Zaanstad','NLD', 'Noord-Holland',135621), (214,'Porto Alegre','BRA', 'Rio Grande do Si
11 (547,'Dobric','BGR', 'Varna',180399), (552,'Bujumbura','BDI', 'Bujumbura',380000), (554,'Santiago de Chile','CHL', 'Santiago',4783954), (626,'al-Minya','EGY', 'a
12 (762,'Bahir Dar','ETH', 'Amhara',96140), (796,'Baguio','PHL', 'CAR',252386), (896,'Malungon','PHL', 'Southern Mindanao',93232), (904,'Banjul','GNB', 'Banjul',423
13 (990,'Waru','IDN', 'East Java',124300), (1155,'Latur','IND', 'Maharashtra',197488), (1222,'Tenali','IND', 'Andhra Pradesh',143726), (1235,'Tirunelveli','IND', 'T
14 (1279,'Neyyveli','IND', 'Tamil Nadu',118000), (1293,'Pallavaram','IND', 'Tamil Nadu',111866), (1350,'Dehri','IND', 'Bihar',94526), (1383,'Tabriz','IRN', 'East Aze
15 (1508,'Bolzano','ITA', 'Trentino-Alto Adige',97232), (1520,'Cesena','ITA', 'Emilia-Romagna',89852), (1613,'Neyagawa','JPN', 'Osaka',257315), (1630,'Ageo','JPN',
16 (1681,'Omuta','JPN', 'Fukuoka',142889), (1739,'Tokuyama','JPN', 'Yamaguchi',107078), (1793,'Novi Sad','YUG', 'Vojvodina',179626), (1857,'Kelowna','CAN', 'British
17 (1900,'Changchun','CHN', 'Jilin',2812000), (1913,'Lanzhou','CHN', 'Gansu',1565800), (1947,'Changzhou','CHN', 'Jiangsu',530000), (2070,'Dezhou','CHN', 'Shandong',
18 (2153,'Xianning','CHN', 'Hubei',136811), (2192,'Lhasa','CHN', 'Tibet',120000), (2193,'Lianyuan','CHN', 'Hunan',118858), (2227,'Xingcheng','CHN', 'Liaoning',10238
19 (2386,'Yongju','KOR', 'Kyongsangbuk',131097), (2387,'Chinhae','KOR', 'Kyongsangnam',125997), (2388,'Sangju','KOR', 'Kyongsangbuk',124116), (2406,'Herakleion','G
20 (2462,'Lilongwe','MKI', 'Lilongwe',435964), (2505,'Taza','MAR', 'Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX', 'Veracruz',390058), (2602,'Ocosingo','MEX',
21 (2670,'San Pedro Cholula','MEX', 'Puebla',99734), (2689,'Palikir','FSM', 'Pohnpei',8600), (2706,'Tete','MOZ', 'Tete',101984), (2716,'Sittwe (Akyab)','MMR', 'Rakh
22 (2972,'Malabo','GNQ', 'Bioko',40000), (3073,'Essen','DEU', 'Nordrhein-Westfalen',599515), (3169,'Apia','WSM', 'Upolu',35900), (3198,'Dakar','SEN', 'Cap-Vert',785
23 (3353,'Sousse','TUN', 'Sousse',145900), (3377,'Kahramanmaraş','TUR', 'Kahramanmaraş',245772), (3430,'Odesa','UKR', 'Odesa',1011000), (3581,'St Petersburg','RUS',
24 (3878,'Scottsdale','USA', 'Arizona',202705), (3965,'Corona','USA', 'California',124966), (3973,'Concord','USA', 'California',121780), (3977,'Cedar Rapids','USA',
25 (4058,'Boulder','USA', 'Colorado',91238), (4061,'Fall River','USA', 'Massachusetts',90555);
> Execute
26 select name from city where countrycode = 'USA' and population > 120000;
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

The query results are displayed in a table:

name	varchar
1	El Paso
2	Scottsdale
3	Corona
4	Concord
5	Cedar Rapids

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

**CITY**

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

Solution:

```
select * from city;
```

select \* from city
Cost: 5ms < 1 2 3 > Total: 83

🔍
🔄
🗑️

	id int	name varchar	countrycode varchar	district varchar
1	6	Rotterdam	NLD	Zuid-Holland
2	19	Zaanstad	NLD	Noord-Holland
3	214	Porto Alegre	BRA	Rio Grande do Sul
4	397	Lauro de Freitas	BRA	Bahia
5	547	Dobric	BGR	Varna
6	552	Bujumbura	BDI	Bujumbura
7	554	Santiago de Chile	CHL	Santiago
8	626	al-Minya	EGY	al-Minya
9	646	Santa Ana	SLV	Santa Ana
10	762	Bahir Dar	ETH	Amhara
11	796	Baguio	PHL	CAR
12	896	Malungon	PHL	Southern Mindanao
13	904	Banjul	GMB	Banjul
14	924	Villa Nueva	GTM	Guatemala
15	990	Waru	IDN	East Java
16	1155	Latur	IND	Maharashtra
17	1222	Tenali	IND	Andhra Pradesh
18	1235	Tirunelveli	IND	Tamil Nadu
19	1256	Alandur	IND	Tamil Nadu
20	1279	Neyveli	IND	Tamil Nadu
21	1293	Pallavaram	IND	Tamil Nadu
22	1350	Dehri	IND	Bihar
23	1383	Tabriz	IRN	East Azerbaijan
24	1385	Karaj	IRN	Teheran
25	1508	Bolzano	ITA	Trentino-Alto Adige
26	1520	Cesena	ITA	Emilia-Romagna
27	1613	Neyagawa	JPN	Osaka
28	1630	Ageo	JPN	Saitama
29	1661	Sayama	JPN	Saitama
30	1681	Omuta	JPN	Fukuoka

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1:3306 > create database city;
5  name varchar(17),
6  countrycode varchar(3),
7  district varchar(20),
8  population int
9 );
> Execute
10 insert into city Values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','NLD',
11 (547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDI','Bujumbura',300000), (554
12 (762,'Bahir Dar','ETH','Amhara',96140), (796,'Baguio','PHL','CAR',252386), (896,'Malu
13 (990,'Waru','IDN','East Java',124300), (1155,'Latur','IND','Maharashtra',197408), (12
14 (1279,'Neyveli','IND','Tamil Nadu',118000), (1293,'Pallavaram','IND','Tamil Nadu',111
15 (1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Roma
16 (1681,'Omuta','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107078), (1
17 (1900,'Changchun','CHN','Jilin',2812000), (1913,'Lanzhou','CHN','Gansu',1565800), (19
18 (2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',120000), (2193,'L
19 (2386,'Yongju','KOR','Kyongsangbuk',131097), (2387,'Chinhae','KOR','Kyongsangnam',125
20 (2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',
21 (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600
22 (2972,'Malabo','GNO','Bioko',40000), (3073,'Essen','DEU','Nordrhein-Westfalen',599515
23 (3353,'Sousse','TUN','Sousse',145900), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',24
24 (3878,'Scottsdale','USA','Arizona',202705), (3965,'Corona','USA','California',124966)
25 (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',985
> Execute
26 select * from city
27 limit 30;
28
29
30

```

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-d
5 name varchar(17),
6 countrycode varchar(3),
7 district varchar(20),
8 population int
9 );
10 > Execute
11 insert into city Values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','N
12 (547,'Dobric','BGR','Varna',188399), (552,'Bujumbura','BDI','Bujumbura',388888), (554
13 (762,'Bahir Dar','ETH','Amhara',86146), (796,'Baguio','PHL','CAR',252386), (886,'Malu
14 (998,'Waru','IDN','East Java',124388), (1155,'Latur','IND','Kaharashtra',197488), (12
15 (1279,'Meyveli','IND','Tamil Nadu',118888), (1293,'Pallavaram','IND','Tamil Nadu',111
16 (1588,'Bolzano','ITA','Trentino-Alto Adige',97232), (1528,'Casena','ITA','Emilia-Roma
17 (1681,'Omuta','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107878), (1
18 (1980,'Changchun','CHN','Jilin',281288), (1913,'Lanzhou','CHN','Gansu',156588), (19
19 (2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',128888), (2193,'L
20 (2386,'Yongju','KOR','Kyongsangbuk',131897), (2387,'Chinhae','KOR','Kyongsangnam',125
21 (2462,'Lilongwe','MTI','Lilongwe',435964), (2585,'Taza','MAR','Taza-Al Hoceima-Taou',
22 (2678,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8688
23 (2972,'Malabo','GNQ','Bioko',48888), (3073,'Essen','DEU','Nordrhein-Westfalen',599515
24 (3353,'Sousse','TUN','Sousse',145988), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',24
25 (3878,'Scottsdale','USA','Arizona',282785), (3965,'Corona','USA','California',124966)
26 (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',985
27 > Execute
28 select * from city
29 limit 30;
```

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

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

Solution:

```
select * from city where id = 1661;
```

The screenshot shows a SQL client interface with a query editor and a results pane. The query editor contains the following SQL code:

```
5 name varchar(17),
6 countrycode varchar(3),
7 district varchar(20),
8 population int
9 );
10 insert into city Values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','NLD','Noord-Holland',135621), (214,'
11 (547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDI','Bujumbura',300000), (554,'Santiago de Chile','CHL','Santiago
12 (762,'Bahir Dar','ETH','Ankara',96140), (796,'Baguio','PHL','CAR',252386), (896,'Malungon','PHL','Southern Mindanao',932
13 (990,'Waru','IDN','East Java',124300), (1155,'Latur','IND','Maharashtra',197408), (1222,'Tenali','IND','Andhra Pradesh',1
14 (1279,'Neyveli','IND','Tamil Nadu',118080), (1293,'Pallavaram','IND','Tamil Nadu',111866), (1350,'Dehri','IND','Bihar',9
15 (1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Romagna',89852), (1613,'Neyagawa','JPN',
16 (1681,'Omura','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107078), (1793,'Novi Sad','YUG','Vojvodina',17
17 (1900,'Changchun','CHN','Jilin',2812000), (1913,'Lanzhou','CHN','Gansu',1565000), (1947,'Changzhou','CHN','Jiangsu',5300
18 (2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',120000), (2193,'Lianyuan','CHN','Hunan',118858), (22
19 (2386,'Yongju','KOR','Kyongsangbuk',131097), (2387,'Chinhae','KOR','Kyongsangnam',125997), (2388,'Sangju','KOR','Kyongsar
20 (2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX','Veracr
21 (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600), (2706,'Tete','MOZ','Tete',101984
22 (2972,'Malabo','GNQ','Bioko',40000), (3073,'Essen','DEU','Nordrhein-Westfalen',599515), (3169,'Apia','WSM','Upolu',35900
23 (3353,'Sousse','TUN','Sousse',145900), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',245772), (3430,'Odesa','UKR','Odesa',1
24 (3878,'Scottsdale','USA','Arizona',202705), (3965,'Corona','USA','California',124966), (3973,'Concord','USA','California
25 (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',90555);
26 select * from city where id = 1661;
27
28
29
30
```

The results pane shows the following table:

id	name	countrycode	district	population	
1	1661	Sayama	JPN	Saitama	162472

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

The CITY table is described as follows:



## CITY

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

Solution:

```
select * from city where countrycode = 'JPN';
```

create-db-template.sql X

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

```

3  create table city
4  ( id int,
5   name varchar(17),
6   countrycode varchar(3),
7   district varchar(20),
8   population int
9  );
10
11  > Execute
12  Insert into city Values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','NLD','Noord-Holland',135621), (214,'Porto Alegre','BRA','R1
13  (547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDI','Bujumbura',300000), (554,'Santiago de Chile','CHL','Santiago',4703954), (626,'al-Mi
14  (762,'Bahir Dar','ETH','Amhara',96140), (796,'Baguio','PHL','CAR',252386), (896,'Malungon','PHL','Southern Mindanao',93232), (904,'Banjul','GMB'
15  (990,'Waru','IDN','East Java',124300), (1155,'Latur','IND','Maharashtra',197408), (1222,'Tenali','IND','Andhra Pradesh',143726), (1235,'Tiruneliv
16  (1279,'Neyvelli','IND','Tamil Nadu',118000), (1293,'Pallavaram','IND','Tamil Nadu',111866), (1350,'Dehri','IND','Bihar',94526), (1383,'Tabriz','I
17  (1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Romagna',89852), (1613,'Neyagawa','JPN','Osaka',257315), (1630,
18  (1681,'Omuta','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107078), (1793,'Novi Sad','YUG','Vojvodina',179626), (1857,'Kelowna','
19  (1900,'Changchun','CHN','Jilin',2812000), (1913,'Lanzhou','CHN','Gansu',1565800), (1947,'Changzhou','CHN','Jiangsu',530000), (2070,'Dezhou','CHN
20  (2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',120000), (2193,'Lianyuan','CHN','Hunan',118858), (2227,'Xingcheng','CHN','Li
21  (2386,'Yongju','KOR','Kyongsangbuk',131097), (2387,'Chinhae','KOR','Kyongsangnam',125997), (2388,'Sangju','KOR','Kyongsangbuk',124116), (2406,'H
22  (2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX','Veracruz',390058), (2602,'Oco
23  (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600), (2706,'Tete','MOZ','Tete',101984), (2716,'Sittwe (Akyab)
24  (2972,'Malabo','GNQ','Bioko',40000), (3073,'Essen','DEU','Nordrhein-Westfalen',599515), (3169,'Apia','WSM','Upolu',35900), (3198,'Dakar','SEN','
25  (3353,'Sousse','TUN','Sousse',145900), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',245772), (3430,'Odesa','UKR','Odesa',1011000), (3581,'St Pete
26  (3878,'Scottsdale','USA','Arizona',202705), (3965,'Corona','USA','California',124966), (3973,'Concord','USA','California',121780), (3977,'Cedar
27  (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',90555);
28
29  > Execute
30  select * from city where countrycode = 'JPN';
31
32
33

```

city X

select \* from city where countrycode = 'JPN'

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

	id	name	countrycode	district	population
	int	varchar	varchar	varchar	int
1	1613	Neyagawa	JPN	Osaka	257315
2	1630	Ageo	JPN	Saitama	209442
3	1661	Sayama	JPN	Saitama	162472
4	1681	Omuta	JPN	Fukuoka	142889
5	1739	Tokuyama	JPN	Yamaguchi	107078

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

The CITY table is described as follows:

**CITY**

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

Solution:

```
select name from city where countrycode = 'JPN';
```

```
create-db-template.sql X
mysql> data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
5  name varchar(17),
6  countrycode varchar(3),
7  district varchar(20),
8  population int
9  );
10 > Execute
11 insert into city values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','NLD','Noord-Holland',135621), (214,'Pek
12 (547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDI','Bujumbura',300000), (554,'Santiago de Chile','CHL','Santiago',
13 (762,'Bahir Dar','ETH','Amhara',96140), (796,'Baguio','PHL','CAR',252386), (896,'Malungon','PHL','Southern Mindanao',93232),
14 (998,'Waru','IDN','East Java',124300), (1155,'Latun','IND','Maharashtra',197408), (1222,'Tenali','IND','Andhra Pradesh',145
15 (1279,'Neyveli','IND','Tamil Nadu',118000), (1293,'Pallavaram','IND','Tamil Nadu',111866), (1350,'Dehri','IND','Bihar',945
16 (1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Romagna',89852), (1613,'Neyagawa','JPN','C
17 (1681,'Omuta','JPN','Fukuoka',142889), (1719,'Tokuyama','JPN','Yamaguchi',107078), (1793,'Novi Sad','YUG','Vojvodina',1796
18 (1900,'Changchun','CHN','Jilin',2812000), (1913,'Lanzhou','CHN','Gansu',1565800), (1947,'Changzhou','CHN','Jiangsu',530000),
19 (2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',120000), (2193,'Lianyuan','CHN','Hunan',118858), (2227,
20 (2386,'Yongju','KOR','Kyongsangbuk',131097), (2387,'Chinhae','KOR','Kyongsangnam',125997), (2388,'Sangju','KOR','Kyongsangn
21 (2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX','Veracruz
22 (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600), (2706,'Tete','MOZ','Tete',101984),
23 (2972,'Malabo','GNQ','Bioko',40000), (3073,'Essen','DEU','Nordrhein-Westfalen',599515), (3169,'Apia','WSM','Upolu',35900),
24 (3353,'Sousse','TUN','Sousse',145900), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',245772), (3430,'Odessa','UKR','Odessa',101
25 (3878,'Scottsdale','USA','Arizona',202705), (3965,'Corona','USA','California',124966), (3973,'Concord','USA','California',1
26 (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',90555);
27 > Execute
28 select name from city where countrycode = 'JPN';
29
city X
select name from city where countrycode = 'JPN'
Q Input to filter result
name varchar
1 Neyagawa
2 Ageo
3 Sayama
4 Omuta
5 Tokuyama
```

station-table: <https://docs.google.com/spreadsheets/d/1sHPhE7waIQD5mL7ppFNqybyoOJY3E51N0cWYzhp2UH4/edit?usp=sharing>

Q7. Query a list of CITY and STATE from the STATION table.  
The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select city, state from station;
```

station	
select city, state from station	
Input to filter result	
Cost: 7ms	
Total 499	
city	state
varchar	varchar
1 Kisse Mills	MO
2 Loma Mar	CA
3 Sandy Hook	CT
4 Tipton	IN
5 Arlington	CO
6 Turner	AR
7 Slidell	LA
8 Negreet	LA
9 Glencoe	KY
10 Chelsea	IA
11 Chignik Lagoon	AK
12 Pelahatchie	MS
13 Hanna City	IL
14 Dorrance	KS
15 Albany	CA
16 Monument	KS
17 Manchester	MD
18 Prescott	IA
19 Graettinger	IA
20 Cahone	CO
21 Sturgis	MS
22 Upperco	MD
23 Highwood	IL
24 Waipahu	HI
25 Bowdon	GA
26 Tyler	MN
27 Watkins	CO
28 Republic	MI
29 Millville	CA
30 Aguanga	CA
31 Bowdon Junction	GA
32 Moresni	AZ
33 South El Monte	CA

442 (82, 'Many', 'LA', 36, 94),
443 (644, 'Seward', 'AK', 120, 35),
444 (391, 'Berryton', 'KS', 60, 139),
445 (696, 'Chilhowee', 'MO', 79, 49),
446 (905, 'Newark', 'IL', 72, 129),
447 (81, 'Cowgill', 'MO', 136, 27),
448 (31, 'Nowinger', 'MO', 108, 111),
449 (299, 'Goodman', 'MS', 101, 117),
450 (84, 'Cobalt', 'CT', 87, 26),
451 (754, 'South Haven', 'MI', 144, 52),
452 (144, 'Eskeledge', 'KS', 107, 63),
453 (305, 'Bennington', 'KS', 93, 83),
454 (226, 'Decatur', 'MS', 71, 117),
455 (224, 'West Hyannisport', 'MA', 58, 96),
456 (924, 'Ozone', 'FL', 144, 129),
457 (623, 'Jackson', 'AL', 111, 67),
458 (543, 'Lapeer', 'MI', 128, 114),
459 (819, 'Peaks Island', 'ME', 59, 110),
460 (241, 'Razlehurst', 'MS', 49, 108),
461 (457, 'Chester', 'CA', 69, 123),
462 (871, 'Clarkston', 'MI', 93, 80),
463 (470, 'Healdsburg', 'CA', 111, 54),
464 (785, 'Hotchkiss', 'CO', 69, 71),
465 (690, 'Ravenden Springs', 'AR', 67, 108),
466 (62, 'Monroe', 'AR', 131, 150),
467 (365, 'Payson', 'IL', 81, 92),
468 (68, 'Kell', 'IL', 70, 58),
469 (830, 'Strasburg', 'CO', 89, 47),
470 (286, 'Five Points', 'AL', 45, 122),
471 (968, 'Morris City', 'IL', 53, 76),
472 (928, 'Coaling', 'AL', 144, 52),
473 (746, 'Orange City', 'IA', 93, 162),
474 (892, 'Effingham', 'KS', 132, 97),
475 (193, 'Corcoran', 'CA', 81, 139),
476 (225, 'Garden City', 'IA', 24, 119),
477 (573, 'Alton', 'MO', 79, 112),
478 (830, 'Greenway', 'AR', 119, 35),
479 (241, 'Woodsboro', 'MD', 76, 141),
480 (783, 'Strawn', 'IL', 29, 51),
481 (675, 'Dent', 'MI', 70, 136),
482 (270, 'Shingletown', 'CA', 61, 102),
483 (378, 'Clilo', 'IA', 46, 115),
484 (104, 'Valaha', 'FL', 120, 119),
485 (406, 'Leakesville', 'MS', 107, 72),
486 (804, 'Fort Lupton', 'CO', 38, 53),
487 (53, 'Shasta', 'CA', 99, 155),
488 (448, 'Canton', 'MN', 123, 151),
489 (751, 'Agency', 'MO', 59, 95),
490 (29, 'South Carrollton', 'KY', 57, 116),
491 (718, 'Taft', 'CA', 107, 146),
492 (213, 'Calpine', 'CA', 46, 43),
493 (903, 'Knobel', 'AR', 95, 62),
494 (908, 'Bullhead City', 'AZ', 94, 30),
495 (845, 'Tina', 'MO', 131, 28),
496 (685, 'Anthony', 'KS', 45, 161),
497 (257, 'ID', 107, 31),
498 (311, 'South Haven', 'MN', 30, 87),
499 (866, 'Haverhill', 'IA', 61, 109),
500 (598, 'Middleboro', 'MA', 108, 149),
501 (541, 'Siloam', 'GA', 105, 32),
502 (889, 'Lena', 'LA', 78, 129),
503 (654, 'Lee', 'IL', 27, 51),
504 (841, 'Freeport', 'MI', 113, 50),
505 (905, 'Red Florida', 'FL', 110, 50),
506 (249, 'Acme', 'LA', 73, 67),
507 (376, 'Gorham', 'KS', 111, 64),
508 (136, 'Bass Harbor', 'ME', 137, 61),
509 (455, 'Granger', 'IA', 33, 102),
510 select city, state from station;
511



**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.

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude

Solution:

```
select distinct city from station where id%2 = 0;
```

The screenshot shows a SQL IDE with two panes. The left pane displays the query `select distinct city from station where id%2 = 0` and its results, which are 34 distinct city names. The right pane shows a list of cities with their corresponding state and latitude/longitude coordinates.

city	state	lat	long
Kissae Mills	KS	68	119
Loma Mar	MO	79	49
Tipton	IL	72	129
Glencoe	MO	136	27
Chignik Lagoon	MO	108	111
Albany	MS	101	117
Manchester	CT	87	26
Cahone	MI	144	52
Bowdon	KS	107	63
Watkins	KS	93	83
Millville	MS	71	117
Aguanga	MA	58	96
Morenci	FL	144	128
Mccomb	AL	111	67
Gustine	MI	128	114
Delano	ME	59	110
Roy	MS	49	108
Pattonsburg	CA	69	123
Centertown	ME	93	80
Norvell	CA	111	54
Raymondville	CO	69	71
West Hills	AR	67	108
Wickliffe	AR	131	150
Forest Lakes	IL	81	92
Little Rock	IL	70	58
Hampden	CO	89	47
Pine City	AL	45	122
Prince Frederick	IL	53	76
Yazoo City	AL	144	52
Jolon	IA	93	162
Childs	KS	132	97
Shreveport	CA	81	139
Forest	IA	54	119
Sizerock	MO	79	112

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

The STATION table is described as follows:

### STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

For example, if there are three records in the table with CITY values 'New York', 'New York', 'Bengaluru', there are 2 different city names: 'New York' and 'Bengaluru'. The query returns , because total number of records - number of unique city names = 3-2 =1

Solution:

```
(455,'Granger','IA',33,102);select (count(city) - count(distinct city)) as  
'CityCount-DistCityCount' from station;
```

The screenshot shows a MySQL client interface with a list of city records and a query result. The records are as follows:

City	State	Lat	Long
Granger	IA	33	102
Jackson	AL	111	67
Lapeer	MI	128	114
Peaks Island	ME	59	110
Hazlehurst	MS	49	108
Chester	CA	69	123
Clarkston	MI	93	80
Healdsburg	CA	111	54
Hotchkiss	CO	69	71
Ravenden Springs	AR	67	108
Monroe	AR	131	150
Payson	IL	81	92
Kell	IL	70	58
Strasburg	CO	89	47
Five Points	AL	45	122
Norris City	IL	53	76
Coaling	AL	144	52
Orange City	IA	93	162
Effingham	KS	132	97
Corcoran	CA	81	139
Garden City	IA	54	119
Alton	MO	79	112
Greenway	AR	119	35
Woodsboro	MD	76	141
Strawn	IL	29	51
Dent	MN	70	136
Shingletown	CA	61	102
Clio	IA	46	115
Yalaha	FL	120	119
Leakesville	MS	107	72
Fort Lupton	CO	38	93
Shasta	CA	99	155
Canton	MN	123	151
Agency	MO	59	95
South Carrollton	KY	57	116
Taft	CA	107	146
Calpine	CA	46	43
Knobel	AR	95	62
Bullhead City	AZ	94	30
Tina	MO	131	28
Anthony	KS	45	161
Emmett	ID	57	31
South Haven	MN	30	87
Haverhill	IA	61	109
Middleboro	MA	108	149
Siloam	GA	105	92
Lena	LA	78	129
Lee	IL	27	51
Freeport	MI	113	50
Mid Florida	FL	110	50
Acme	LA	73	67
Gorham	KS	111	64
Bass Harbor	ME	137	61

The query result shows the following:

CityCount-DistCityCount
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.

The STATION table is described as follows:

## STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Sample Input

For example, CITY has four entries: DEF, ABC, PQRS and WXY.

Sample Output:

ABC 3

PQRS 4

### Hint -

When ordered alphabetically, the CITY names are listed as ABC, DEF, PQRS, and WXY, with lengths and. The longest name is PQRS, but there are options for shortest named city. Choose ABC, because it comes first alphabetically.

Note

You can write two separate queries to get the desired output. It need not be a single query.

Solution:

```
(select city, length(city) as length from station order by length(city) asc,city asc limit 1)
union
(select city, length(city) as length from station order by length(city) desc,city asc limit 1);
```





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

Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select distinct city from station where left(city,1) in ('a','e','i','o','u');
```



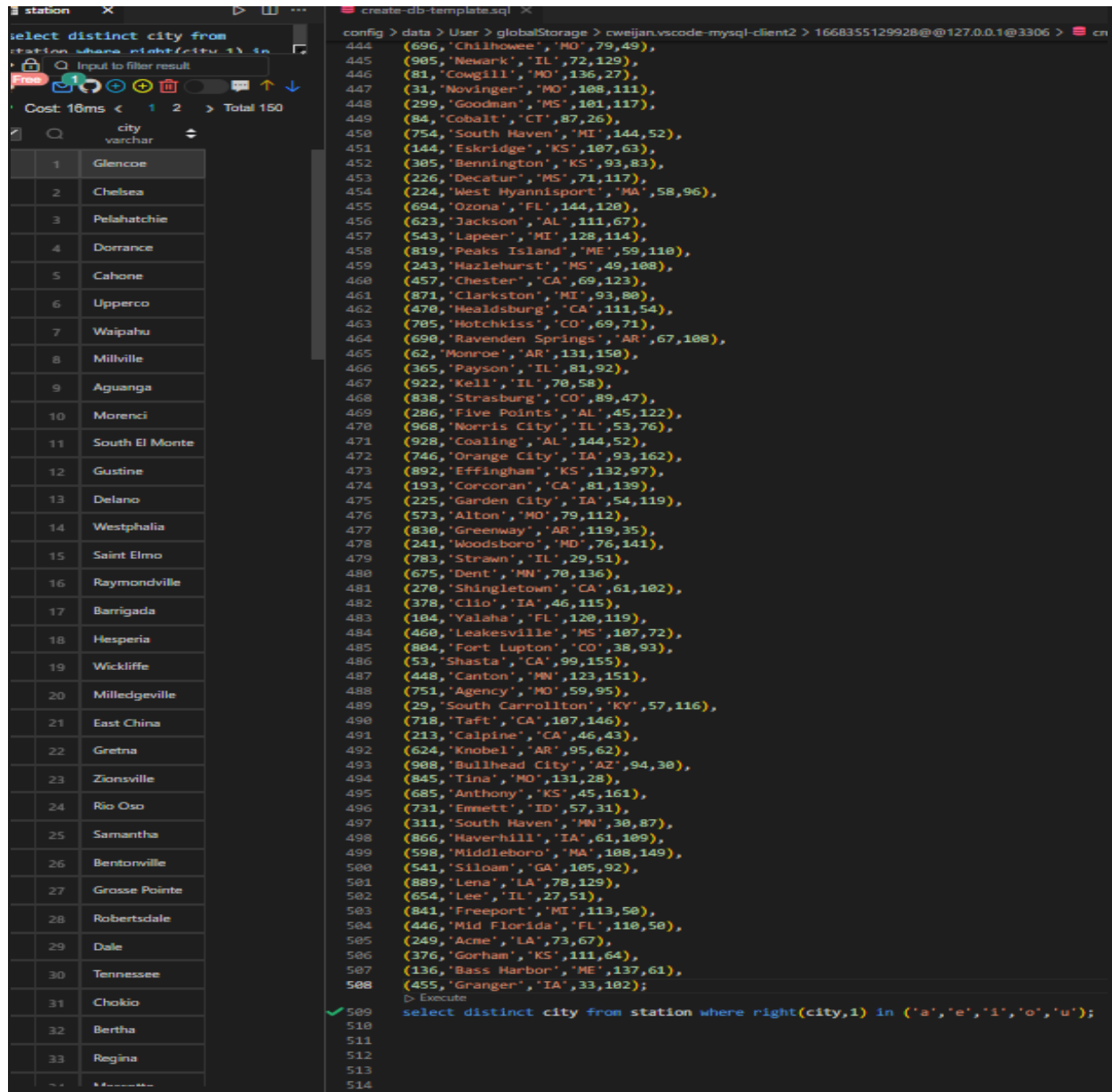
## STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select distinct city from station where right(city,1) in ('a','e','i','o','u');
```



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

Input Format

The STATION table is described as follows:

### STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select distinct city from station where left(city,1) not in ('a','e','i','o','u');
```

The screenshot shows a database client interface with two panes. The left pane displays the query result for the query `select distinct city from station where left(city,1) not in ('a','e','i','o','u');`. The right pane shows a list of cities from the `station` table, ordered by their first letter.

city
Glencoe
Loma Mar
Sandy Hook
Tipton
Turner
Slidell
Negreet
Glencoe
Chelsea
Chignik Lagoon
Pelahatchie
Hanna City
Dorrance
Monument
Manchester
Prescott
Graettinger
Cahone
Sturgis
Highwood
Waipahu
Bowdon
Tyler
Watkins
Republic
Milville
Bowdon Junction
Morenci
South El Monte
Hoskinson
Talbert
Mccomb
Kirk

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

Input Format

The STATION table is described as follows:



## STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select distinct city from station where right(city,1) not in ('a','e','i','o','u');
```

The screenshot displays a database IDE with two panes. The left pane shows the query editor with the following SQL query:

```
select distinct city from station where right(city,1) not in ('a','e','i','o','u');
```

The right pane shows the results of the query, which are 336 distinct city names. The results are displayed in a table with two columns: ID and city. The cities listed are:

ID	city
1	Glencoe
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Arlington
6	Turner
7	Slidell
8	Negreet
9	Chignik Lagoon
10	Hanna City
11	Albany
12	Monument
13	Manchester
14	Prescott
15	Graettinger
16	Sturgis
17	Highwood
18	Bowdon
19	Tyler
20	Watkins
21	Republic
22	Bowdon Junction
23	Hoskinson
24	Talbert
25	Mccomb
26	Kirk
27	Carlock
28	Seward
29	Roy
30	Pattonsburg
31	Centertown
32	Norvell
33	Beaver Island

The right pane shows the SQL script used to create the database and the query results. The script includes the following lines:

```
create database station;
use station;
create table station (
  ID number(4,0) primary key,
  CITY varchar2(21) not null,
  STATE varchar2(2) not null,
  LAT_N number(9,6) not null,
  LONG_W number(9,6) not null
);
insert into station values (1,'Glencoe','IL',72,129),
(2,'Loma Mar','MO',136,27),
(3,'Sandy Hook','MO',108,111),
(4,'Tipton','MS',181,117),
(5,'Arlington','CT',87,26),
(6,'Turner','MI',144,52),
(7,'Slidell','KS',107,63),
(8,'Negreet','KS',93,83),
(9,'Chignik Lagoon','MS',71,117),
(10,'Hanna City','MA',58,96),
(11,'Albany','FL',144,128),
(12,'Monument','AL',111,67),
(13,'Manchester','MI',128,114),
(14,'Prescott','ME',59,118),
(15,'Graettinger','MS',49,108),
(16,'Sturgis','CA',69,123),
(17,'Highwood','MI',93,80),
(18,'Bowdon','CA',111,64),
(19,'Tyler','CO',69,71),
(20,'Watkins','AR',67,108),
(21,'Republic','AR',131,158),
(22,'Bowdon Junction','IL',81,92),
(23,'Hoskinson','IL',78,58),
(24,'Talbert','CO',89,47),
(25,'Mccomb','FL',45,122),
(26,'Kirk','CA',81,119),
(27,'Carlock','IA',54,119),
(28,'Seward','MO',79,112),
(29,'Roy','AR',119,35),
(30,'Pattonsburg','MD',76,141),
(31,'Centertown','IL',29,51),
(32,'Norvell','MN',70,136),
(33,'Beaver Island','CA',61,102),
(34,'Glencoe','IA',46,115),
(35,'Loma Mar','FL',128,115),
(36,'Sandy Hook','MS',107,72),
(37,'Tipton','CO',38,93),
(38,'Arlington','CA',99,155),
(39,'Turner','MN',123,151),
(40,'Slidell','MO',59,93),
(41,'Negreet','KY',57,116),
(42,'Chignik Lagoon','CA',107,146),
(43,'Hanna City','CA',46,43),
(44,'Albany','AR',95,62),
(45,'Monument','AZ',94,38),
(46,'Manchester','MO',131,28),
(47,'Prescott','KS',45,161),
(48,'Graettinger','ID',57,31),
(49,'Sturgis','MN',38,87),
(50,'Highwood','IA',61,108),
(51,'Bowdon','MA',108,149),
(52,'Tyler','GA',105,92),
(53,'Watkins','LA',78,129),
(54,'Republic','IL',27,51),
(55,'Bowdon Junction','MI',113,50),
(56,'Hoskinson','FL',110,50),
(57,'Talbert','LA',73,67),
(58,'Mccomb','KS',111,64),
(59,'Kirk','ME',137,61),
(60,'Carlock','IA',33,102),
(61,'Seward','LA',73,67),
(62,'Roy','KS',111,64),
(63,'Pattonsburg','ME',137,61),
(64,'Centertown','IA',33,102);
```

The query results are displayed in the bottom pane, showing the distinct city names from the station table where the last character of the city name is not a vowel.

**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.

Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select distinct city from station where left(city,1) not in ('a','e','i','o','u') or right(city,1) not in ('a','e','i','o','u');
```

station

select distinct city from station where left(city,1) not in ('a','e','i','o','u') or right(city,1) not in ('a','e','i','o','u');

Cost: 8ms

1 2 3 4 5 > Total 458

	city
	varchar
1	Glencoe
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Arlington
6	Turner
7	Slidell
8	Negreet
9	Glencoe
10	Chelsea
11	Chignik Lagoon
12	Pelahatchie
13	Hanna City
14	Dorrance
15	Albany
16	Monument
17	Manchester
18	Prescott
19	Graettinger
20	Cahone
21	Sturgis
22	Highwood
23	Waipahu
24	Bowdon
25	Tyler
26	Watkins
27	Republic
28	Millville
29	Bowdon Junction
30	Morenci
31	South El Monte
32	Hoskinston
33	Talbert

create-db-templates.sql

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-t

444 (696,'Chilhowee','MO',79,49),

445 (905,'Newark','IL',72,125),

446 (81,'Cowgill','MO',136,27),

447 (31,'Novinger','MO',108,111),

448 (299,'Goodman','MS',101,117),

449 (84,'Cobalt','CT',87,26),

450 (754,'South Haven','MI',144,52),

451 (144,'Eskridge','KS',107,63),

452 (305,'Bennington','KS',93,83),

453 (226,'Decatur','MS',71,117),

454 (224,'West Hyannisport','MA',58,96),

455 (694,'Ozona','FL',144,128),

456 (623,'Jackson','AL',111,67),

457 (543,'Lapeer','MI',128,114),

458 (819,'Peaks Island','ME',59,110),

459 (243,'Hazlehurst','MS',49,108),

460 (457,'Chester','CA',69,123),

461 (871,'Clarkston','MI',93,88),

462 (470,'Healdsburg','CA',111,54),

463 (705,'Hotchkiss','CO',69,71),

464 (690,'Ravenden Springs','AR',67,108),

465 (62,'Monroe','AR',131,150),

466 (365,'Payson','IL',81,92),

467 (922,'Kell','IL',70,58),

468 (838,'Strasburg','CO',89,47),

469 (286,'Five Points','AL',45,122),

470 (968,'Morris City','IL',53,76),

471 (928,'Coaling','AL',144,52),

472 (746,'Orange City','IA',93,162),

473 (892,'Effingham','KS',132,97),

474 (193,'Corcoran','CA',81,139),

475 (225,'Garden City','IA',54,119),

476 (573,'Alton','MO',79,112),

477 (830,'Greenway','AR',119,35),

478 (241,'Woodsboro','MD',76,141),

479 (783,'Strawn','IL',29,51),

480 (675,'Dent','MN',70,136),

481 (270,'Shingletown','CA',61,102),

482 (378,'Clilo','IA',46,115),

483 (104,'Yalaha','FL',120,119),

484 (460,'Leakesville','MS',107,72),

485 (804,'Fort Lupton','CO',38,93),

486 (53,'Shasta','CA',99,155),

487 (448,'Canton','MN',123,151),

488 (751,'Agency','MO',59,95),

489 (29,'South Carrollton','KY',57,116),

490 (718,'Taft','CA',107,146),

491 (213,'Calpine','CA',46,43),

492 (624,'Knobel','AR',95,62),

493 (908,'Bullhead City','AZ',94,30),

494 (845,'Tina','MO',131,28),

495 (685,'Anthony','KS',45,161),

496 (731,'Emmett','ID',57,31),

497 (311,'South Haven','MN',30,87),

498 (866,'Haverhill','IA',61,109),

499 (598,'Middleboro','MA',108,149),

500 (541,'Siloam','GA',105,92),

501 (889,'Lena','LA',78,129),

502 (654,'Lee','IL',27,51),

503 (841,'Freeport','MI',113,50),

504 (446,'Mid Florida','FL',110,50),

505 (249,'Acme','LA',73,67),

506 (376,'Gorham','KS',111,64),

507 (136,'Bass Harbor','ME',137,61),

508 (455,'Granger','IA',33,102);

> Execute

509 select distinct city from station where left(city,1) not in ('a','e','i','o','u')

510 or

511 right(city,1) not in ('a','e','i','o','u');

512

513

514

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

Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution:

```
select distinct city from station where left(city,1) not in
('a','e','i','o','u') and right(city,1) not in ('a','e','i','o','u');
```

station

select distinct city from station where

Free 1

Cost 13ms

< 1 2 3 > Total 299

☒ city varchar

1	Glencoe
2	Loma Mar
3	Sandy Hook
4	Tipton
5	Turner
6	Slidell
7	Negreet
8	Chignik Lagoon
9	Hanna City
10	Monument
11	Manchester
12	Prescott
13	Graettinger
14	Sturgis
15	Highwood
16	Bowdon
17	Tyler
18	Watkins
19	Republic
20	Bowdon Junction
21	Hoskinston
22	Talbert
23	Mccomb
24	Kirk
25	Carlock
26	Seward
27	Roy
28	Pattonsburg
29	Centertown
30	Norvell
31	Beaver Island
32	Jemison
33	West Hills

create-db-template.sql

config > data > User > globalStorage > cweijan.vscode-mysql-client2

```
446 (81,'Cowgill','MO',136,27),
447 (31,'Novinger','MO',108,111),
448 (299,'Goodman','MS',101,117),
449 (84,'Cobalt','CT',87,26),
450 (754,'South Haven','MI',144,52),
451 (144,'Eskridge','KS',107,63),
452 (305,'Bennington','KS',93,83),
453 (226,'Decatur','MS',71,117),
454 (224,'West Hyannisport','MA',58,96),
455 (694,'Ozona','FL',144,120),
456 (623,'Jackson','AL',111,67),
457 (543,'Lapeer','MI',128,114),
458 (819,'Peaks Island','ME',59,110),
459 (243,'Hazlehurst','MS',49,108),
460 (457,'Chester','CA',69,123),
461 (871,'Clarkston','MI',93,80),
462 (470,'Healdsburg','CA',111,54),
463 (705,'Hotchkiss','CO',69,71),
464 (690,'Ravenden Springs','AR',67,108),
465 (62,'Monroe','AR',131,150),
466 (365,'Payson','IL',81,92),
467 (922,'Kell','IL',70,58),
468 (838,'Strasburg','CO',89,47),
469 (286,'Five Points','AL',45,122),
470 (968,'Norris City','IL',53,76),
471 (928,'Coaling','AL',144,52),
472 (746,'Orange City','IA',93,162),
473 (892,'Effingham','KS',132,97),
474 (193,'Corcoran','CA',81,139),
475 (225,'Garden City','IA',54,119),
476 (573,'Alton','MO',79,112),
477 (830,'Greenway','AR',119,35),
478 (241,'Woodboro','MD',76,141),
479 (783,'Strawn','IL',29,51),
480 (675,'Dent','MN',70,136),
481 (270,'Shingletown','CA',61,102),
482 (378,'Clio','IA',46,115),
483 (104,'Yalaha','FL',120,119),
484 (460,'Leakesville','MS',107,72),
485 (804,'Fort Lupton','CO',38,93),
486 (53,'Shasta','CA',99,155),
487 (448,'Canton','MN',123,151),
488 (751,'Agency','MO',59,95),
489 (29,'South Carrollton','KY',57,116),
490 (718,'Taft','CA',107,146),
491 (213,'Calpine','CA',46,43),
492 (624,'Knobel','AR',95,62),
493 (908,'Bullhead City','AZ',94,30),
494 (845,'Tina','MO',131,28),
495 (685,'Anthony','KS',45,161),
496 (731,'Emmett','ID',57,31),
497 (311,'South Haven','MN',30,87),
498 (866,'Haverhill','IA',61,109),
499 (598,'Middleboro','MA',108,149),
500 (541,'Siloam','GA',105,92),
501 (889,'Lena','LA',78,129),
502 (654,'Lee','IL',27,51),
503 (841,'Freeport','MI',113,50),
504 (446,'Mid Florida','FL',110,50),
505 (249,'Acme','LA',73,67),
506 (376,'Gorham','KS',111,64),
507 (136,'Bass Harbor','ME',137,61),
508 (455,'Granger','IA',33,102);|
509 > Execute
510 select distinct city from station where
511 left(city,1) not in ('a','e','i','o','u')
512 and
513 right(city,1) not in ('a','e','i','o','u');
514
515
516
```

**Q17.**

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product\_id is the primary key of this table.

Each row of this table indicates the name and the price of each product. Table:

Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows. product\_id is a foreign key to the Product table.

Each row of this table contains some information about one sale.

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

Return the result table in any order.

The query result format is in the following example.

Input: Product

table:

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

Sales table:

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

Output:

product_id	product_name
1	S8

Explanation:

The product with id 1 was only sold in the spring of 2019.

The product with id 2 was sold in the spring of 2019 but was also sold after the spring of 2019.

The product with id 3 was sold after spring 2019.

We return only product 1 as it is the product that was only sold in the spring of 2019.

Solution:

```
Solution:
(select p.product_id, p.product_name FROM
Product p
INNER JOIN
Sales s
on p.product_id = s.product_id
where s.sale_date >= '2019-01-01' and s.sale_date <= '2019-03-31')
EXCEPT
(select p.product_id, p.product_name FROM
Product p
INNER JOIN
Sales s
on p.product_id = s.product_id
where s.sale_date < '2019-01-01' OR s.sale_date > '2019-03-31')
```

create-db-template.sql

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sc

3 create table Product

4 (product\_id int,

5 product\_name varchar(20),

6 unit\_price int,

7 primary key(product\_id)

8 );

> Execute

9 create table Sales

10 (seller\_id int,

11 product\_id int,

12 buyer\_id int,

13 sale\_date date,

14 quantity int,

15 price int,

16 foreign key(product\_id) references Product(product\_id)

17 );

> Execute

18 insert into Product values

19 (1, 'S8', 1000),

20 (2, 'G4', 800),

21 (3, 'iPhone', 1400);

> Execute

22 insert into Sales values

23 (1, 1, 1, '2019-01-21', 2, 2000),

24 (1, 2, 2, '2019-02-17', 1, 800),

25 (2, 2, 3, '2019-06-02', 1, 800),

26 (3, 3, 4, '2019-05-13', 2, 2800);

> Execute

27 (select p.product\_id, p.product\_name FROM

28 Product p

29 INNER JOIN

30 Sales s

31 on p.product\_id = s.product\_id

32 where s.sale\_date >= '2019-01-01' and s.sale\_date <= '2019-03-31')

33 EXCEPT

34 (select p.product\_id, p.product\_name FROM

35 Product p

36 INNER JOIN

37 Sales s

38 on p.product\_id = s.product\_id

39 where s.sale\_date < '2019-01-01' OR s.sale\_date > '2019-03-31')

40

Users

(select p.product\_id, p.product\_name FROM

Product p

+

🔒

🔍 Input to filter result

⚙️ Free

📧 1

🔄

⊕

⊕

🗑️

🔇

💬

⬆️

⬆️

▶️

Cost: 2ms

<

1

>

Total 1

☑️

🔍

product\_id

int

product\_name

varchar

1

1

S8

**Q18.**

Table: Views

Column Name	Type
article_id	int
author_id	int
viewer_id	int
view_date	date

There is no primary key for this table, it may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

Write an SQL query to find all the authors that viewed at least one of their own articles.

Return the result table sorted by id in ascending order.

The query result format is in the following example.

Input:

Views table:

article_id	author_id	viewer_id	view_date
1	3	5	2019-08-01
1	3	6	2019-08-02
2	7	7	2019-08-01
2	7	6	2019-08-02
4	7	1	2019-07-22
3	4	4	2019-07-21
3	4	4	2019-07-21

Output:

Id
4
7

Solution:

```
select distinct author_id as id from views where author_id = viewer_id order by author_id asc;
```

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
2 use test;
  > Execute
3 create table views(
4     article_id int,
5     author_id int,
6     viewer_id int,
7     view_date date
8 );
  > Execute
9 insert into views values
10 (1, 3, 5, '2019-08-01'),
11 (1, 3, 6, '2019-08-02'),
12 (2, 7, 7, '2019-08-01'),
13 (2, 7, 6, '2019-08-02'),
14 (4, 7, 1, '2019-07-22'),
15 (3, 4, 4, '2019-07-21'),
16 (3, 4, 4, '2019-07-21');
  > Execute
17 select distinct author_id as id from views where author_id = viewer_id order by author_id asc;
✓
```

views X

```
select distinct author_id as id from views where author_id = viewer_id order by author_id asc
```

Input to filter result Free 1 Cost: 9ms < 1 > Total 2

id	int
1	4
2	7

Q19.

Table: Delivery

Column Name	Type
delivery_id	Int
customer_id	Int
order_date	date
customer_pref_delivery_date	date

delivery\_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the customer's preferred delivery date is the same as the order date, then the order is called immediately; otherwise, it is called scheduled.

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

The query result format is in the following example.

Input: Delivery

table:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	5	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-11
4	3	2019-08-24	2019-08-26
5	4	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13

Output:

immediate_percentage
33.33

Explanation: The orders with delivery id 2 and 3 are immediate while the others are scheduled.

Solution:

```
select round((select count(*) from delivery where order_date =
customer_pref_delivery_date)/count(*)*100,2) as immediate_percentage from
delivery;
```

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...
4 create table delivery
5 (
6 delivery_id Int,
7 customer_id Int,
8 order_date date,
9 customer_pref_delivery_date date
10 );
11 > Execute
12 insert into delivery values(1, 1, '2019-08-01', '2019-08-02'),
13 (2, 5, '2019-08-02', '2019-08-02'),
14 (3, 1, '2019-08-11', '2019-08-11'),
15 (4, 3, '2019-08-24', '2019-08-26'),
16 (5, 4, '2019-08-21', '2019-08-22'),
17 (6, 2, '2019-08-11', '2019-08-13');
18 > Execute
19 select round((select count(*) from delivery where order_date = customer_pref_delivery_date)/count(*)*100,2)
20 as immediate_percentage from delivery;
```

employee

```
select round((select count(*) from delivery where order_date =
customer_pref_delivery_date)/count(*)*100,2)
as immediate_percentage
newdecimal
```

1	33.33
---	-------



**Q20.**

Table: Ads

Column Name	Type
ad_id	Int
user_id	Int
action	Enum

(ad\_id, user\_id) is the primary key for this table.

Each row of this table contains the ID of an Ad, the ID of a user, and the action taken by this user regarding this Ad.

The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').

A company is running Ads and wants to calculate the performance of each Ad. Performance of the Ad is measured using Click-Through Rate (CTR) where:

$$CTR = \begin{cases} 0, & \text{if Ad total clicks} + \text{Ad total views} = 0 \\ \frac{\text{Ad total clicks}}{\text{Ad total clicks} + \text{Ad total views}} \times 100, & \text{otherwise} \end{cases}$$

Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.

Return the result table ordered by ctr in descending order and by ad\_id in ascending order in case of a tie.

The query result format is in the following example.

Input:

Ads table:

ad_id	user_id	Action
1	1	Clicked
2	2	Clicked
3	3	Viewed
5	5	Ignored
1	7	Ignored
2	7	Viewed
3	5	Clicked
1	4	Viewed
2	11	Viewed
1	2	Clicked

Output:

ad_id	Ctr
1	66.67
3	50
2	33.33
5	0

Explanation:

for ad\_id = 1, ctr = (2/(2+1)) \* 100 = 66.67

for ad\_id = 2, ctr = (1/(1+2)) \* 100 = 33.33

for ad\_id = 3, ctr = (1/(1+1)) \* 100 = 50.00

for ad\_id = 5, ctr = 0.00, Note that ad\_id = 5 has no clicks or views.

Note that we do not care about Ignored Ads.

Solution:

```
select t.ad_id, (case
when base != 0 then round(t.num/t.base*100,2) else 0 end) as Ctr from (select
ad_id,
sum( case when action = 'clicked' or action = 'viewed' then 1 else 0 end) as
base,
sum( case when action = 'clicked' then 1 else 0 end) as num
from ads
group by ad_id)t
order by Ctr desc, t.ad_id asc;
```

```

create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
3 create table ads(
4   ad_id int,
5   user_id int,
6   action varchar(10),
7   primary key(ad_id, user_id)
8 );
> Execute
9 insert into ads values
10 (1, 1, 'Clicked'),
11 (2, 2, 'Clicked'),
12 (3, 3, 'Viewed'),
13 (5, 5, 'Ignored'),
14 (1, 7, 'Ignored'),
15 (2, 7, 'Viewed'),
16 (3, 5, 'Clicked'),
17 (1, 4, 'Viewed'),
18 (2, 11, 'Viewed'),
19 (1, 2, 'Clicked');
> Execute
20 select t.ad_id, (case
21   when base != 0 then round(t.num/t.base*100,2) else 0 end) as Ctr from (select ad_id,
22   sum( case when action = 'clicked' or action = 'viewed' then 1 else 0 end) as base,
23   sum( case when action = 'clicked' then 1 else 0 end) as num
24   from ads
25   group by ad_id)t
26   order by Ctr desc, t.ad_id asc;
27

```

ads

select t.ad\_id, (case  
when base != 0 then round(t.num/t.base\*100,2) else 0 end) as Ctr from (select ad\_id,  
sum( case when action = 'clicked' or action = 'viewed' then 1 else 0 end) as base,  
sum( case when action = 'clicked' then 1 else 0 end) as num  
from ads  
group by ad\_id)t  
order by Ctr desc, t.ad\_id asc;

Cost: 8ms < 1 > Total 4

	ad_id int	Ctr newdecimal
1	1	66.67
2	3	50.00
3	2	33.33
4	5	0

Q21.

Table: Employee

Column Name	Type
employee_id	int
team_id	int

employee\_id is the primary key for this table.

Each row of this table contains the ID of each employee and their respective team.

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

Return result table in any order.

The query result format is in the following example.

Input:

Employee Table:

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

6	9
---	---

Output:

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

Explanation:

Employees with Id 1,2,3 are part of a team with team\_id = 8.

An employee with Id 4 is part of a team with team\_id = 7.

Employees with Id 5,6 are part of a team with team\_id = 9.

Solution:

```
select employee_id, count(team_id) over (partition by team_id) as team_size from
employee order by employee_id;
```

The screenshot shows a MySQL IDE interface. The top panel displays a SQL script in a file named 'create-db-template.sql'. The script includes the following code:

```
1 create database test;
2 use test;
3 create table employee
4 (employee_id int,
5 team_id int
6 );
7 insert into employee values(1, 8),
8 (2, 8),
9 (3, 8),
10 (4, 7),
11 (5, 9),
12 (6, 9);
13 select employee_id, count(team_id) over (partition by team_id) as team_size from employee order by employee_id;
```

The bottom panel shows the query results for the SQL statement: `select employee_id, count(team_id) over (partition by team_id) as team_size from employee order by employee_id`. The results are displayed in a table with columns `employee_id` and `team_size`.

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

The interface also shows a status bar at the bottom indicating 'Cost: 5ms' and 'Total 6' rows.

**Q22.**

Table: Countries

Column Name	Type
country_id	int
country_name	varchar

country\_id is the primary key for this table.

Each row of this table contains the ID and the name of one country.

Table: Weather

Column Name	Type
country_id	int
weather_state	int
day	date

(country\_id, day) is the primary key for this table.

Each row of this table indicates the weather state in a country for one day.

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

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

Return result table in any order.

The query result format is in the following example.

Input: Countries

table:

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

Weather table:

country_id	weather_state	Day
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
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

Output:

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

Explanation:

Average weather\_state in the USA in November is  $(15) / 1 = 15$  so the weather type is Cold.

Average weather\_state in Australia in November is  $(-2 + 0 + 3) / 3 = 0.333$  so the weather type is Cold.

Average weather\_state in Peru in November is  $(25) / 1 = 25$  so the weather type is Hot.

The average weather\_state in China in November is  $(16 + 18 + 21) / 3 = 18.333$  so the weather type is warm.

Average weather\_state in Morocco in November is  $(25 + 27 + 31) / 3 = 27.667$  so the weather type is Hot.

We know nothing about the average weather\_state in Spain in November so we do not include it in the result table.

Solution:

```
select c.country_name, case
when avg(weather_state) <= 15 then 'Cold'
when avg(weather_state) >= 25 then 'Hot'
else 'Warm'
end as weather_state
from
countries c
left join
weather w
on c.country_id = w.country_id
where month(day) = 11
group by c.country_name;
```



create-db-template.sql

config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template

18 (5, 'China'),

19 (8, 'Morocco'),

20 (9, 'Spain');

> Execute

21 insert into weather values

22 (2, 15, '2019-11-01'),

23 (2, 12, '2019-10-28'),

24 (2, 12, '2019-10-27'),

25 (3, -2, '2019-11-10'),

26 (3, 0, '2019-11-11'),

27 (3, 3, '2019-11-12'),

28 (5, 16, '2019-11-07'),

29 (5, 18, '2019-11-09'),

30 (5, 21, '2019-11-23'),

31 (7, 25, '2019-11-28'),

32 (7, 22, '2019-12-01'),

33 (7, 20, '2019-12-02'),

34 (8, 25, '2019-11-05'),

35 (8, 27, '2019-11-15'),

36 (8, 31, '2019-11-25'),

37 (9, 7, '2019-10-23'),

38 (9, 3, '2019-12-23');

> Execute

39 select c.country\_name, case

40 when avg(weather\_state) <= 15 then 'Cold'

41 when avg(weather\_state) >= 25 then 'Hot'

42 else 'Warm'

43 end as weather\_state

44 from

45 countries c

46 left join

47 weather w

48 on c.country\_id = w.country\_id

49 where month(day) = 11

50 group by c.country\_name;

views

select c.country\_name, case

when avg(weather\_state) <= 15 then 'Cold'

when avg(weather\_state) >= 25 then 'Hot'

else 'Warm'

end as weather\_state

from

countries c

left join

weather w

on c.country\_id = w.country\_id

where month(day) = 11

group by c.country\_name;

Free 1

Input to filter result

country\_name varchar weather\_state varchar

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

Cost: 3ms < 1 > Total 5

**Q23.**

Table: Prices

Column Name	Type
product_id	Int
start_date	Date
end_date	Date
Price	Int

(product\_id, start\_date, end\_date) is the primary key for this table.

Each row of this table indicates the price of the product\_id in the period from start\_date to end\_date. For each product\_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product\_id.

Table: UnitsSold

Column Name	Type
product_id	Int
purchase_date	Date
Units	Int

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates the date, units, and product\_id of each product sold.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Prices table:

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

UnitsSold table:

product_id	purchase_date	units
1	2019-02-25	100
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Output:

product_id	average_price
1	6.96
2	16.96

Explanation:

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 =  $((100 * 5) + (15 * 20)) / 115 = 6.96$

Average selling price for product 2 =  $((200 * 15) + (30 * 30)) / 230 = 16.96$

Solution:

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

```

create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
3 create table prices
4 (product_id int,
5 start_date date,
6 end_date date,
7 price int,
8 primary key(product_id, start_date, end_date)
9 );
10 > Execute
11 create table unitssold
12 (product_id int,
13 purchase_date date,
14 units int
15 );
16 > Execute
17 insert into prices VALUES
18 (1, '2019-02-17', '2019-02-28', 5),
19 (1, '2019-03-01', '2019-03-22', 20),
20 (2, '2019-02-01', '2019-02-20', 15),
21 (2, '2019-02-21', '2019-03-31', 30);
22 > Execute
23 insert into unitssold VALUES
24 (1, '2019-02-25', 100),
25 (1, '2019-03-01', 15),
26 (2, '2019-02-10', 200),
27 (2, '2019-03-22', 30);
28 > Execute
29 select p.product_id, round(sum(u.units*p.price)/sum(u.units),2) as average_price
30 from
31 prices p
32 left join
33 unitssold u
34 on p.product_id = u.product_id
35 where u.purchase_date >= start_date and u.purchase_date <= end_date
36 group by product_id
37 order by product_id;

```

Data

```

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

```

Free 1

Cost: 20ms < 1 > Total 2

product_id	int	average_price	newdecimal
1	1	6.96	
2	2	16.96	

Q24.

Table: Activity

Column Name	Type
player_id	Int
device_id	Int
event_date	Date
games_played	Int

(player\_id, event\_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

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

Return the result table in any order.

The query result format is in the following example.

Input: Activity

table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1

3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

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

Solution:

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

The screenshot shows a MySQL client interface with a dark theme. The top panel displays the SQL script being executed, which includes creating an 'activity' table, inserting data, and a query to find the first login for each player. The bottom panel shows the results of the query, which is a table with three rows and two columns: 'player\_id' and 'first\_login'.

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
> Execute
3 create table activity(
4     player_id int,
5     device_id int,
6     event_date date,
7     games_played int,
8     primary key(player_id, event_date));
> Execute
9 insert into activity values
10 (1, 2, '2016-03-01', 5),
11 (1, 2, '2016-05-02', 6),
12 (2, 3, '2017-06-25', 1),
13 (3, 1, '2016-03-02', 0),
14 (3, 4, '2018-07-03', 5);
> Execute
15 select t.player_id, t.event_date as first_login
16 from (select player_id, event_date, row_number() over(partition by player_id order by event_date) as num from activity)t
17 where t.num = 1;
18
```

activity X

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

Cost: 5ms < 1 > Total 3

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

Q25.

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player\_id, event\_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.



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

Input: Activity  
table:

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output:

player_id	device_id
1	2
2	3
3	1

Solution:

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

The screenshot shows a MySQL client interface with a dark theme. At the top, there's a tab labeled 'create-db-template.sql'. Below it, the command line shows the connection path: 'config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >'. The main editor area contains the following SQL code:

```
3 create table activity(
4     player_id int,
5     device_id int,
6     event_date date,
7     games_played int,
8     primary key(player_id, event_date));
9
10 insert into activity values
11 (1, 2, '2016-03-01', 5),
12 (1, 2, '2016-05-02', 6),
13 (2, 3, '2017-06-25', 1),
14 (3, 1, '2016-03-02', 0),
15 (3, 4, '2018-07-03', 5);
16
17 select t.player_id, t.device_id
18 from (select player_id, device_id, row_number() over(partition by player_id order by event_date) as num from activity)t
19 where t.num = 1;
```

Below the editor, there's a section titled 'activity' showing the results of the query. It includes a table with 3 columns: 'player\_id', 'device\_id', and 'games\_played'. The results are:

player_id	device_id	games_played
1	2	5
2	3	1
3	1	0

At the bottom, there's a status bar showing 'Cost: 5ms' and 'Total 3'.

**Q26.**

Table: Products

Column Name	Type
product_id	int
product_name	varchar
product_category	Varchar

product\_id is the primary key for this table.

This table contains data about the company's products.

Table: Orders

Column Name	Type
product_id	Int
order_date	Date
Unit	Int

There is no primary key for this table. It may have duplicate rows.

product\_id is a foreign key to the Products table.

unit is the number of products ordered in order\_date.

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

Return result table in any order.

The query result format is in the following example.

Input: Products

table:

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

Orders table:

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

Output:

product_name	Unit
Leetcode Solutions	130
Leetcode Kit	100

Explanation:

Products with product\_id = 1 is ordered in February a total of  $(60 + 70) = 130$ .

Products with product\_id = 2 is ordered in February a total of 80.

Products with product\_id = 3 is ordered in February a total of  $(2 + 3) = 5$ .

Products with product\_id = 4 was not ordered in February 2020.

Products with product\_id = 5 is ordered in February a total of  $(50 + 50) = 100$ .

Solution:

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

```

create-db-template.sql X
config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
  > Execute
8  create table Orders
9  (product_id int,
10 order_date date,
11 Unit int,
12 foreign key(product_id) references Products(product_id)
13 );
  > Execute
14 insert into Products values
15 (1, 'Leetcode Solutions', 'Book'),
16 (2, 'Jewels of Stringology', 'Book'),
17 (3, 'HP', 'Laptop'),
18 (4, 'Lenovo', 'Laptop'),
19 (5, 'Leetcode Kit', 'T-shirt');
  > Execute
20 insert into Orders values
21 (1, '2020-02-05', 60),
22 (1, '2020-02-10', 70),
23 (2, '2020-01-18', 30),
24 (2, '2020-02-11', 80),
25 (3, '2020-02-17', 2),
26 (3, '2020-02-24', 3),
27 (4, '2020-03-01', 20),
28 (4, '2020-03-04', 30),
29 (4, '2020-03-04', 60),
30 (5, '2020-02-25', 50),
31 (5, '2020-02-27', 50),
32 (5, '2020-03-01', 50);
  > Execute
33 select p.product_name, sum(o.unit) as unit
34 from
35 Products p
36 left join
37 Orders o
38 on p.product_id = o.product_id
39 where month(o.order_date) = 2 and year(o.order_date) = 2020
40 group by p.product_id
41 having unit >= 100;
42
Product X
select p.product_name, sum(o.unit) as unit
from
Q Input to filter result
Free 1
product_name varchar unit newdecimal
1 Leetcode Solutions 130
2 Leetcode Kit 100
Cost: 2ms < 1

```

Q27.

Table: Users

Column Name	Type
user_id	Int
Name	Varchar
Mail	Varchar

user\_id is the primary key for this table.

This table contains information of the users signed up in a website. Some emails are invalid.

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

valid e-mail has a prefix name and a domain where:

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

Return the result table in any order.

The query result format is in the following example.

Input:

Users table:

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

Output:

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

Explanation:

The mail of user 2 does not have a domain.

The mail of user 5 has the # sign which is not allowed.

The mail of user 6 does not have the leetcode domain.

The mail of user 7 starts with a period.

Solution:

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

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@
  > Execute
2  use test;
  > Execute
3  create table Users
4  (user_id    int,
5   name      varchar(15),
6   mail      varchar(30)
7  );
  > Execute
8  insert into Users values
9  (1, 'Winston', 'winston@leetcode.com'),
10 (2, 'Jonathan', 'jonathanisgreat'),
11 (3, 'Annabelle', 'bella-@leetcode.com'),
12 (4, 'Sally', 'sally.come@leetcode.com'),
13 (5, 'Marwan', 'quarz#2020@leetcode.com'),
14 (6, 'David', 'david69@gmail.com'),
15 (7, 'Shapiro', '.shapo@leetcod e.com');
  > Execute
✓ 16 select user_id, name, mail from Users
17 where
18 mail regexp '^[a-zA-Z]+[a-zA-Z0-9_\.\\-]*@leetcode[\.]com'
19 order by user_id;
20
```

Users X

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

Input to filter result

Free 1

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

Q28.

Table: Customers

Column Name	Type
customer_id	Int
Name	Varchar
Country	Varchar

customer\_id is the primary key for this table.

This table contains information about the customers in the company.

Table: Product



Column Name	Type
customer_id	Int
Name	Varchar
Country	Varchar

product\_id is the primary key for this table.

This table contains information on the products in the company.

price is the product cost.

Table: Orders

Column Name	Type
order_id	Int
customer_id	Int
product_id	Int
order_date	Date
Quantity	Int

order\_id is the primary key for this table.

This table contains information on customer orders.

customer\_id is the id of the customer who bought "quantity" products with id "product\_id".

Order\_date is the date in format ('YYYY-MM-DD') when the order was shipped.

Write an SQL query to report the customer\_id and customer\_name of customers who have spent at least \$100 in each month of June and July 2020.

Return the result table in any order.

The query result format is in the following example.

Input:

Customers table:

customer_id	Name	Country
1	Winston	USA
2	Jonathan	Peru
3	Moustafa	Egypt

Product table:

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

Orders table:

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

Output:

customer_id	Name
1	Winston

Explanation:

Winston spent \$300 ( $300 * 1$ ) in June and \$100 ( $10 * 1 + 45 * 2$ ) in July 2020.

Jonathan spent \$600 ( $300 * 2$ ) in June and \$20 ( $2 * 10$ ) in July 2020.

Moustafa spent \$110 ( $10 * 2 + 45 * 2$ ) in June and \$0 in July 2020.

Solution:

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

```

create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...
17 order_date Date,
18 quantity Int,
19 primary key(order_id)
20 );
  > Execute
21 insert into Customers values
22 (1, 'Winston', 'USA'),
23 (2, 'Jonathan', 'Peru'),
24 (3, 'Moustafa', 'Egypt');
  > Execute
25 insert into Product values
26 (10, 'LC Phone', 300),
27 (20, 'LC T-Shirt', 10),
28 (30, 'LC Book', 45),
29 (40, 'LC Keychain', 2);
  > Execute
30 insert into Orders values
31 (1, 1, 10, '2020-06-10', 1),
32 (2, 1, 20, '2020-07-01', 1),
33 (3, 1, 30, '2020-07-08', 2),
34 (4, 2, 10, '2020-06-15', 2),
35 (5, 2, 40, '2020-07-01', 10),
36 (6, 3, 20, '2020-06-24', 2),
37 (7, 3, 30, '2020-06-25', 2),
38 (9, 3, 30, '2020-05-08', 3);
  > Execute
39 select t.customer_id, t.name from
40 (select c.customer_id, c.name,
41 sum(case when month(o.order_date) = 6 and year(o.order_date) = 2020 then p.price*o.quantity else 0 end) as june_spent,
42 sum(case when month(o.order_date) = 7 and year(o.order_date) = 2020 then p.price*o.quantity else 0 end) as july_spent
43 from
44 Orders o
45 left join
46 Product p
47 on o.product_id = p.product_id
48 left join
49 Customers c
50 on o.customer_id = c.customer_id
51 group by c.customer_id) t
52 where june_spent >= 100 and july_spent >= 100;
53
Data
select t.customer_id, t.name from
(select c.customer_id, c.name
+
Q Input to filter result
Free 1
Cost: 5ms < 1 > Total 1
customer_id name
int varchar
1 1 Winston

```

Q29.

Table: TVProgram

Column Name	Type
program_date	Date
content_id	Int
channel	Varchar

(program\_date, content\_id) is the primary key for this table.  
 This table contains information about the programs on the TV.  
 content\_id is the id of the program in some channel on the TV.

Table: Content

Column Name	Type
content_id	Varchar
Title	Varchar
Kids_content	Enum
content_type	Varchar

content\_id is the primary key for this table.  
 Kids\_content is an enum that takes one of the values ('Y', 'N') where:

'Y' means content for kids, otherwise 'N' is not content for kids.  
content\_type is the category of the content as movies, series, etc.

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.

The query result format is in the following example.

Input: TVProgram  
table:

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

Content table:

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

Output:

title
Aladdin

Explanation:

"Leetcode Movie" is not a content for kids.

"Alg. for Kids" is not a movie.

"Database Sols" is not a movie

"Alladin" is a movie, content for kids and was streamed in June 2020.

"Cinderella" was not streamed in June 2020.

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

create-db-template.sql X

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...

3 create table TVProgram

4 (program\_date Date,

5 content\_id Int,

6 channel Varchar(20),

7 primary key(program\_date, content\_id)

8 );

▷ Execute

9 create table Content

10 (content\_id int,

11 Title Varchar(20),

12 Kids\_content Varchar(1),

13 content\_type Varchar(15),

14 primary key(content\_id)

15 );

▷ Execute

16 insert into TVProgram values

17 ('2020-06-10 08:00', 1, 'LC-Channel'),

18 ('2020-05-11 12:00', 2, 'LC-Channel'),

19 ('2020-05-12 12:00', 3, 'LC-Channel'),

20 ('2020-05-13 14:00', 4, 'Disney Ch'),

21 ('2020-06-18 14:00', 4, 'Disney Ch'),

22 ('2020-07-15 16:00', 5, 'Disney Ch');

▷ Execute

23 insert into Content values

24 (1, 'Leetcode Movie', 'N', 'Movies'),

25 (2, 'Alg. for Kids', 'Y', 'Series'),

26 (3, 'Database Sols', 'N', 'Series'),

27 (4, 'Aladdin', 'Y', 'Movies'),

28 (5, 'Cinderella', 'Y', 'Movies');

▷ Execute

✓ 29 select c.Title from

30 Content c

31 left join

32 TVProgram t

33 on c.content\_id = t.content\_id

34 where c.Kids\_content = 'Y' and c.content\_type = 'Movies' and month(t.program\_date) = 6 and year(t.program\_date) = 2020;

~

TVProgram X

select c.Title from

Content c

+

🔒

🔍

Input to filter result

Free

📧

🔄

⊕

⊖

🗑️

🔧

💬

⬆️

⬇️

⬆️

▶️

Cost: 3ms

<

1

>

Total 1

✓

🔍

Title

varchar

1

Aladdin

**Q30.**

Table: NPV

Column Name	Type
Id	Int
Year	Int
Npv	Int

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory and the corresponding net present value.

Table: Queries

Column Name	Type
Id	Int
Year	Int

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory query.

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

The query result format is in the following example.

Input:

NPV table:

Id	Year	Npv
1	2018	100
7	2020	30
13	2019	40
1	2019	113
2	2008	121
3	2009	12
11	2020	99
7	2019	0

Queries table:

Id	Year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

Output:

Id	Year	Npv
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

Explanation:

The npv value of (7, 2018) is not present in the NPV table, we consider it 0. The npv values of all other queries can be found in the NPV table.

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



create-db-template.sql X

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
  ✨ Active Connection | > Execute
1  use test;
  > Execute
2  create table NPV
3  (
4  Id Int,
5  Year Int,
6  Npv Int,
7  PRIMARY KEY(Id, Year)
8  );
  > Execute
9  create table Queries
10 (
11 Id Int,
12 Year Int,
13 PRIMARY KEY(Id, Year)
14 );
  > Execute
15 insert into NPV values
16 (1, 2018, 100),
17 (7, 2020, 30),
18 (13, 2019, 40),
19 (1, 2019, 113),
20 (2, 2008, 121),
21 (3, 2009, 12),
22 (11, 2020, 99),
23 (7, 2019, 0);
  > Execute
24 insert into Queries values
25 (1, 2019),
26 (2, 2008),
27 (3, 2009),
28 (7, 2018),
29 (7, 2019),
30 (7, 2020),
31 (13, 2019);
  > Execute
32 select q.*, coalesce(n.Npv,0) as Npv
33 from
34 Queries q
35 left join
36 NPV n
37 on q.Id = n.Id and q.Year = n.Year;
38

```

Data X

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

from

Q Input to filter result

Free

Cost: 4ms < 1 > Total 7

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

Q31.

Table: NPV

Column Name	Type
Id	Int
Year	Int
Npv	Int

(id, year) is the primary key of this table.

The table has information about the id and the year of each inventory and the corresponding net present value.

Table: Queries

Column Name	Type
Id	Int
Year	Int

(id, year) is the primary key of this table.  
The table has information about the id and the year of each inventory query.

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

The query result format is in the following example.

Input:

NPV table:

Id	Year	Npv
1	2018	100
7	2020	30
13	2019	40
1	2019	113
2	2008	121
3	2009	12
11	2020	99
7	2019	0

Queries table:

Id	Year
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

Output:

Id	Year	Npv
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

Explanation:

The npv value of (7, 2018) is not present in the NPV table, we consider it 0. The npv values of all other queries can be found in the NPV table.

Solution:

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

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...
  * Active Connection | > Execute
1 use test;
  > Execute
2 create table NPV
3 (Id Int,
4 Year Int,
5 Npv Int,
6 PRIMARY KEY(Id, Year)
7 );
  > Execute
8 create table Queries
9 (Id Int,
10 Year Int,
11 PRIMARY KEY(Id, Year)
12 );
  > Execute
13 insert into NPV values
14 (1, 2018, 100),
15 (7, 2020, 30),
16 (13, 2019, 40),
17 (1, 2019, 113),
18 (2, 2008, 121),
19 (3, 2009, 12),
20 (11, 2020, 99),
21 (7, 2019, 0);
  > Execute
22 insert into Queries values
23 (1, 2019),
24 (2, 2008),
25 (3, 2009),
26 (7, 2018),
27 (7, 2019),
28 (7, 2020),
29 (13, 2019);
  > Execute
31 select q.*, coalesce(n.Npv,0) as Npv
32 from
33 Queries q
34 left join
35 NPV n
36 on q.Id = n.Id and q.Year = n.Year;
37
38
```

Data				
select q.*, coalesce(n.Npv,0) as Npv				
from				
Input to filter result				
	Id	Year	Npv	
	int	int	bigint	
1	1	2019	113	
2	2	2008	121	
3	3	2009	12	
4	7	2018	0	
5	7	2019	0	
6	7	2020	30	
7	13	2019	40	

**Q32.**

Table: Employees

Column Name	Type
Id	Int
Name	Varchar

id is the primary key for this table.

Each row of this table contains the id and the name of an employee in a company.

Table: EmployeeUNI

Column Name	Type
Id	Int
unique_id	Int

(id, unique\_id) is the primary key for this table.

Each row of this table contains the id and the corresponding unique id of an employee in the company.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Employees table:

Id	Name
1	Alice
7	Bob
11	Meir
90	Winston
3	Jonathan

EmployeeUNI table:

Id	unique_id
3	1
11	2
90	3

Output:

unique_id	Name
Null	Alice
Null	Bob
2	Meir
3	Winston
1	Jonathan

Explanation:

Alice and Bob do not have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

Solution:

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

The screenshot shows a MySQL client interface with a SQL editor and a results pane. The SQL editor contains the following code:

```
create table employeeUNI
(id Int,
unique_id Int,
primary key(id, unique_id)
);
insert into employees values
(1, 'Alice'),
(7, 'Bob'),
(11, 'Meir'),
(90, 'Winston'),
(3, 'Jonathan');
insert into employeeUNI values
(3, 1),
(11, 2),
(90, 3);
select u.unique_id, e.name
from
employees e
left join
employeeUNI u
on e.id = u.id;
```

The results pane shows the output of the query:

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

**Q33.**

Table: Users

Column Name	Type
Id	Int
Name	Varchar

id is the primary key for this table.

name is the name of the user.

Table: Rides

Column Name	Type
Id	Int
user_id	Int
Distance	Int

id is the primary key for this table.

user\_id is the id of the user who travelled the distance "distance".

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.

The query result format is in the following example.

Input: Users

table:

Id	Name
1	Alice
2	Bob
3	Alex
4	Donald
7	Lee

13	Jonathan
19	Elvis

Rides table:

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

Output:

name	travelled_distance
Elvis	450
Lee	450
Bob	317
Jonathan	312
Alex	222
Alice	120
Donald	0

Explanation:

Elvis and Lee travelled 450 miles, Elvis is the top traveller as his name is alphabetically smaller than Lee.

Bob, Jonathan, Alex, and Alice have only one ride and we just order them by the total distances of the ride.

Donald did not have any rides, the distance travelled by him is 0.

Solution:

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



create-db-template.sql

config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...

11 distance Int,

12 primary key(id));

> Execute

13 insert into users values

14 (1, 'Alice'),

15 (2, 'Bob'),

16 (3, 'Alex'),

17 (4, 'Donald'),

18 (7, 'Lee'),

19 (13, 'Jonathan'),

20 (19, 'Elvis');

> Execute

21 insert into rides values

22 (1, 1, 120),

23 (2, 2, 317),

24 (3, 3, 222),

25 (4, 7, 100),

26 (5, 13, 312),

27 (6, 19, 50),

28 (7, 7, 120),

29 (8, 19, 400),

30 (9, 7, 230);

> Execute

31 select u.name, coalesce(sum(r.distance),0) as travelled\_distance

32 from

33 users u

34 left join

35 rides r

36 on u.id = r.user\_id

37 group by u.name

38 order by travelled\_distance desc, u.name;

Data

select u.name, coalesce(sum(r.distance),0) as travelled\_distance

from

Q Input to filter result

Free

Cost: 3ms < 1 > Total 7

name

varchar

travelled\_distance

newdecimal

1

Elvis

450

2

Lee

450

3

Bob

317

4

Jonathan

312

5

Alex

222

6

Alice

120

7

Donald

0

**Q34.**

Table: Products

Column Name	Type
product_id	Int
product_name	Varchar
product_category	Varchar

product\_id is the primary key for this table.

This table contains data about the company's products.

Table: Orders

Column Name	Type
product_id	Int
order_date	Date
Unit	Int

There is no primary key for this table. It may have duplicate rows.

product\_id is a foreign key to the Products table.

unit is the number of products ordered in order\_date.

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

Return result table in any order.

The query result format is in the following example.

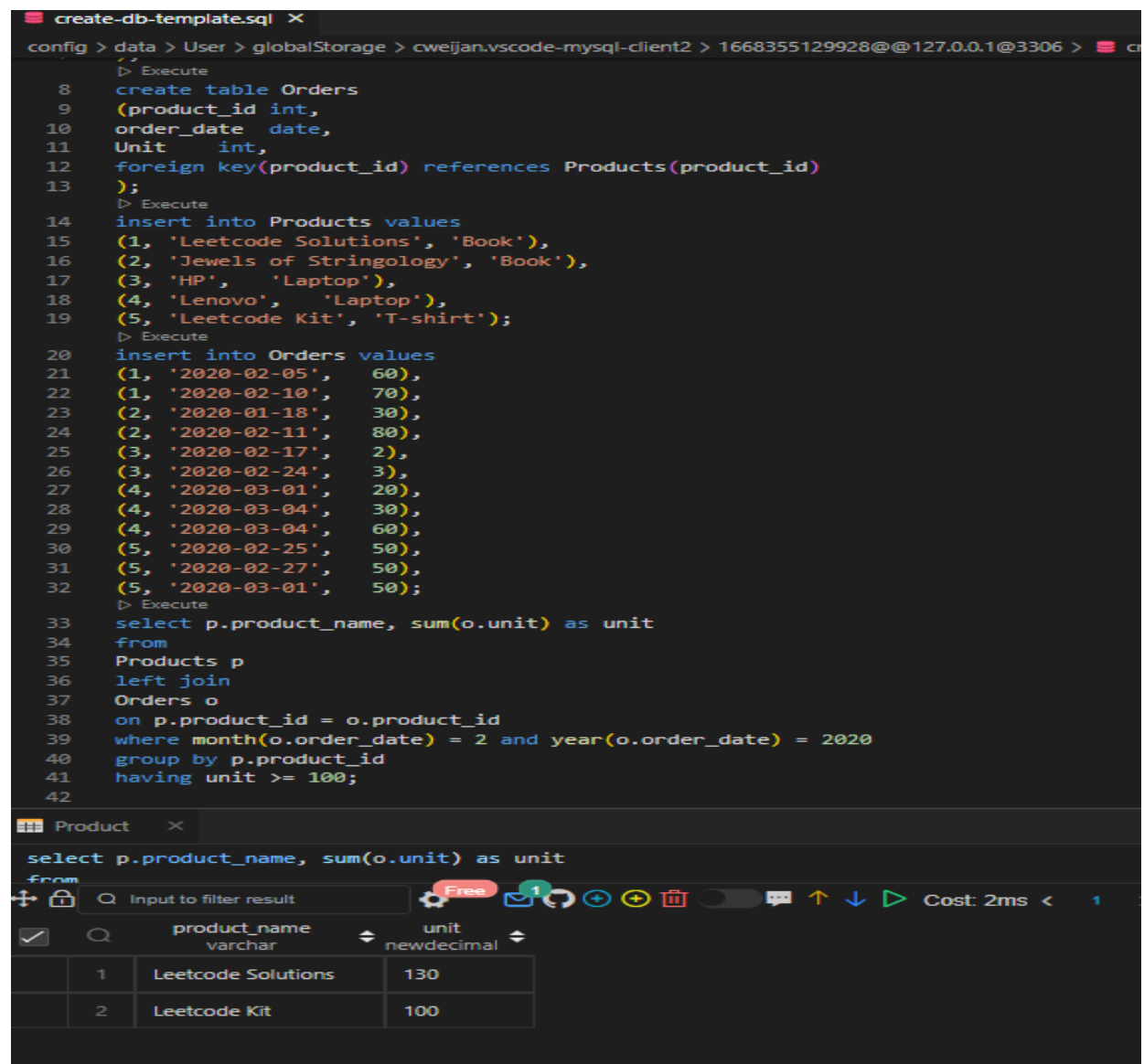
Input: Products

table:

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

Solution:

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



The screenshot shows a MySQL client interface with a dark theme. The top panel displays the SQL script being executed, which includes creating an 'Orders' table, inserting data into 'Products' and 'Orders' tables, and running a query to find products with a total unit count of 100 or more in February 2020. The bottom panel shows the results of the query in a table format.

```
create-db-template.sql X
config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
8 create table Orders
9 (product_id int,
10 order_date date,
11 Unit int,
12 foreign key(product_id) references Products(product_id)
13 );
14 insert into Products values
15 (1, 'Leetcode Solutions', 'Book'),
16 (2, 'Jewels of Stringology', 'Book'),
17 (3, 'HP', 'Laptop'),
18 (4, 'Lenovo', 'Laptop'),
19 (5, 'Leetcode Kit', 'T-shirt');
20 insert into Orders values
21 (1, '2020-02-05', 60),
22 (1, '2020-02-10', 70),
23 (2, '2020-01-18', 30),
24 (2, '2020-02-11', 80),
25 (3, '2020-02-17', 2),
26 (3, '2020-02-24', 3),
27 (4, '2020-03-01', 20),
28 (4, '2020-03-04', 30),
29 (4, '2020-03-04', 60),
30 (5, '2020-02-25', 50),
31 (5, '2020-02-27', 50),
32 (5, '2020-03-01', 50);
33 select p.product_name, sum(o.unit) as unit
34 from
35 Products p
36 left join
37 Orders o
38 on p.product_id = o.product_id
39 where month(o.order_date) = 2 and year(o.order_date) = 2020
40 group by p.product_id
41 having unit >= 100;
42
```

	product_name	unit
1	Leetcode Solutions	130
2	Leetcode Kit	100

Q35.

Table: Movies

Column Name	Type
movie_id	Int
Title	Varchar

movie\_id is the primary key for this table. The title is the name of the movie.

Table: Users

Column Name	Type
user_id	Int
Name	Varchar

user\_id is the primary key for this table.

Table: MovieRating

Column Name	Type
movie_id	Int
user_id	Int
Rating	Int
created_at	Date

(movie\_id, user\_id) is the primary key for this table.  
This table contains the rating of a movie by a user in their review.  
created\_at is the user's review date.

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.

The query result format is in the following example.

Input:

Movies table:

movie_id	Title
1	Avengers
2	Frozen 2
3	Joker

Users table:

user_id	Name
1	Daniel
2	Monica
3	Maria
4	James

MovieRating table:

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

Output:

Results
Daniel
Frozen 2

Explanation:

Daniel and Monica have rated 3 movies ("Avengers", "Frozen 2" and "Joker") but Daniel is smaller lexicographically.

Frozen 2 and Joker have a rating average of 3.5 in February but Frozen 2 is smaller lexicographically.

Solution:

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

The screenshot shows a MySQL client window with a SQL editor and a results pane. The SQL editor contains a script to create a database template, insert data into Movies, Users, and MovieRating tables, and then execute two queries. The first query ranks users by the number of ratings they have, and the second query ranks movies by their average rating in February 2020. The results pane shows the output of the first query, displaying the names of the top two users: Daniel and Frozen 2.

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...

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

Data X
(select t1.name as Results from
(select u.name, count(u.user_id), dense_rank() over(order by count(user_id) desc, u.name) as r1
Results
varchar
1 Daniel
2 Frozen 2
```

**Q36.**

Table: Users

Column Name	Type
Id	Int
name	varchar

id is the primary key for this table.

name is the name of the user.

Table: Rides

Column Name	Type
Id	int
user_id	int
distance	int

id is the primary key for this table.

user\_id is the id of the user who travelled the distance "distance".

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.

The query result format is in the following example.

Input: Users

table:

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

Rides table:

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

Output:

name	travelled_distance
Elvis	450
Lee	450
Bob	317
Jonathan	312
Alex	222
Alice	120
Donald	0

Explanation:

Elvis and Lee travelled 450 miles, Elvis is the top traveller as his name is alphabetically smaller than Lee.

Bob, Jonathan, Alex, and Alice have only one ride and we just order them by the total distances of the ride.

Donald did not have any rides, the distance travelled by him is 0.

Solution:

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



```

create-db-template.sql x
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...

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

```

Data

```

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

```

Input to filter result

Cost: 3ms < 1 > Total 7

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

Q37.

Table: Employees

Column Name	Type
id	int
name	varchar

id is the primary key for this table.

Each row of this table contains the id and the name of an employee in a company.

Table: EmployeeUNI

Column Name	Type
id	int
unique_id	int

(id, unique\_id) is the primary key for this table.

Each row of this table contains the id and the corresponding unique id of an employee in the company.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Employees table:

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

EmployeeUNI table:

id	unique_id
3	1
11	2
90	3

Output:

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

Explanation:

Alice and Bob do not have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

Solution:

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

create-db-template.sql

config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-templ

8

create table employeeUNI

9

(id Int,

10

unique\_id Int,

11

primary key(id, unique\_id)

12

);

13

insert into employees values

14

(1, 'Alice'),

15

(7, 'Bob'),

16

(11, 'Meir'),

17

(90, 'Winston'),

18

(3, 'Jonathan');

19

insert into employeeUNI values

20

(3, 1),

21

(11, 2),

22

(90, 3);

23

select u.unique\_id, e.name

24

from

25

employees e

26

left join

27

employeeUNI u

28

on e.id = u.id;

29

Data

select u.unique\_id, e.name

from

Q Input to filter result

Free

1

Cost: 0ms

1

Total 5

unique\_id

int

name

varchar

1	(NULL)	Alice
2	(NULL)	Bob
3	1	Jonathan
4	2	Meir
5	3	Winston

**Q38.**

Table: Departments

Column Name	Type
Id	int
name	varchar

id is the primary key of this table.

The table has information about the id of each department of a university.

Table: Students

Column Name	Type
Id	int
name	varchar
department_id	int

id is the primary key of this table.

The table has information about the id of each student at a university and the id of the department he/she studies at.

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

Return the result table in any order.

The query result format is in the following example.

Input: Departments

table:

id	Name
1	Electrical Engineering
7	Computer Engineering
13	Business Administration

Students table:

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

Output:

id	name
2	John
7	Daiana
4	Jasmine
3	Steve

Explanation:

John, Daiana, Steve, and Jasmine are enrolled in departments 14, 33, 74, and 77 respectively. Department 14, 33, 74, and 77 do not exist in the Departments table.

Solution:

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

The screenshot shows a MySQL client window with a SQL script being executed. The script creates two tables, 'Departments' and 'Students', and inserts data into them. The 'Departments' table has columns 'Id' (int) and 'name' (varchar(30)). The 'Students' table has columns 'Id' (int), 'name' (varchar(20)), and 'department\_id' (int). The script inserts data into both tables. Finally, it executes a query to select the 'id' and 'name' of students whose 'department\_id' is not in the 'id' column of the 'Departments' table. The query results are displayed in a table below the script.

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-ten
  > Execute
3 create table Departments
4 (Id int,
5 name varchar(30),
6 primary key(Id)
7 );
  > Execute
8 create table Students
9 (Id int,
10 name varchar(20),
11 department_id int,
12 primary key(Id)
13 );
  > Execute
14 insert into Departments values
15 (1, 'Electrical Engineering'),
16 (7, 'Computer Engineering'),
17 (13, 'Business Administration');
  > Execute
18 insert into Students values
19 (23, 'Alice', 1),
20 (1, 'Bob', 7),
21 (5, 'Jennifer', 13),
22 (2, 'John', 14),
23 (4, 'Jasmine', 77),
24 (3, 'Steve', 74),
25 (6, 'Luis', 1),
26 (8, 'Jonathan', 7),
27 (7, 'Daiana', 33),
28 (11, 'Madelynn', 1);
  > Execute
29 select id, name from Students
30 where department_id not in (select id from Departments);
31
Users X
select id, name from Students
where department_id not in (select id from Departments)
Input to filter result
Free
Cost: 3ms < 1 > Total 4
id int name varchar
1 2 John
2 3 Steve
3 4 Jasmine
4 7 Daiana
```

Q39.

Table: Calls

Column Name	Type
from_id	int
to_id	int
duration	int

This table does not have a primary key, it may contain duplicates.

This table contains the duration of a phone call between from\_id and to\_id.

from\_id != to\_id

Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2.

Return the result table in any order.

The query result format is in the following example.

Input:

Calls table:

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

Output:

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

Explanation:

Users 1 and 2 had 2 calls and the total duration is 70 (59 + 11).

Users 1 and 3 had 1 call and the total duration is 20.

Users 3 and 4 had 4 calls and the total duration is 999 (100 + 200 + 200 + 499).

Solution:

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

The screenshot shows a database client interface with a SQL editor and a results pane. The SQL editor contains the following code:

```
3 create table Calls
4 (from_id int,
5 to_id int,
6 duration int
7 );
8 insert into Calls values
9 (1, 2, 59),
10 (2, 1, 11),
11 (1, 3, 20),
12 (3, 4, 100),
13 (3, 4, 200),
14 (3, 4, 200),
15 (4, 3, 499);
16 select t.person1, t.person2, count(*) as call_count, sum(t.duration) as total_duration
17 from
18 (select duration,
19 case when from_id < to_id then from_id else to_id end as person1,
20 case when from_id > to_id then from_id else to_id end as person2
21 from Calls) t
22 group by t.person1, t.person2;
```

The results pane shows the following table:

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

Q40.

Table: Prices

Column Name	Type
product_id	Int
start_date	Date
end_date	Date

Price	Int
-------	-----

(product\_id, start\_date, end\_date) is the primary key for this table.

Each row of this table indicates the price of the product\_id in the period from start\_date to end\_date. For each product\_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product\_id.

Table: UnitsSold

Column Name	Type
product_id	Int
purchase_date	Date
Units	Int

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates the date, units, and product\_id of each product sold.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Prices table:

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

UnitsSold table:



product_id	purchase_date	Units
1	2019-02-25	100
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Output:

product_id	average_price
1	6.96
2	16.96

Explanation:

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 =  $((100 * 5) + (15 * 20)) / 115 = 6.96$

Average selling price for product 2 =  $((200 * 15) + (30 * 30)) / 230 = 16.96$

Solution:

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

The screenshot shows a MySQL database client interface with a dark theme. The top panel displays a series of SQL queries executed in a script editor. The queries create two tables: 'prices' and 'unitssold'. The 'prices' table has columns for product\_id, start\_date, end\_date, and price. The 'unitssold' table has columns for product\_id, purchase\_date, and units. Data is inserted into both tables. The final query is a SELECT statement that calculates the average price for each product, rounded to two decimal places, using a LEFT JOIN between the 'prices' and 'unitssold' tables. The bottom panel shows the results of this query in a table format with columns 'product\_id' and 'average\_price'.

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
3 create table prices
4 (product_id int,
5 start_date date,
6 end_date date,
7 price int,
8 primary key(product_id, start_date, end_date)
9 );
10 > Execute
11 create table unitssold
12 (product_id int,
13 purchase_date date,
14 units int
15 );
16 > Execute
17 insert into prices VALUES
18 (1, '2019-02-17', '2019-02-28', 5),
19 (1, '2019-03-01', '2019-03-22', 20),
20 (2, '2019-02-01', '2019-02-20', 15),
21 (2, '2019-02-21', '2019-03-31', 30);
22 > Execute
23 insert into unitssold VALUES
24 (1, '2019-02-25', 100),
25 (1, '2019-03-01', 15),
26 (2, '2019-02-10', 200),
27 (2, '2019-03-22', 30);
28 > Execute
29 select p.product_id, round(sum(u.units*p.price)/sum(u.units),2) as average_price
30 from
31 prices p
32 left join
33 unitssold u
34 on p.product_id = u.product_id
35 where u.purchase_date >= start_date and u.purchase_date <= end_date
36 group by product_id
37 order by product_id;
```

Results:

product_id	average_price
1	6.96
2	16.96

**Q41.**

Table: Warehouse

Column Name	Type
Name	Varchar
product_id	Int
Units	Int

(name, product\_id) is the primary key for this table.

Each row of this table contains the information of the products in each warehouse.

Table: Products

Column Name	Type
product_id	Int
product_name	Varchar
Width	Int
Length	Int
Height	Int

product\_id is the primary key for this table.

Each row of this table contains information about the product dimensions (Width, Length, and Height) in feet of each product.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Warehouse table:

Name	product_id	Units
LCHouse1	1	1
LCHouse1	2	10
LCHouse1	3	5
LCHouse2	1	2
LCHouse2	2	2
LCHouse3	4	1

Products table:

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

Output:

warehouse_name	Volume
LCHouse1	12250
LCHouse2	20250
LCHouse3	800

Solution:

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



**Q42.**

Table: Sales

Column Name	Type
sale_date	date
Fruit	enum
sold_num	int

(sale\_date, fruit) is the primary key for this table.

This table contains the sales of "apples" and "oranges" sold each day.

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.

The query result format is in the following example.

Input:

Sales table:

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

Output:

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

Explanation:

Day 2020-05-01, 10 apples and 8 oranges were sold (Difference  $10 - 8 = 2$ ).

Day 2020-05-02, 15 apples and 15 oranges were sold (Difference  $15 - 15 = 0$ ).

Day 2020-05-03, 20 apples and 0 oranges were sold (Difference  $20 - 0 = 20$ ).

Day 2020-05-04, 15 apples and 16 oranges were sold (Difference  $15 - 16 = -1$ ).

**Solution:**

```
select t.sale_date, (t.apples_sold - t.oranges_sold) as diff
from
(select sale_date,
  max(CASE WHEN fruit = 'apples' THEN sold_num ELSE 0 END )as apples_sold,
  max(CASE WHEN fruit = 'oranges' THEN sold_num ELSE 0 END )as oranges_sold
FROM sales
group by sale_date) t
ORDER BY t.sale_date;
```

create-db-template.sql X

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql >

```
2 create table sales
3 (sale_date date,
4 fruit varchar(10),
5 sold_num int,
6 primary key(sale_date, fruit));
7 insert into sales values
8 ('2020-05-01', 'apples', 10),
9 ('2020-05-01', 'oranges', 8),
10 ('2020-05-02', 'apples', 15),
11 ('2020-05-02', 'oranges', 15),
12 ('2020-05-03', 'apples', 20),
13 ('2020-05-03', 'oranges', 0),
14 ('2020-05-04', 'apples', 15),
15 ('2020-05-04', 'oranges', 16);
16 select t.sale_date, (t.apples_sold - t.oranges_sold) as diff
17 from
18 (select sale_date,
19 max(CASE WHEN fruit = 'apples' THEN sold_num ELSE 0 END )as apples_sold,
20 max(CASE WHEN fruit = 'oranges' THEN sold_num ELSE 0 END )as oranges_sold
21 FROM sales
22 group by sale_date) t
23 ORDER BY t.sale_date;
24
```

sales X

select t.sale\_date, (t.apples\_sold - t.oranges\_sold) as diff  
from

Q Input to filter result Free 1

Cost: 49ms < 1 > Total 4

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

Solution:

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

create-db-template.sql

config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.s

Execute

```
2 create table sales
3 (sale_date date,
4 fruit varchar(10),
5 sold_num int,
6 primary key(sale_date, fruit));
7
8 insert into sales values
9 ('2020-05-01', 'apples', 10),
10 ('2020-05-01', 'oranges', 8),
11 ('2020-05-02', 'apples', 15),
12 ('2020-05-02', 'oranges', 15),
13 ('2020-05-03', 'apples', 20),
14 ('2020-05-03', 'oranges', 0),
15 ('2020-05-04', 'apples', 15),
16 ('2020-05-04', 'oranges', 16);
17
18 select sale_date,
19 sum(case when fruit = 'apples' then sold_num
20 else (-sold_num) end) as diff
21 from sales
22 group by sale_date;
```

sales

select sale\_date,

sum(case when fruit = 'apples' then sold\_num

Input to filter result

Free

1

Cost: 4ms < 1 > Total 4

sale\_date

diff

date

newdecimal

1	2020-05-01	2
2	2020-05-02	0
3	2020-05-03	20
4	2020-05-04	-1

**Q43.**

Table: Activity

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

(player\_id, event\_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

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.

The query result format is in the following example.

Input: Activity

table:

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

Output:

fraction
0.33

Explanation:

Only the player with id 1 logged back in after the first day he had logged in so the answer is  $1/3 = 0.33$

Solution:

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



create-db-template.sql X

config > data > User > globalStorage > cweijian.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...  
▷ execute

```
3 create table activity
4 (player_id int,
5 device_id int,
6 event_date date,
7 games_played int,
8 primary key(player_id, event_date)
9 );
```

▷ Execute

```
10 insert into activity VALUES
11 (1, 2, '2016-03-01', 5),
12 (1, 2, '2016-03-02', 6),
13 (2, 3, '2017-06-25', 1),
14 (3, 1, '2016-03-02', 0),
15 (3, 4, '2018-07-03', 5);
```

▷ Execute

```
✓ 16 select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
17 from
18 (
19 select distinct player_id,
20 datediff(event_date, lead(event_date, 1) over(partition by player_id order by event_date)) as diff
21 from activity ) t
22 where diff = -1;
```

activity X

select round(t.player\_id/(select count(distinct player\_id) from activity),2) as fraction

from

🔍 Input to filter result

Free

1

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

+

✓

Q

fraction

newdecimal

1

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

0.33

**Q44.**

Table: Employee

Column Name	Type
Id	int
Name	varchar
Department	varchar
managerId	int

id is the primary key column for this table.

Each row of this table indicates the name of an employee, their department, and the id of their manager.

If managerId is null, then the employee does not have a manager.

No employee will be the manager of themselves.

Write an SQL query to report the managers with at least five direct reports.

Return the result table in any order.

The query result format is in the following example.

Input:

Employee table:

Id	name	department	managerId
101	John	A	None
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

Output:

name
John

**Solution:**

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

create-db-template.sql X

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306

```
3 create table employee
4 (id int,
5  name   varchar(10),
6  department varchar(10),
7  managerId int,
8  primary key(id)
9 );
```

▷ Execute

```
10 insert into employee values
11 (101, 'John', 'A', Null),
12 (102, 'Dan', 'A', 101),
13 (103, 'James', 'A', 101),
14 (104, 'Amy', 'A', 101),
15 (105, 'Anne', 'A', 101),
16 (106, 'Ron', 'B', 101);
```

▷ Execute

```
17 select t.name from
18 (select a.id, a.name, count(b.managerID) as no_of_direct_reports from
19  employee a
20  INNER JOIN
21  employee b
22  on a.id = b.managerID
23  group by b.managerID) t
24 where no_of_direct_reports >= 5
25 order by t.name;
26
```

Data X

```
select t.name from
(select a.id, a.name, count(b.managerID) as no of direct reports from
```

✚ 🔒 🔍 Input to filter result ⚙️ Free 1 📧 🔄 ➕ ➕ 🗑️ 🌑 🗨️ ⬆️ ⬆️ ▶️ Cost: 3ms <

✓ 🔍 name  
varchar

	1	John
--	---	------

**Q45.**

Table: Student

Column Name	Type
student_id	Int
student_name	Varchar
Gender	Varchar
dept_id	Int

student\_id is the primary key column for this table.

dept\_id is a foreign key to dept\_id in the Department tables.

Each row of this table indicates the name of a student, their gender, and the id of their department.

Table: Department

Column Name	Type
dept_id	Int
dept_name	Varchar

dept\_id is the primary key column for this table.

Each row of this table contains the id and the name of a department.

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.

The query result format is in the following example.

Input: Student

table:

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

Department table:

dept_id	dept_name
1	Engineering
2	Science
3	Law

Output:

dept_name	student_number
Engineering	2
Science	1
Law	0

**Q46.**

Table: Customer

Column Name	Type
customer_id	int
product_key	int

There is no primary key for this table. It may contain duplicates.  
product\_key is a foreign key to the Product table.

Table: Product

Column Name	Type
product_key	int

product\_key is the primary key column for this table.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Customer table:

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

Product table:

product_key
5
6

Output:

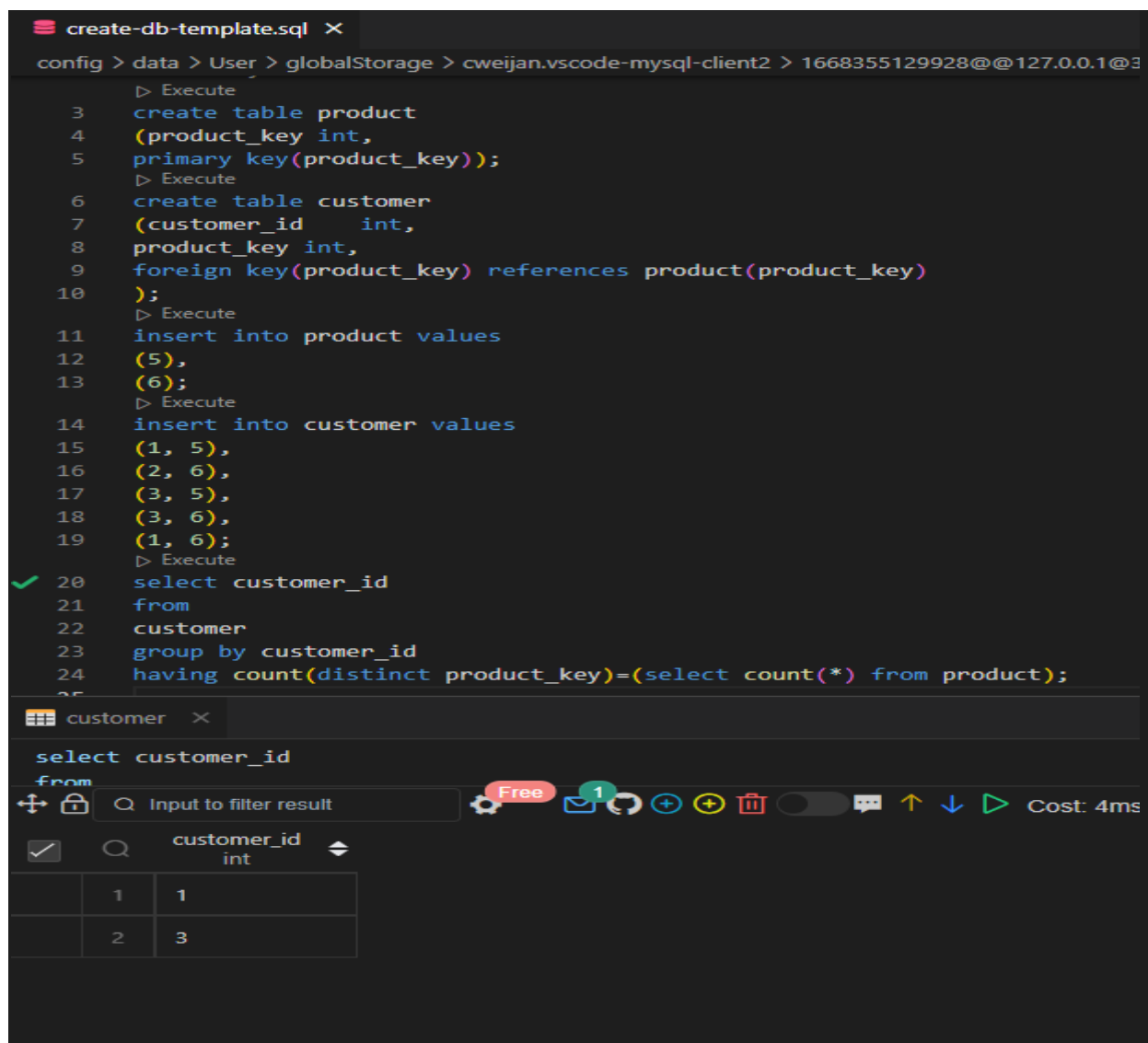
customer_id
1
3

Explanation:

The customers who bought all the products (5 and 6) are customers with IDs 1 and 3.

Solution:

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



The screenshot shows a MySQL client window with a SQL script being executed. The script creates two tables, 'product' and 'customer', and inserts data into them. The 'product' table has two rows with 'product\_key' values 5 and 6. The 'customer' table has three rows with 'customer\_id' values 1, 2, and 3, and 'product\_key' values 5, 6, and 5 respectively. The script then executes a query to find customers who bought all products, which returns two rows: customer\_id 1 and 3.

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

customer_id	product_key
1	5
2	6
3	5
3	6

customer_id	product_key
1	5
2	6
3	5
3	6

Q47.

Table: Project

Column Name	Type
project_id	Int
employee_id	Int

(project\_id, employee\_id) is the primary key of this table.

employee\_id is a foreign key to the Employee table.

Each row of this table indicates that the employee with employee\_id is working on the project with project\_id.

Table: Employee

Column Name	Type
employee_id	Int
Name	Varchar
experience_years	Int

employee\_id is the primary key of this table.

Each row of this table contains information about one employee.

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

Return the result table in any order.

The query result format is in the following example.

Input: Project

table:

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

Employee table:

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



Output:

project_id	employee_id
1	1
1	3
2	1

Explanation:

Both employees with id 1 and 3 have the most experience among the employees of the first project.  
For the second project, the employee with id 1 has the most experience.

Solution:

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

The screenshot shows a MySQL client interface with a SQL editor and a results pane. The SQL editor contains the following code:

```
3 create table employee
4 (employee_id int,
5 name varchar(10),
6 experience_years int,
7 primary key(employee_id)
8 );
9
10 create table project
11 (project_id int,
12 employee_id int,
13 primary key(project_id, employee_id),
14 foreign key(employee_id) references employee(employee_id)
15 );
16
17 insert into employee values
18 (1, 'Khaled', 3),
19 (2, 'Ali', 2),
20 (3, 'John', 3),
21 (4, 'Doe', 2);
22
23 insert into project values
24 (1, 1),
25 (1, 2),
26 (1, 3),
27 (2, 1),
28 (2, 4);
29
30 select t.project_id, t.employee_id
31 from
32 (select p.project_id, e.employee_id, dense_rank() over(partition by p.project_id order by e.experience_years desc) as r
33 from
34 project p
35 left join
36 employee e
37 on p.employee_id = e.employee_id) t
38 where r = 1
39 order by t.project_id;
```

The results pane shows the output of the query:

project_id	employee_id
1	1
2	1
3	2

**Q48.**

Table: Books

Column Name	Type
book_id	Int
Name	Varchar
available_from	Date

book\_id is the primary key of this table.

Table: Orders

Column Name	Type
order_id	Int
book_id	Int
quantity	Int
dispatch_date	date

order\_id is the primary key of this table.

book\_id is a foreign key to the Books table.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Books table:

book_id	name	available_from
1	"Kalila And Demna"	2010-01-01
2	"28 Letters"	2012-05-12
3	"The Hobbit"	2019-06-10
4	"13 Reasons Why"	2019-06-01
5	"The Hunger Games"	2008-09-21

Orders table:

order_id	book_id	quantity	dispatch_date
1	1	2	2018-07-26
2	1	1	2018-11-05
3	3	8	2019-06-11
4	4	6	2019-06-05
5	4	5	2019-06-20
6	5	9	2009-02-02
7	5	8	2010-04-13

Result table:

book_id	Name
1	"Kalila And Demna"
2	"28 Letters"
5	"The Hunger Games"

Solution:

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

create-db-template.sql X

```

config > data > User > globalStorage > oweiian.vscod-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
  > Execute
3  create table Books
4  (book_id int primary key,
5   name varchar(40),
6   available_from date);
  > Execute
7  create table Orders
8  (order_id int primary key,
9   book_id int,
10  quantity int,
11  dispatch_date date,
12  foreign key(book_id) references Books(book_id));
  > Execute
13 insert into Books values
14 (1, "Kalila And Demna", '2010-01-01'),
15 (2, "28 Letters", '2012-05-12'),
16 (3, "The Hobbit", '2019-06-10'),
17 (4, "13 Reasons Why", '2019-06-01'),
18 (5, "The Hunger Games", '2008-09-21');
  > Execute
19 insert into Orders values
20 (1, 1, 2, '2018-07-26'),
21 (2, 1, 1, '2018-11-05'),
22 (3, 3, 8, '2019-06-11'),
23 (4, 4, 6, '2019-06-05'),
24 (5, 4, 5, '2019-06-20'),
25 (6, 5, 9, '2009-02-02'),
26 (7, 5, 8, '2010-04-13');
  > Execute
27 select t1.book_id, t1.name
28 from
29 (
30  (select book_id, name from Books where
31   available_from < '2019-05-23') t1
32  left join
33   (select book_id, sum(quantity) as quantity
34    from Orders
35     where dispatch_date > '2018-06-23' and dispatch_date<= '2019-06-23'
36     group by book_id
37     having quantity < 10) t2
38   on t1.book_id = t2.book_id
39  );
40

```

Orders X

```

select t1.book_id, t1.name
from

```

Free 1

Input to filter result

Cost 3ms < 1 > Total 3

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

Q49.

Table: Enrollments

Column Name	Type
student_id	Int
course_id	Int
Grade	Int

(student\_id, course\_id) is the primary key of this table.

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

Input: Enrollments

table:

student_id	course_id	Grade
2	2	95
2	3	95
1	1	90
1	2	99
3	1	80
3	2	75
3	3	82

Output:

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

Solution:

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



**Q50.**

Table: Teams

Column Name	Type
team_id	Int
team_name	Varchar

team\_id is the primary key of this table.

Each row of this table represents a single football team.

Table: Matches

Column Name	Type
match_id	Int
host_team	Int
guest_team	Int
host_goals	Int
guest_goals	Int

match\_id is the primary key of this table.

Each row is a record of a finished match between two different teams.

Teams host\_team and guest\_team are represented by their IDs in the Teams table (team\_id), and they scored host\_goals and guest\_goals goals, respectively.

The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player\_id wins.

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

The query result format is in the following example.

Input: Players

table:

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

Matches table:

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

Output:

group_id	player_id
1	15
2	35
3	40

Solution:


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



```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
```

Players X

✚ 🔒 🔍 Input to filter result ⚙️ Free 1 📧 🔄 ➕ ➕ 🗑️ ⏸️ 💬 ⬆️ ⬇️ ▶️ Cost: 12ms < 1 > Total 3

<input checked="" type="checkbox"/>		group_id int	player_id int
	1	1	15
	2	2	35
	3	3	40