

City-Dataset:<https://docs.google.com/spreadsheets/d/1dk9kRwcMxj5USuJqxtlTD05S-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;
```

Q2. Query the NAME field for all American cities in the CITY table with populations larger than 1200000.

The CountryCode for America is USA.

The CITY table is described as follows:

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 > 1200000;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** The query `select name from city where countrycode = 'USA' and population > 1200000;` is entered.
- Results:** The results show five rows of city names: El Paso, Scottsdale, Corona, Concord, and Cedar Rapids.
- Status Bar:** Shows the cost of 4ms and a total of 5 rows.
- Left Panel:** Shows the database structure with tables like CITY, COUNTRY, and STATE.
- Bottom Status:** Shows the current connection status and session information.

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;
```

The screenshot shows a MySQL Workbench interface. On the left is a results grid for a query of the CITY table, displaying columns: id, name, countrycode, and district. The data includes rows for cities like Rotterdam, Zaandstad, Porto Alegre, etc. On the right is a SQL editor window containing a large block of SQL code. The code consists of several INSERT statements into the CITY table, listing numerous cities with their respective IDs, names, country codes (e.g., NLD, BRA, BGR), and districts. The SQL editor has syntax highlighting and line numbers.

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-dl
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'), (19,'Zaanstad','NLD',
11  (547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDI','Bujumbura',300000), (554
12  (762,'Bahir Dar','ETH','Amhara',96148), (796,'Baguio','PHL','CAR',252386), (896,'Malu
13  (990,'Wuru','IDN','East Java',124308), (1155,'Latun','IND','Maharashtra',197488), (12
14  (1279,'Neyveli','IND','Tamil Nadu',118886), (1293,'Pallavaram','IND','Tamil Nadu',111
15  (1588,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Roma
16  (1681,'Omata','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yanaguchi',107078), (1
17  (1900,'Changchun','CHN','Jilin',2812800), (1913,'Lanzhou','CHN','Gansu',1565800), (19
18  (2153,'Xianyang','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), (2585,'Taza','MAR','Taza-Al Hoceima-Tau','
21  (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600
22  (2972,'Malabo','GNQ','Bioko',48088), (3073,'Essen','DEU','Nordrhein-Westfalen',599515
23  (3353,'Sousse','TUN','Sousse',145980), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',24
24  (3878,'Scottsdale','USA','Arizona',282705), (3965,'Corona','USA','California',124966)
25  (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',995
26  > Execute
27  select * from city
28
29
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 the MySQL Workbench interface. At the top, there's a file tree with 'create-db-template.sql' selected. Below it is a code editor containing SQL statements to create a 'city' table and insert data into it. The table has columns: name (varchar(17)), countrycode (varchar(3)), district (varchar(28)), and population (int). The insert statement includes 25 rows of data. After the table definition, there are two 'Execute' buttons. The second button is highlighted. Below the code editor is a results pane titled 'city'. It contains a single row of data: id=1, name='Sayama', countrycode='JPN', district='Saitama', and population=162472. The results pane also shows a 'Cost: 11ms' and a 'Total 1' message.

```
5  name varchar(17),
6  countrycode varchar(3),
7  district varchar(28),
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','Amhara',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',118000), (1293,'Pallavaram','IND','Tamil Nadu',111866), (1358,'Dehri','IND','Bihar',9
15 (1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1528,'Cesena','ITA','Emilia-Romagna',89852), (1613,'Neyagawa','JPN',
16 (1681,'Onuta','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107078), (1793,'Novi Sad','YUG','Vojvodina',17
17 (1900,'Changchun','CHN','Jilin',2812000), (1913,'Lanzhou','CHN','Gansu',1565800), (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','Kyongsan
20 (2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX','Veraci
21 (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600), (2706,'Tete','MOZ','Tete',101984
22 (2972,'Malabo','GNQ','Bioko',48000), (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;
```

city

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

	id	name	countrycode	district	population
<input checked="" type="checkbox"/>	1	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';
```

The screenshot shows the MySQL Workbench interface. At the top, there's a tab bar with 'create-db-template.sql' selected. Below it, the 'config' section shows the SQL code for creating the 'city' table and inserting data. A note indicates that the data is from Wikipedia. The main pane displays the results of the 'select * from city where countrycode = 'JPN'' query, showing five rows of Japanese cities.

```

3  create table city
4  (
5      id int,
6      name varchar(17),
7      countrycode varchar(3),
8      district varchar(20),
9      population int
9 );
D Execute
10 insert into city Values (6,'Rotterdam','NLD','Zuid-Holland',593321), (19,'Zaanstad','NLD','Noord-Holland',135621), (214,'Porto Alegre','BRA','Ri
11 (547,'Dobric','BGR','Varna',100399), (552,'Bujumbura','BDT','Bujumbura',300000), (554,'Santiago de Chile','CHL','Santiago',4703954), (626,'al-Mi
12 (762,'Bahir Dar','ETH','Amhara',96140), (796,'Baguio','PHL','CAR',252386), (896,'Malungon','PHL','Southern Mindanao',93232), (904,'Banjul','GMB
13 (998,'Waru','IDN','East Java',124300), (1155,'Latun','IND','Maharashtra',197408), (1222,'Tenali','IND','Andhra Pradesh',143726), (1235,'Tirunelv
14 (1279,'Neyveli','IND','Tamil Nadu',118880), (1293,'Pallavaram','IND','Tamil Nadu',111866), (1358,'Dehri','IND','Bihar',94526), (1383,'Fabriz','I
15 (1508,'Bolzano','ITA','Trentino-Alto Adige',97232), (1520,'Cesena','ITA','Emilia-Romagna',89852), (1613,'Neyagawa','JPN','Osaka',257315), (1630,
16 (1681,'Omura','JPN','Fukuoka',142889), (1739,'Tokuyama','JPN','Yamaguchi',107078), (1793,'Novi Sad','YUG','Vojvodina',179626), (1857,'Kelowna','
17 (1900,'Changchun','CHN','Jilin',2812000), (1913,'Lanzhou','CHN','Gansu',1565800), (1947,'Changzhou','CHN','Jiangsu',530000), (2070,'Dezhou','CHN
18 (2153,'Xianning','CHN','Hubei',136811), (2192,'Lhasa','CHN','Tibet',120000), (2193,'Lianyuan','CHN','Hunan',118858), (2227,'Xingcheng','CHN','Li
19 (2386,'Yongju','KOR','Kyongsangbuk',131097), (2387,'Chinhae','KOR','Kyongsangnam',125997), (2388,'Sangju','KOR','Kyongsangbuk',124116), (2406,'H
20 (2462,'Lilongwe','MWI','Lilongwe',435964), (2505,'Taza','MAR','Taza-Al Hoceima-Taou',92700), (2555,'Xalapa','MEX','Veracruz',390058), (2602,'Oco
21 (2670,'San Pedro Cholula','MEX','Puebla',99734), (2689,'Palikir','FSM','Pohnpei',8600), (2706,'Tete','MOZ','Tete',101984), (2716,'Sittwe (Akyab)
22 (2972,'Malabou','GNQ','Bioko',40000), (3073,'Essen','DEU','Nordrhein-Westfalen',99515), (3169,'Apia','WSM','Upolu',35908), (3198,'Dakar','SEN','
23 (3353,'Sousse','TUN','Sousse',145900), (3377,'Kahramanmaraş','TUR','Kahramanmaraş',245772), (3430,'Odesa','UKR','Odesa',1011000), (3581,'St Pete
24 (3878,'Scottsdale','USA','Arizona',202705), (3965,'Corona','USA','California',124966), (3973,'Concord','USA','California',121780), (3977,'Cedar
25 (4058,'Boulder','USA','Colorado',91238), (4061,'Fall River','USA','Massachusetts',90555);
D Execute
26 select * from city where countrycode = 'JPN';
27
28
```

city

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

	id	name	countrycode	district	population
1	1613	Neyagawa	JPN	Osaka	257315
2	1630	Ageo	JPN	Saitama	209442
3	1661	Sayama	JPN	Saitama	162472
4	1681	Omura	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';
```

station-table:<https://docs.google.com/spreadsheets/d/1sHPhE7walOD5mL7ppFNqybyoOJY3E51NQcWYzhp2UH4/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

Input to filter result

Cost: 7ms Total 499

#	city	state
1	Kissee Mills	MO
2	Loma Mar	CA
3	Sandy Hook	CT
4	Tipton	IN
5	Arlington	CO
6	Turner	AR
7	Slidell	LA
8	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	Morenci	AZ
33	South El Monte	CA

create-db-template.sql

```
config > data > User > globalStorage > cweijan.vscode-mysql-client
```

```
(82, 'Many', 'LA', 36, 94),  
(644, 'Seward', 'AK', 120, 35),  
(391, 'Berryton', 'KS', 60, 139),  
(696, 'Chilhowee', 'MO', 79, 49),  
(905, 'Newark', 'IL', 72, 129),  
(81, 'Cowgill', 'MO', 136, 27),  
(31, 'Novinger', 'MO', 108, 111),  
(385, 'Goodman', 'MS', 100, 117),  
(84, 'Cohait', 'CT', 87, 26),  
(754, 'South Haven', 'MI', 144, 52),  
(144, 'Eskridge', 'KS', 107, 63),  
(305, 'Bennington', 'KS', 93, 83),  
(226, 'Decatur', 'MS', 71, 117),  
(224, 'West Hyannisport', 'MA', 58, 96),  
(456, 'Ozark', 'FL', 148, 126),  
(623, 'Jackson', 'AL', 111, 67),  
(548, 'Lapeer', 'MI', 128, 114),  
(819, 'Peaks Island', 'ME', 59, 110),  
(243, 'Hazlehurst', 'MS', 49, 108),  
(457, 'Chester', 'CA', 69, 123),  
(871, 'Clarkston', 'MI', 93, 88),  
(478, 'Healdsburg', 'CA', 111, 54),  
(445, 'Wadsworth', 'CA', 69, 71),  
(698, 'Ravenden Springs', 'AR', 67, 108),  
(62, 'Monroe', 'AR', 131, 150),  
(365, 'Payson', 'IL', 81, 92),  
(922, 'Kelli', 'IL', 78, 58),  
(838, 'Strasburg', 'CO', 89, 47),  
(286, 'Five Points', 'AL', 45, 122),  
(471, 'Montgomery City', 'IL', 83, 76),  
(928, 'Callings', 'AL', 144, 52),  
(746, 'Orange City', 'IA', 93, 162),  
(892, 'Effingham', 'KS', 132, 97),  
(193, 'Corcoran', 'CA', 81, 139),  
(225, 'Garden City', 'IA', 54, 119),  
(476, 'Alton', 'MO', 79, 112),  
(478, 'Glenway', 'AR', 118, 135),  
(479, 'Woodboro', 'MD', 76, 141),  
(783, 'Strawn', 'IL', 29, 51),  
(875, 'Dent', 'MN', 70, 136),  
(278, 'Shingletown', 'CA', 61, 102),  
(378, 'Clio', 'IA', 46, 115),  
(184, 'Yalaha', 'FL', 120, 119),  
(485, 'Leakesville', 'MS', 187, 72),  
(486, 'Locust Tupelo', 'MS', 188, 93),  
(487, 'Shasta', 'CA', 98, 155),  
(448, 'Canton', 'MN', 123, 151),  
(751, 'Agency', 'MO', 59, 95),  
(29, 'South Carrollton', 'KY', 57, 116),  
(718, 'Taft', 'CA', 107, 146),  
(213, 'Calpine', 'CA', 46, 43),  
(493, 'Wardell', 'AL', 95, 62),  
(988, 'Bullhead City', 'AZ', 94, 38),  
(845, 'Tina', 'MO', 131, 28),  
(685, 'Anthony', 'KS', 45, 161),  
(731, 'Emmett', 'ID', 57, 31),  
(311, 'South Haven', 'MN', 38, 87),  
(866, 'Maverhill', 'IA', 61, 109),  
(598, 'Middleboro', 'MA', 188, 149),  
(511, 'Piney Creek', 'GA', 105, 92),  
(889, 'Lenia', 'LA', 78, 129),  
(654, 'Lee', 'IL', 27, 51),  
(844, 'Freeport', 'MD', 113, 58),  
(446, 'Mid Florida', 'FL', 110, 50),  
(249, 'Acme', 'LA', 73, 67),  
(376, 'Gorham', 'KS', 111, 64),  
(508, 'Bass Harbor', 'ME', 137, 61),  
(455, 'Orange', '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 MySQL client interface with two panes. The left pane displays the results of the query `select distinct city from station where id%2 = 0;`. The results are as follows:

city
Kissee Mills
Loma Mar
Tipton
Glencoe
Chignik Lagoon
Albany
Manchester
Cahone
Bowdon
Watkins
Millville
Aguanga
Morenci
Mccomb
Gustine
Delano
Roy
Pattensburg
Centertown
Norvell
Raymondville
West Hills
Wickliffe
Forest Lakes
Little Rock
Hampden
Pine City
Prince Frederick
Yazoo City
Jolon
Childs
Shreveport
Forest
Sizerock

The right pane shows the source code of `create-db-template.sql`, which contains numerous entries for cities and their coordinates. The code ends with the command `select distinct city from station where id%2 = 0;`.

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', 'Bengalaru', there are 2 different city names: 'New York' and 'Bengalaru'. 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 two tabs in MySQL Workbench: 'create-db-template.sql' and 'station'.

In the 'create-db-template.sql' tab, the code is as follows:

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
455  (694,'Uzona','FL',144,140),
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  (478,'Healdsburg','CA',111,54),
463  (785,'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,'Woodsboro','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  (184,'Yalahaa','FL',128,119),
484  (460,'Leakesville','MS',107,72),
485  (884,'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  (998,'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',38,87),
498  (866,'Haverhill','IA',61,109),
499  (598,'Middleboro','MA',188,149),
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',110,58),
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 (count(city) - count(distinct city)) as 'CityCount-DistCityCount' from station;
510
511
```

In the 'station' tab, the results of the query are shown:

```
select (count(city) - count(distinct city)) as 'CityCount-DistCityCount' from station
+-----+
| 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);
```

```

create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
468    (457, 'Chester', 'CA', 69, 123),
469    (871, 'Clarkston', 'MI', 93, 80),
470    (470, 'Healdsburg', 'CA', 111, 54),
471    (705, 'Hotchkiss', 'CO', 69, 71),
472    (690, 'Ravenden Springs', 'AR', 67, 108),
473    (62, 'Monroe', 'AR', 131, 158),
474    (365, 'Payson', 'IL', 81, 92),
475    (922, 'Kell', 'IL', 78, 58),
476    (838, 'Strasburg', 'CO', 89, 47),
477    (286, 'Five Points', 'AL', 45, 122),
478    (968, 'Norris City', 'IL', 53, 76),
479    (928, 'Coaling', 'AL', 144, 52),
480    (746, 'Orange City', 'IA', 93, 162),
481    (892, 'Effingham', 'KS', 132, 97),
482    (193, 'Corcoran', 'CA', 81, 139),
483    (225, 'Garden City', 'IA', 54, 119),
484    (573, 'Alton', 'MO', 79, 112),
485    (838, 'Greenway', 'AR', 119, 35),
486    (241, 'Woodsboro', 'MD', 76, 141),
487    (783, 'Strawn', 'IL', 29, 51),
488    (675, 'Dent', 'MN', 78, 136),
489    (270, 'Shingletown', 'CA', 61, 102),
490    (378, 'Clio', 'IA', 46, 115),
491    (184, 'Yalaha', 'FL', 120, 119),
492    (468, 'Leakesville', 'MS', 107, 72),
493    (804, 'Fort Lupton', 'CO', 38, 93),
494    (53, 'Shasta', 'CA', 99, 155),
495    (448, 'Canton', 'MN', 123, 151),
496    (751, 'Agency', 'MO', 59, 95),
497    (29, 'South Carrollton', 'KY', 57, 116),
498    (718, 'Taft', 'CA', 107, 146),
499    (213, 'Calpine', 'CA', 46, 43),
500    (624, 'Knobel', 'AR', 95, 62),
501    (908, 'Bullhead City', 'AZ', 94, 38),
502    (845, 'Tina', 'MO', 131, 28),
503    (685, 'Anthony', 'KS', 45, 161),
504    (731, 'Emmett', 'ID', 57, 31),
505    (311, 'South Haven', 'MN', 30, 87),
506    (866, 'Maverhill', 'IA', 61, 109),
507    (598, 'Middleboro', 'MA', 108, 149),
508    (541, 'Siloam', 'GA', 105, 92),
509    (889, 'Lena', 'LA', 78, 129),
510    (654, 'Lee', 'IL', 27, 51),
511    (841, 'Freeport', 'MI', 113, 50),
512    (446, 'Mid Florida', 'FL', 110, 58),
513    (249, 'Acme', 'LA', 73, 67),
514    (376, 'Gorham', 'KS', 111, 64),
515    (136, 'Bass Harbor', 'ME', 137, 61),
516    (455, 'Granger', 'IA', 33, 102);
> Execute
509  (select city, length(city) as length from station order by length(city) asc,city asc limit 1)
510  union
511  (select city, length(city) as length from station order by length(city) desc,city asc limit 1);
512
513
514
station ✘
(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)
Input to filter result: Free 1 ↻ + + + Cost: 4ms < 1 2 > Total 2


|   | city                  | length |
|---|-----------------------|--------|
|   | varchar               | bigint |
| 1 | Amo                   | 3      |
| 2 | Marine On Saint Croix | 21     |


```

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');
```

The screenshot shows a code editor with two tabs. The left tab contains a MySQL query to select distinct city names from the STATION table where the first character of the city name is a vowel ('a', 'e', 'i', 'o', or 'u'). The right tab shows the results of this query, listing 65 cities starting with vowels.

```

station ×
SELECT DISTINCT city FROM station WHERE
    LEFT(city,1) IN ('a','e','i','o','u')
Input to filter result: ↵
Cost: 10ms
1 > Total 65
city
varchar
1 Arlington
2 Albany
3 Upperco
4 Aguanga
5 Odin
6 East China
7 Algonac
8 Onaway
9 Irvington
10 Arrowsmith
11 Udall
12 Oakfield
13 Elkton
14 East Irvine
15 Amo
16 Alanson
17 Elele
18 Auburn
19 Oconee
20 Amazonia
21 Aliso Viejo
22 Andersonville
23 Eros
24 Arkadelphia
25 Erline
26 Edgewater
27 East Haddam
28 Eastlake
29 Addison
30 Everton
31 Eustis
32 Arispe
33 Union Star

```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > ↵
440 (450, 'Keweenaw', 'MI', 54, 124),
441 (82, 'Many', 'LA', 36, 94),
442 (644, 'Seward', 'AK', 120, 35),
443 (391, 'Berryton', 'KS', 66, 139),
444 (696, 'Chilhowee', 'MO', 79, 49),
445 (905, 'Newark', 'IL', 72, 129),
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', 187, 63),
452 (385, '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, 118),
459 (243, 'Hazlehurst', 'MS', 49, 108),
460 (457, 'Chester', 'CA', 69, 123),
461 (871, 'Clarkston', 'MI', 93, 88),
462 (478, 'Healdsburg', 'CA', 111, 54),
463 (785, 'Hotchkiss', 'CO', 69, 71),
464 (698, 'Ravenden Springs', 'AR', 67, 108),
465 (62, 'Monroe', 'AR', 131, 150),
466 (365, 'Payson', 'IL', 81, 92),
467 (922, 'Kell', 'IL', 78, 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, 'Woodsboro', '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 (184, 'Yalaha', 'FL', 120, 119),
484 (460, 'Leakesville', 'MS', 187, 72),
485 (884, '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', 187, 146),
491 (213, 'Calpine', 'CA', 46, 43),
492 (624, 'Knobel', 'AR', 95, 62),
493 (908, 'Bullhead City', 'AZ', 94, 38),
494 (845, 'Tlma', 'MO', 131, 28),
495 (685, 'Anthony', 'KS', 45, 161),
496 (731, 'Emmett', 'ID', 57, 31),
497 (311, 'South Haven', 'MN', 38, 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, 58),
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) in ('a','e','i','o','u');
510

```

Q12. Query the list of CITY names ending with vowels (a, e, i, o, 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 right(city,1) in ('a','e','i','o','u');
```

The screenshot shows a MySQL client interface. On the left, a results table displays city names from the STATION table where the rightmost character of the city name is in ('a', 'e', 'i', 'o', 'u'). The table has columns for ID (auto-increment), CITY (VARCHAR(21)), and STATE (VARCHAR(2)). The results list cities like Glencoe, Chelsea, Pelahatchie, Dorrance, Cahone, Upperco, Waipahu, Millville, Aguanga, Morenci, South El Monte, Gustine, Delano, Westphalia, Saint Elmo, Raymondville, Barrigada, Hesperia, Wickliffe, Milledgeville, East China, Gretna, Zionsville, Rio Oso, Samantha, Bentonville, Grosse Pointe, Robertsdale, Dale, Tennessee, Chokio, Bertha, and Regina. On the right, a code editor shows a SQL script named 'create-db-template.sql' with numerous commented-out lines (444 to 588) and a final executable command:

```

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	VARCHAR(21)
STATE	VARCHAR(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');
```

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

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-d
445   (985,'Newark', 'IL',172,120),
446   (81,'Cedartown', 'MO',136,27),
447   (31,'Novinger', 'MO',198,111),
448   (299,'Goodman', 'MS',181,117),
449   (84,'Cobalt', 'CT',87,26),
450   (754,'South Haven', 'MI',144,52),
451   (144,'Eskridge', 'KS',187,63),
452   (385,'Bennington', 'KS',93,83),
453   (226,'Decatur', 'MS',71,117),
454   (224,'West Hyannisport', 'MA',58,96),
455   (693,'Ozona', 'FL',140,128),
456   (623,'Jackson', 'MI',111,59),
457   (543,'Lapeer', 'MI',128,114),
458   (819,'Peaks Island', 'ME',59,118),
459   (243,'Hazelhurst', 'MS',49,108),
460   (457,'Chester', 'CA',69,123),
461   (871,'Clarkston', 'MI',93,80),
462   (478,'Meadlsburg', 'CA',111,54),
463   (785,'Hotchkiss', 'CO',69,71),
464   (698,'Ravenden Springs', 'AR',67,108),
465   (62,'Monroe', 'AR',131,158),
466   (365,'Payson', 'IL',81,92),
467   (932,'Kelli', 'IL',78,58),
468   (831,'Waukesburg', 'CO',109,47),
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   (838,'Greenway', 'AR',119,35),
478   (248,'Woodsboro', 'MD',106,141),
479   (733,'Vtreaton', 'IL',59,51),
480   (675,'Denton', 'MN',70,136),
481   (279,'Shingletown', 'CA',61,182),
482   (378,'Clio', 'IA',46,115),
483   (184,'Valaha', 'FL',120,119),
484   (460,'Leakesville', 'MS',187,72),
485   (884,'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   (40,'Taylors', 'CA',107,13),
491   (213,'Calpine', 'CA',105,43),
492   (624,'Korobel', 'AR',95,62),
493   (988,'Bullhead City', 'AZ',94,38),
494   (845,'Tina', 'MO',131,28),
495   (685,'Anthony', 'KS',45,161),
496   (731,'Emmett', 'ID',57,31),
497   (311,'South Haven', 'MN',38,87),
498   (866,'Havenhill', 'IA',61,109),
499   (598,'Middleboro', 'MA',188,149),
500   (541,'Siloam', 'GA',105,92),
501   (889,'Lena', 'IL',78,129),
502   (62,'Lee', 'IL',21,11),
503   (841,'Pleasant', 'MI',113,58),
504   (446,'Mid Florida', 'FL',110,58),
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
510
511
512
513
514
c-e
```

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');
```

```
station
Cost: 12ms
1 2 3 4 > Total 336
city      varchar
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 Hoskinton
24 Talbert
25 Mccomb
26 Kirk
27 Carlock
28 Seward
29 Roy
30 Pattonsburg
31 Centertown
32 Norvell
33 Beaver Island
```

```
create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template
445  (905,'Newark','IL',72,129),
446  (81,'Cowgill','MO',136,27),
447  (77,'Lindbergh','MO',136,11),
448  (299,'Gooding','ME',181,117),
449  (84,'Cobalt','CT',87,26),
450  (754,'South Haven','MI',144,52),
451  (144,'Eskridge','KS',197,63),
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  (453,'Jacksboro','AL',128,120),
457  (543,'Upperville','MI',128,114),
458  (819,'Peaks Island','ME',59,118),
459  (243,'Hazlehurst','MS',49,108),
460  (457,'Chester','CA',69,123),
461  (871,'Clarkston','MI',93,80),
462  (476,'Healdsburg','CA',111,54),
463  (705,'Hotchkiss','CO',69,71),
464  (696,'Ravenden Springs','AR',67,108),
465  (459,'Concordia','KS',132,98),
466  (365,'Prairie City','IL',81,92),
467  (922,'Kelli','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,'Cooling','AL',344,52),
472  (746,'Orange City','IA',93,162),
473  (892,'Effingham','KS',132,97),
474  ('Concordia CA',81,39),
475  (225,'Gardens City','IL',54,119),
476  (573,'Altom','MO',79,112),
477  (830,'Greenway','AR',119,35),
478  (241,'Woodbury','MD',76,141),
479  (783,'Strawn','IL',29,51),
480  (675,'Dent','MN',70,136),
481  (276,'Shingletown','CA',61,102),
482  (378,'Clio','IA',46,115),
483  (418,'Valders','WI',120,105),
484  (660,'Leakesville','MS',187,72),
485  (804,'Port 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',187,146),
491  (213,'Calpine','CA',46,43),
492  (629,'Moline','IL',95,111),
493  (688,'Bullhead City','AZ',194,38),
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  (614,'Lena','IL',27,51),
502  (654,'Vandalia','IL',78,59),
503  (841,'Freeport','MI',113,58),
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 right(city,1) not in ('a','e','i','o','u');
```

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

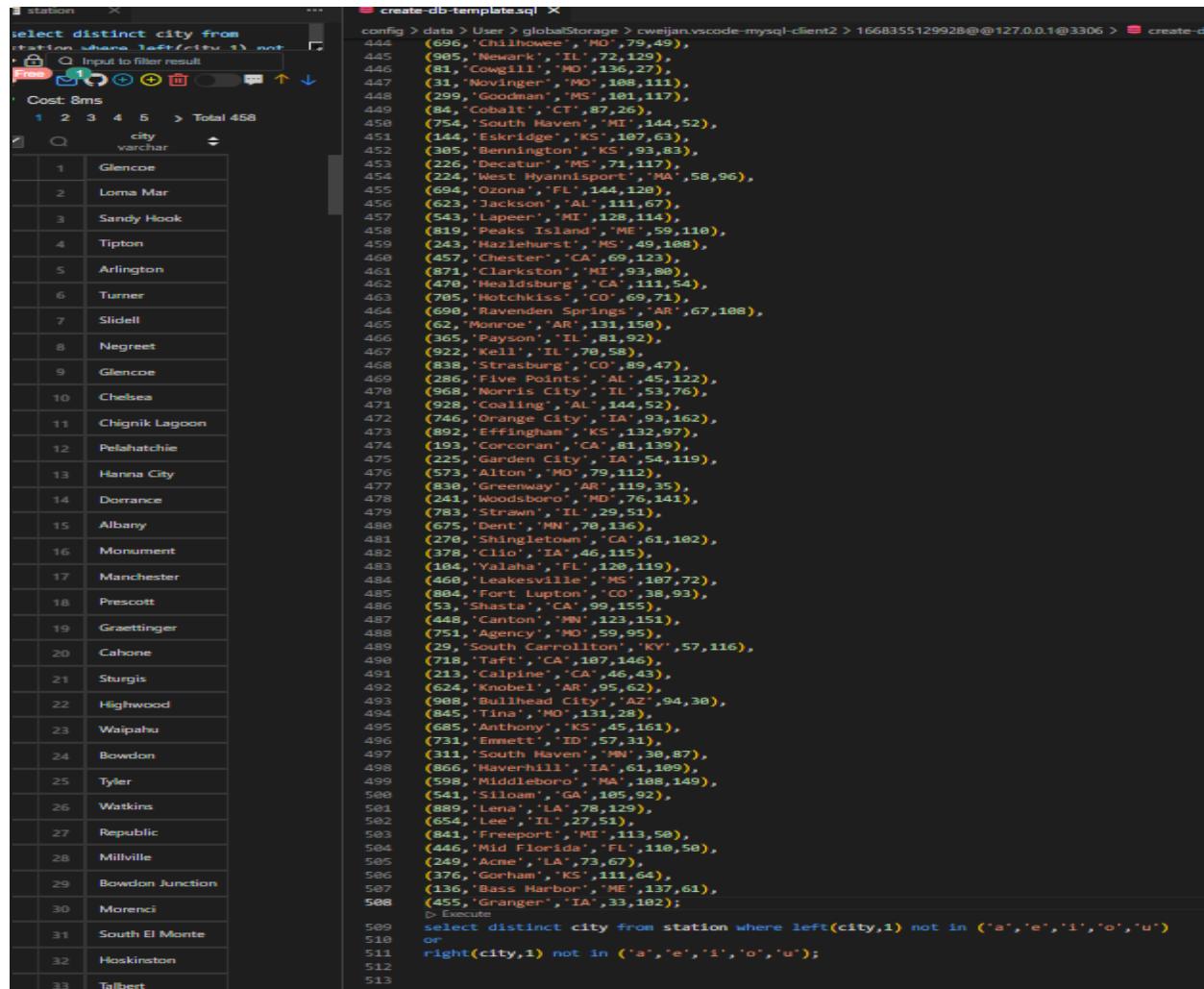
The STATION table is described as follows:

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');
```



The screenshot shows the MySQL Workbench interface. On the left, there is a tree view with 'station' selected. In the center, the 'station' table is displayed with 458 rows. The columns are 'city' (VARCHAR) and 'id' (NUMBER). The first few rows are: 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 Hoskinstion, 33 Talbert.

On the right, the SQL editor contains the following code:

```

-- create-db-template.sql
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template
444   (696,'Chilhowee','MO',79,49),
445   (985,'Newark','IL',72,129),
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','MA',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,'Mazlehurst','MS',49,108),
460   (457,'Chester','CA',69,123),
461   (871,'Clarkston','MI',93,80),
462   (478,'Meadsbury','CA',111,54),
463   (785,'Hotchkiss','CO',69,71),
464   (690,'Ravenden Springs','AR',67,108),
465   (62,'Monroe','AR',131,158),
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   (838,'Greenway','AR',119,35),
478   (241,'Woodsboro','MD',76,141),
479   (783,'Stawn','IL',29,51),
480   (675,'Dent','MN',78,136),
481   (278,'Shingletown','CA',61,182),
482   (378,'Clio','IA',46,115),
483   (184,'Yalaha','FL',120,119),
484   (468,'Leakesville','MS',107,72),
485   (804,'Fox 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','IA',78,129),
502   (654,'Lee','IL',27,51),
503   (841,'Freeport','MI',113,50),
504   (446,'Miami Florida','FL',110,58),
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 X

```
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');
```

Cost: 13ms

	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 X

```
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, 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, 158),
466   (365, 'Payson', 'IL', 81, 92),
467   (922, 'Kell', 'IL', 78, 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, 'Woodsboro', 'MD', 76, 141),
479   (783, 'Strawn', 'IL', 29, 51),
480   (675, 'Dent', 'MN', 78, 136),
481   (270, 'Shingletown', 'CA', 61, 102),
482   (378, 'Clio', 'IA', 46, 115),
483   (184, '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   (988, 'Bullhead City', 'AZ', 94, 38),
494   (845, 'Tina', 'MO', 131, 28),
495   (685, 'Anthony', 'KS', 45, 161),
496   (731, 'Emmett', 'ID', 57, 31),
497   (311, 'South Haven', 'MN', 38, 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, 58),
504   (446, 'Mid Florida', 'FL', 110, 58),
505   (249, 'Acme', 'LA', 73, 67),
506   (376, 'Gorham', 'KS', 111, 64),
507   (136, 'Bass Harbor', 'ME', 137, 61),
508   (455, 'Granger', 'IA', 33, 182),|
```

▷ Execute

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

513

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.sql
  3  create table Product
  4    (product_id int,
  5     product_name  varchar(20),
  6     unit_price  int,
  7     primary key(product_id)
  8   );
  9   /* Execute
10  create table Sales
11    (seller_id  int,
12     product_id  int,
13     buyer_id  int,
14     sale_date  date,
15     quantity  int,
16     price  int,
17     foreign key(product_id) references Product(product_id)
18   );
19   /* Execute
20  insert into Product values
21    (1, 'S8', 1000),
22    (2, 'G4', 800),
23    (3, 'iPhone', 1400);
24   /* Execute
25  insert into Sales values
26    (1, 1, 1, '2019-01-21', 2, 2000),
27    (1, 2, 2, '2019-02-17', 1, 800),
28    (2, 2, 3, '2019-06-02', 1, 800),
29    (3, 3, 4, '2019-05-13', 2, 2800);
30   /* Execute
31  (select p.product_id, p.product_name FROM
32  Product p
33  INNER JOIN
34  Sales s
35  on p.product_id = s.product_id
36  where s.sale_date >= '2019-01-01' and s.sale_date <= '2019-03-31')
37  EXCEPT
38  (select p.product_id, p.product_name FROM
39  Product p
40  INNER JOIN
41  Sales s
42  on p.product_id = s.product_id
43  where s.sale_date < '2019-01-01' OR s.sale_date > '2019-03-31')
44
Users ×
(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')
Cost: 2ms < 1 > Total 1
  Input to filter result Free 1


|  | product_id | product_name |
|--|------------|--------------|
|  | 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;
```

The screenshot shows a MySQL Workbench interface. In the top-left, there's a terminal window titled 'create-db-template.sql' with the following SQL code:

```
2 use test;
3 create table views(
4     | article_id int,
5     author_id    int,
6     viewer_id   int,
7     view_date   date
8 );
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');
17 select distinct author_id as id from views where author_id = viewer_id order by author_id asc;
```

Below the code, a results grid titled 'views' displays the following data:

	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;
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, there is a code editor with the following SQL query:

```
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 insert into delivery values(1, 1, '2019-08-01', '2019-08-02'),
12 (2, 5, '2019-08-02', '2019-08-02'),
13 (3, 1, '2019-08-11', '2019-08-11'),
14 (4, 3, '2019-08-24', '2019-08-26'),
15 (5, 4, '2019-08-21', '2019-08-22'),
16 (6, 2, '2019-08-11', '2019-08-13');
17 select round((select count(*) from delivery where order_date = customer_pref_delivery_date)/count(*)*100,2)
18 as immediate_percentage from delivery;
```

In the bottom-right pane, the results of the query are displayed in a table:

immediate_percentage
33.33

The status bar at the bottom indicates a cost of 24ms and a total of 1 row.

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:

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

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;
```

The screenshot shows a MySQL Workbench interface. The top part is a SQL editor window titled 'create-db-template.sql' with the following code:

```

3   create table ads(
4     ad_id Int,
5     user_id Int,
6     action varchar(10),
7     primary key(ad_id, user_id)
8   );
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');
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

```

The bottom part is a results grid titled 'ads' with the following data:

	ad_id	Ctr
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:

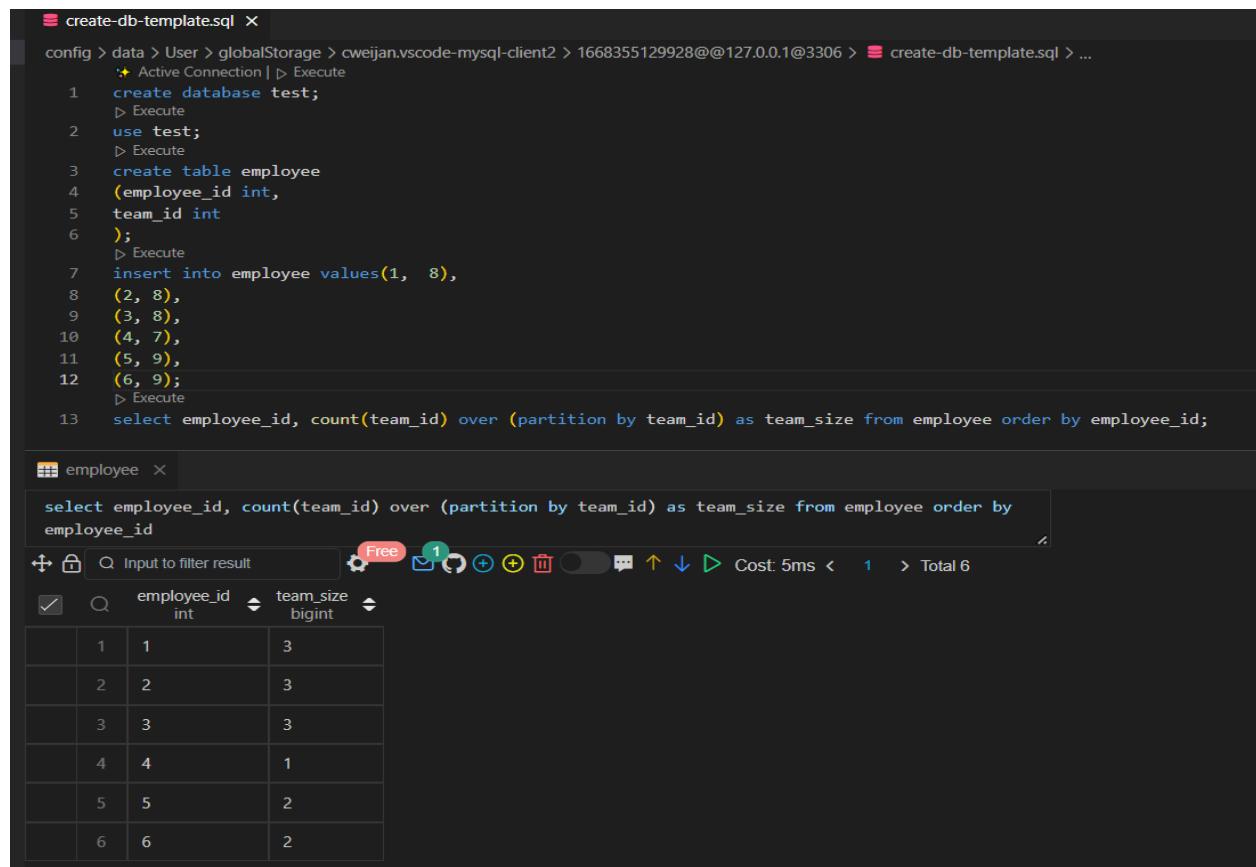
Employees with Id 1,2,3 are part of a team with team_id = 8.

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;
```



```
create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Active Connection | ▶ Execute
1   create database test;
  ▷ Execute
2   use test;
  ▷ Execute
3   create table employee
4     (employee_id int,
5      team_id int
6    );
  ▷ Execute
7   insert into employee values(1,  8),
8     (2, 8),
9     (3, 8),
10    (4, 7),
11    (5, 9),
12    (6, 9);
  ▷ Execute
13  select employee_id, count(team_id) over (partition by team_id) as team_size from employee order by employee_id;

employee ×
select employee_id, count(team_id) over (partition by team_id) as team_size from employee order by
employee_id
Free 1
+-----+-----+-----+
| employee_id | team_size |
+-----+-----+
|         1   |       3   |
|         2   |       3   |
|         3   |       3   |
|         4   |       1   |
|         5   |       2   |
|         6   |       2   |
+-----+-----+
```

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 > cweijan.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

Free 1


|   | country_name | weather_state |
|---|--------------|---------------|
|   | varchar      | 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 ×
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
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;
Input to filter result Free 1
product_id int average_price newdecimal
1 1 6.96
2 2 16.96
Cost: 20ms < 1 > Total 2

```

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 the MySQL Workbench interface. On the left, there's a tree view under 'config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql'. The main area contains the SQL code for creating a table 'activity' and inserting data, followed by a query to find the first login date for each player. Below the code, the results are displayed in a table titled 'activity'.

player_id	first_login
1	2016-03-01
2	2017-06-25
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 two tabs: 'create-db-template.sql' and 'activity'.

In the 'create-db-template.sql' tab, the following SQL code is shown:

```
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.device_id
16 from (select player_id, device_id, row_number() over(partition by player_id order by event_date) as num from activity)t
17 where t.num = 1;
18
```

In the 'activity' tab, the results of the executed query are displayed:

player_id	device_id
1	1
2	2
3	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 ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  ↳ 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

	product_name	unit
1	Leetcode Solutions	130
2	Leetcode Kit	100

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@leetc ode.com
2	Jonathan	jonathanisgreat
3	Annabelle	bella-@leetcod e.com
4	Sally	sally.come@lee tcode.com
5	Marwan	quarz#2020@le etcode.com
6	David	david69@gmail .com
7	Shapiro	.shapo@leetco de.com

Output:

user_id	name	mail
1	Winston	winston@leetc ode.com
3	Annabelle	bella-@leetcod e.com
4	Sally	sally.come@lee tcode.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 ×
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@leetco de.com');
    ▷ Execute
✓ 16  select user_id, name, mail from Users
17  where
18  mail regexp '^[a-zA-Z]+[a-zA-Z0-9_\.\\-]*@leetcode[\\.]com'
19  order by user_id;
20

```

Users

	user_id	name	mail
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 > ...  
10    primary key(order_id)  
11  );  
12  /* Execute  
13  insert into Customers values  
14  (1, 'Winston', 'USA'),  
15  (2, 'Jonathan', 'Peru'),  
16  (3, 'Moustafa', 'Egypt');  
17  /* Execute  
18  insert into Product values  
19  (10, 'LC Phone', 300),  
20  (20, 'LC T-Shirt', 10),  
21  (30, 'LC Book', 45),  
22  (40, 'LC Keychain', 2);  
23  /* Execute  
24  insert into Orders values  
25  (1, 1, 10, '2020-06-10', 1),  
26  (2, 1, 20, '2020-07-01', 1),  
27  (3, 1, 30, '2020-07-08', 2),  
28  (4, 2, 10, '2020-06-15', 2),  
29  (5, 2, 40, '2020-07-01', 10),  
30  (6, 3, 20, '2020-06-24', 2),  
31  (7, 3, 30, '2020-06-25', 2),  
32  (9, 3, 30, '2020-05-08', 3);  
33  /* Execute  
34  select t.customer_id, t.name from  
35  (select c.customer_id, c.name,  
36  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,  
37  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  
38  from  
39  Orders o  
40  left join  
41  Product p  
42  on o.product_id = p.product_id  
43  left join  
44  Customers c  
45  on o.customer_id = c.customer_id  
46  group by c.customer_id) t  
47  where june_spent >= 100 and july_spent >= 100;  
48  
49  
Data  
+-----+-----+  
| customer_id | name |  
+-----+-----+  
| 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

"Aladdin" 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  );
9  > Execute
10 create table Content
11 (content_id int,
12 Title  Varchar(20),
13 Kids_content  Varchar(1),
14 content_type  Varchar(15),
15 primary key(content_id)
16 );
17 > Execute
18 insert into TVProgram values
19 ('2020-06-10 08:00', 1, 'LC-Channel'),
20 ('2020-05-11 12:00', 2, 'LC-Channel'),
21 ('2020-05-12 12:00', 3, 'LC-Channel'),
22 ('2020-05-13 14:00', 4, 'Disney Ch'),
23 ('2020-06-18 14:00', 4, 'Disney Ch'),
24 ('2020-07-15 16:00', 5, 'Disney Ch');
25 > Execute
26 insert into Content values
27 (1, 'Leetcode Movie', 'N', 'Movies'),
28 (2, 'Alg. for Kids', 'Y', 'Series'),
29 (3, 'Database Sols', 'N', 'Series'),
30 (4, 'Aladdin', 'Y', 'Movies'),
31 (5, 'Cinderella', 'Y', 'Movies');
32 > Execute
33 select c.Title from
34 Content c
35 left join
36 TVProgram t
37 on c.content_id = t.content_id
38 where c.Kids_content = 'Y' and c.content_type = 'Movies' and month(t.program_date) = 6 and year(t.program_date) = 2020;
39 > Execute
TVProgram X
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;
Cost: 3ms < 1 > Total 1
Free 1
Input to filter result
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 ×
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);
30 ⌂ 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 ⌂ Execute

```

Data

```

select q.*, coalesce(n.Npv,0) as Npv
From
Input to filter result
Free 1
Cost: 4ms < 1 > Total 7

```

	Id	Year	Npv
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:

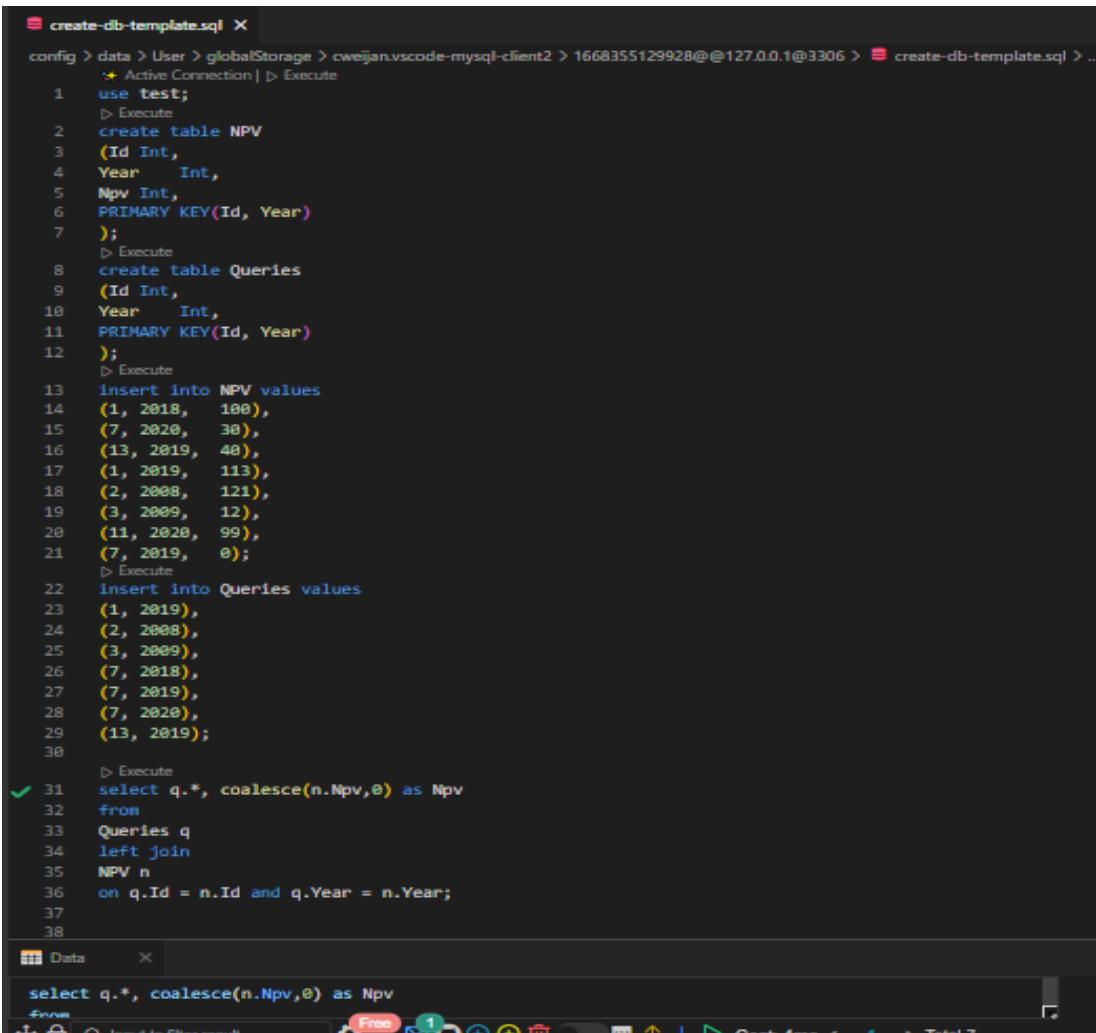
Id	Year	Npv
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

Explanation:

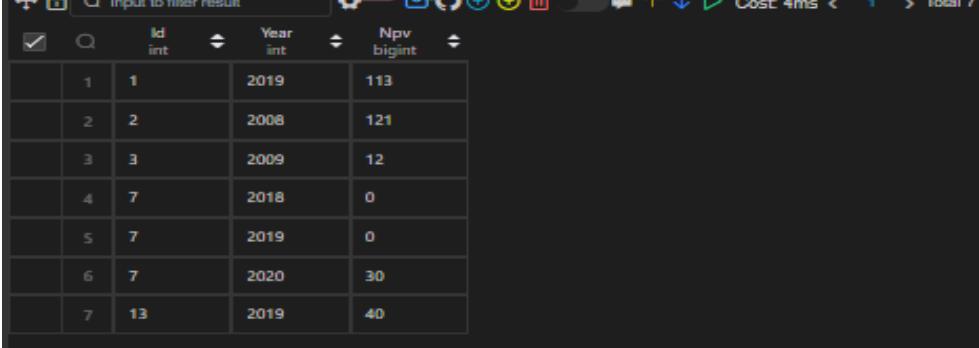
The npv value of (7, 2018) is not present in the NPV table, we consider it 0. The npv values of all other queries can be found in the NPV table.

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 > ...
1  use test;
2  create table NPV
3    (Id Int,
4     Year  Int,
5     Npv  Int,
6     PRIMARY KEY(Id, Year)
7   );
8  create table Queries
9    (Id Int,
10     Year  Int,
11     PRIMARY KEY(Id, Year)
12   );
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);
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);
30
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
```



	Id	Year	Npv
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 terminal window with the following SQL 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;
```

Below the terminal, there is a data viewer window titled "Data". It displays the results 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 > 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

users u

left join

rides r

on u.id = r.user_id

group by u.name

order by travelled_distance desc, u.name;

Input to filter result

	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

Cost: 3ms < 1 > Total 7

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 Workbench interface. At the top, there's a status bar with the current connection details: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql. Below the status bar, the main area displays a SQL script with line numbers. The script creates a 'Orders' table, inserts data into 'Products' and 'Orders' tables, and then runs the same query as the solution. The results are shown in a table at the bottom.

```
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;
```

Product

	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 the MySQL Workbench interface with the following details:

- Query Editor:** The code block above is pasted into the query editor.
- Status Bar:** Shows the connection details: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@@127.0.0.1@3306 > create-db-template.sql > ...
- Execution Results:** The results of the query are displayed in a table titled "Data".
- Table Headers:** The table has two columns: "Results" and "varchar".
- Table Data:**

Results	varchar
1	Daniel
2	Frozen 2
- Performance Metrics:** Below the table, it shows: Cost: 2ms < 1 > Total 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;
```

```

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;

```

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;

	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 x
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  8  create table employeeUNI
  9    (id Int,
10     unique_id  Int,
11     primary key(id, unique_id)
12   );
13   ▷ Execute
14   insert into employees values
15   (1, 'Alice'),
16   (7, 'Bob'),
17   (11, 'Meir'),
18   (90, 'Winston'),
19   (3, 'Jonathan');
20   ▷ Execute
21   insert into employeeUNI values
22   (3, 1),
23   (11, 2),
24   (90, 3);
25   ▷ Execute
26 ✓ 24  select u.unique_id, e.name
27   from
28   employees e
29   left join
30   employeeUNI u
31   on e.id = u.id;
32   |
33
Data x
select u.unique_id, e.name
from
Input to filter result
Free 1
unique_id int
name varchar
1 (NULL) Alice
2 (NULL) Bob
3 1 Jonathan
4 2 Meir
5 3 Winston
```

The screenshot shows a MySQL client interface within VS Code. The top part displays a SQL script named 'create-db-template.sql' with line numbers. The script creates a table 'employeeUNI' with columns 'id' and 'unique_id', and inserts data into 'employees' and 'employeeUNI' tables. A query is run at line 24, which joins 'employees' and 'employeeUNI' on 'id'. The results are shown in a table below.

The table has two columns: 'unique_id' and 'name'. The data is as follows:

	unique_id	name
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 the MySQL Workbench interface. At the top, there is a terminal window titled 'create-db-template.sql' with the following SQL code:

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

Below the terminal, there is a results table titled 'Users' with the following data:

	id	name
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 code editor with a file named 'create-db-template.sql'. The code defines a 'Calls' table and inserts five rows of data. It then executes a query to group calls by person and calculate total duration. The results are shown in a table titled 'Calls'.

Code content:

```
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;
23
```

Table Results:

	person1	person2	call_count	total_duration
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 client interface with two tabs: 'create-db-template.sql' and 'Data'.

In the 'create-db-template.sql' tab, the following SQL code is shown:

```
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  create table unitssold
11  (product_id int,
12  purchase_date date,
13  units int
14  );
15  insert into prices VALUES
16  (1, '2019-02-17', '2019-02-28', 5),
17  (1, '2019-03-01', '2019-03-22', 20),
18  (2, '2019-02-01', '2019-02-20', 15),
19  (2, '2019-02-21', '2019-03-31', 30);
20  insert into unitssold VALUES
21  (1, '2019-02-25', 100),
22  (1, '2019-03-01', 15),
23  (2, '2019-02-10', 200),
24  (2, '2019-03-22', 30);
25  select p.product_id, round(sum(u.units*p.price)/sum(u.units),2) as average_price
26  from
27  prices p
28  left join
29  unitssold u
30  on p.product_id = u.product_id
31  where u.purchase_date >= start_date and u.purchase_date <= end_date
32  group by product_id
33  order by product_id;
```

In the 'Data' tab, the results of the query are displayed:

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;
```

```
create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
  3  create table warehouse
  4    (name  varchar(15),
  5     product_id int,
  6     units int,
  7     primary key(name, product_id)
  8   );
  9   └ Execute
10  create table products
11    (product_id int,
12     product_name varchar(15),
13     Width int,
14     Length int,
15     Height int,
16     primary key(product_id)
17   );
18   └ Execute
19  insert into warehouse values
20    ('LCHouse1', 1, 1),
21    ('LCHouse1', 2, 10),
22    ('LCHouse1', 3, 5),
23    ('LCHouse2', 1, 2),
24    ('LCHouse2', 2, 2),
25    ('LCHouse3', 4, 1);
26   └ Execute
27  insert into products values
28    (1, 'LC-TV', 5, 50, 40),
29    (2, 'LC-KeyChain', 5, 5, 5),
30    (3, 'LC-Phone', 2, 10, 10),
31    (4, 'LC-T-Shirt', 4, 10, 20);
32   └ Execute
33  select w.name as warehouse_name, sum(p.width*p.length*p.height*w.units) as volume
34  from
35  warehouse w
36  left join
37  products p
38  on w.product_id = p.product_id
39  group by w.name
40  order by w.name;
```

Data

	warehouse_name	volume
1	LCHouse1	12250
2	LCHouse2	20250
3	LCHouse3	800

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;
```

The screenshot shows a terminal window in VS Code executing a SQL script named `create-db-template.sql`. The script creates a `sales` table and inserts data for four days in May 2020. It then runs a query to calculate the difference in sales between apples and oranges for each day.

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql >
  1  create table sales
  2    (sale_date date,
  3     Fruit  varchar(10),
  4     sold_num int,
  5     primary key(sale_date, fruit));
  6   > Execute
  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   > Execute
 17   ✓ 16  select t.sale_date, (t.apples_sold - t.oranges_sold) as diff
 18   from
 19   (select sale_date,
 20   |   max(CASE WHEN fruit = 'apples' THEN sold_num ELSE 0 END )as apples_sold,
 21   |   max(CASE WHEN fruit = 'oranges' THEN sold_num ELSE 0 END )as oranges_sold
 22   FROM sales
 23   group by sale_date) t
 24   ORDER BY t.sale_date;
```

The results table shows the following data:

	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 x
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.s
2  create table sales
3    (sale_date  date,
4     Fruit      varchar(10),
5     sold_num   int,
6     primary key(sale_date, fruit));
> Execute
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);
> Execute
✓ 16  select sale_date,
17    sum(case when fruit = 'apples' then sold_num
18    else (-sold_num) end) as diff
19  from sales
20  group by sale_date;
21  |
22
```

sales x

select sale_date,
sum(case when fruit = 'apples' then sold_num
else (-sold_num) end) as diff

Free 1

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

Input to filter result

Cost: 4ms < 1 > Total 4

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 > 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)
  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;
 23
```

activity X

```
select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
from
```

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

	fraction	newdecimal
1	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 themself.

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

```
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    );  
10   ▷ 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);  
16   ▷ 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

```
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;
```

Free 1

Input to filter result

	name	department
1	John	A

Cost: 3ms <

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:

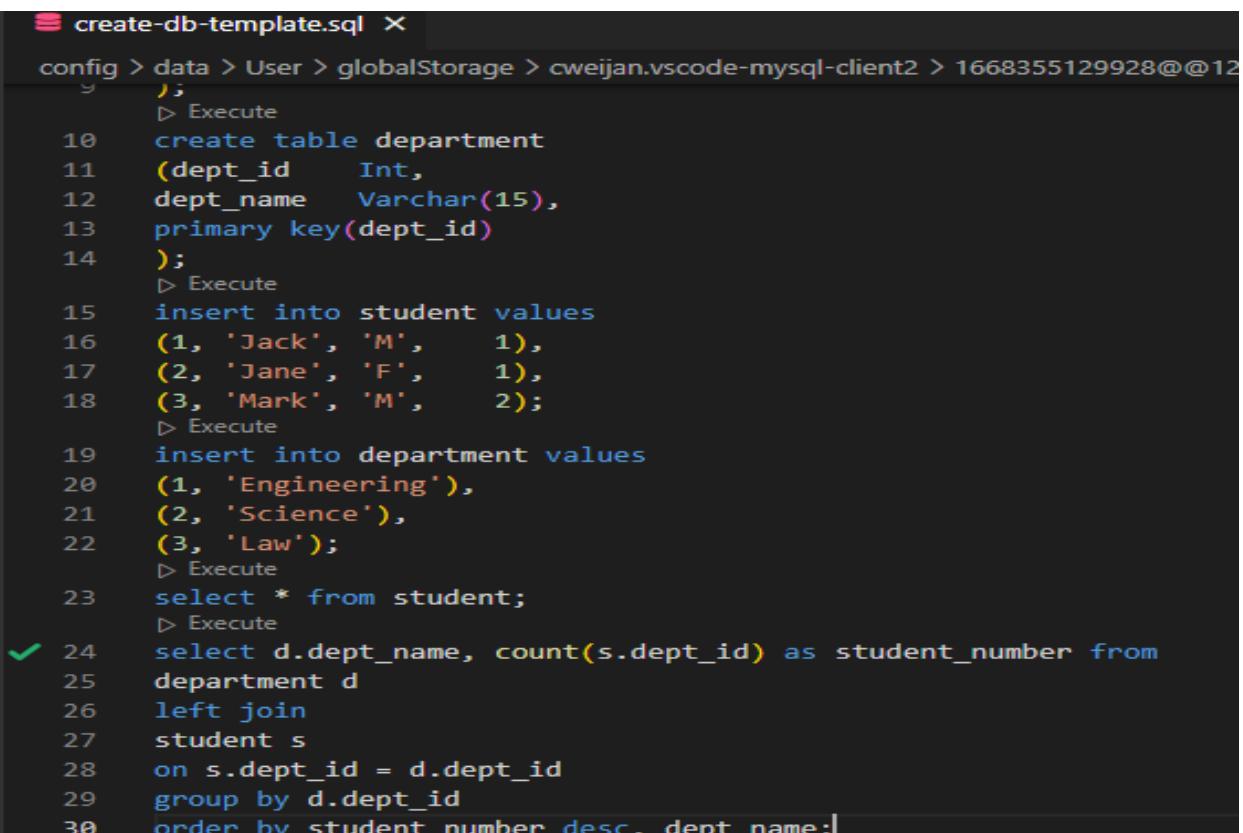
dept_id	dept_name
1	Engineering
2	Science
3	Law

Output:

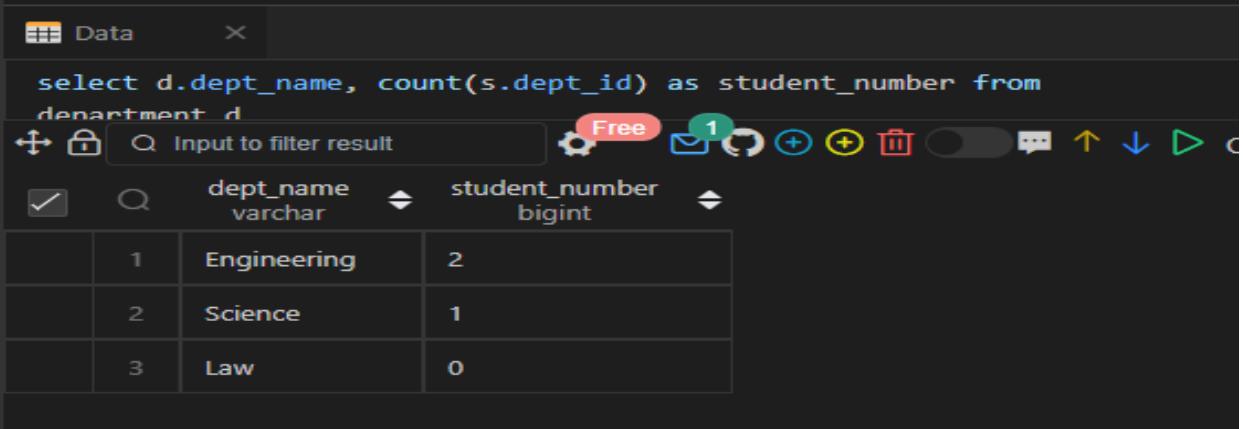
	student_number
dept_name	
Engineering	2
Science	1
Law	0

Solution:

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



```
create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@12
  ↴ ;
    ⌄ Execute
10   create table department
11     (dept_id      Int,
12      dept_name   Varchar(15),
13      primary key(dept_id)
14    );
    ⌄ Execute
15   insert into student values
16     (1, 'Jack', 'M',    1),
17     (2, 'Jane', 'F',    1),
18     (3, 'Mark', 'M',    2);
    ⌄ Execute
19   insert into department values
20     (1, 'Engineering'),
21     (2, 'Science'),
22     (3, 'Law');
    ⌄ Execute
23   select * from student;
    ⌄ Execute
✓ 24   select d.dept_name, count(s.dept_id) as student_number from
25     department d
26   left join
27     student s
28   on s.dept_id = d.dept_id
29   group by d.dept_id
30   order by student_number desc, dept_name;|
```



	dept_name	student_number
1	Engineering	2
2	Science	1
3	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 the MySQL Workbench interface. At the top, there is a code editor window titled "create-db-template.sql" containing SQL code for creating a database, tables, and inserting data. The code includes a section for creating a "product" table, a "customer" table with a foreign key constraint, and data insertions for both tables. Below the code editor is a results window titled "customer" showing the query results. The results show two rows: one for customer_id 1 with product_id 1, and another for customer_id 3 with product_id 3. The results window has a "Cost: 4ms" label at the bottom right.

```
3   create table product
4     (product_key int,
5      primary key(product_key));
6   create table customer
7     (customer_id    int,
8      product_key int,
9      foreign key(product_key) references product(product_key)
10    );
11  insert into product values
12    (5),
13    (6);
14  insert into customer values
15    (1, 5),
16    (2, 6),
17    (3, 5),
18    (3, 6),
19    (1, 6);
20  select customer_id
21    from
22      customer
23    group by customer_id
24    having count(distinct product_key)=(select count(*) from product);
```

	customer_id	product_id
1	1	5
2	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 code editor with a MySQL database connection open. The connection details are: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql. The code in the editor is the SQL query provided above. Below the editor is a 'Data' tab showing the results of the query. The results table has columns 'project_id' and 'employee_id'. The data is:

project_id	employee_id
1	1
1	3
2	1

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 > cweijan.vscode-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
  ↴ Input to filter result
  ↴ Free 1
  ↴ + - ×
  ↴ ↑ ↓ ⟲ ⟳ Cost 3ms < 1 > Total 3
  ↴
  ↴ book_id int name varchar
  ↴
  ↴ 1 1 "Kalila And Demna"
  ↴
  ↴ 2 2 "28 Letters"
  ↴
  ↴ 3 5 "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;
```

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



```
enrollments X
select t.student_id, t.course_id, t.grade
from
 Free 1


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


Cost: 5ms < 1 > Total 3
```

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;

```

```
create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  4   use test;
  5   create table Players
  6   (
  7     player_id    Int primary key,
  8     group_id    Int
  9   );
 10  create table Matches
 11  (
 12    match_id    Int primary key,
 13    first_player  Int,
 14    second_player  Int,
 15    first_score Int,
 16    second_score Int
 17  );
 18  insert into Players values
 19  (15,      '1'),
 20  (25,      '1'),
 21  (30,      '1'),
 22  (45,      '1'),
 23  (10,      '2'),
 24  (35,      '2'),
 25  (50,      '2'),
 26  (20,      '3'),
 27  (40,      '3');
 28  insert into Matches values
 29  (1, 15, 45, 3, 0),
 30  (2, 30, 25, 1, 2),
 31  (3, 30, 15, 2, 0),
 32  (4, 40, 20, 5, 2),
 33  (5, 35, 50, 1, 1);
 34  select t2.group_id, t2.player_id from
 35  (
 36    select t1.group_id, t1.player_id,
 37    dense_rank() over(partition by group_id order by score desc, player_id) as r
 38  from
 39  (
 40    select p.*, case when p.player_id = m.first_player then m.first_score
 41    when p.player_id = m.second_player then m.second_score
 42    end as score
 43  from
 44  Players p, Matches m
 45  where player_id in (first_player, second_player)
 46  |   ) t1
 47  ) t2
 48  where r = 1;
```

Players X

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

Free 1

Input to filter result.

	group_id	player_id
1	1	15
2	2	35
3	3	40

Cost 12ms < 1 > Total 3

Q51.

World table:

Column Name	Type
name	varchar
continent	varchar
area	Int
population	Int
gdp	Int

name is the primary key column for this table.

Each row of this table gives information about the name of a country, the continent to which it belongs, its area, the population, and its GDP value.

A country is big if:

- it has an area of at least three million (i.e., 3000000 km²), or
- it has a population of at least twenty-five million (i.e., 25000000).

Write an SQL query to report the name, population, and area of the big countries.

Return the result table in any order.

The query result format is in the following example.

Input:

World table:

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

Output:

name	population	Area
Afghanistan	25500100	652230
Algeria	37100000	2381741

Solution:

```
select name, population, area
from
World
where area >= 3000000 or population >= 25000000;
```

```
create-db-template.sql ✘

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  ↴ execute
  2  use test;
  3  ▷ Execute
  4  create table World
  5  (
  6    name varchar(15) primary key,
  7    continent  varchar(15),
  8    area      bigint,
  9    population bigint,
 10   gdp      bigint
 11 );
  12 ▷ Execute
 13 insert into World values
 14 ('Afghanistan', 'Asia', 652230, 25500100, 20343000000),
 15 ('Albania', 'Europe', 28748, 2831741, 12960000000),
 16 ('Algeria', 'Africa', 2381741, 37100000, 18868100000),
 17 ('Andorra', 'Europe', 468, 78115, 3712000000),
 18 ('Angola', 'Africa', 1246700, 20609294, 100990000000);
 19 ▷ Execute
 20 select name, population, area
 21 from
 22 World
 23 where area >= 3000000 or population >= 25000000;
 24 |
```

World X

select name, population, area
from

Free 1

	name	population	area
1	Afghanistan	25500100	652230
2	Algeria	37100000	2381741

Cost: 5ms < 1 > Total 2

Q52.

Table: Customer

Column Name	Type
id	int
name	varchar
referee_id	int

id is the primary key column for this table.

Each row of this table indicates the id of a customer, their name, and the id of the customer who referred them.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Customer table:

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

Output:

Name
Will
Jane
Bill
Zack

Solution:

```
select name
from
Customer
where referee_id != 2 or referee_id is NULL;
```

create-db-template.sql X

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1
 Active Connection | ▶ Execute

```
1  create database test;
2  use test;
3  create table Customer
4  (
5      id int primary key,
6      name varchar(10),
7      refree_id int
8  );
9  insert into Customer values
10 (1, 'Will', Null),
11 (2, 'Jane', Null),
12 (3, 'Alex', 2),
13 (4, 'Bill', Null),
14 (5, 'Zack', 1),
15 (6, 'Mark', 2);
16 select name
17 from
18 Customer
19 where refree_id != 2 or refree_id is NULL;
20
```

Customer X

select name from Customer where refree_id != 2 or refree_id is NULL

Free 1

Input to filter result

Cost: 3

	name
1	Will
2	Jane
3	Bill
4	Zack

Q53.

Table: Customers

Column Name	Type
id	int
name	varchar

id is the primary key column for this table.

Each row of this table indicates the ID and name of a customer.

Table: Orders

Column Name	Type
id	int
customerId	int

id is the primary key column for this table.

customerId is a foreign key of the ID from the Customers table.

Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Customers table:

id	name
1	Joe
2	Henry
3	Sam
4	Max

Orders table:

id	customerId
1	3
2	1

Output:

Customers
Henry
Max

Solution:

```
select c.name
from Customers c
left join
Orders o
on c.id = o.customerID
where o.id is NULL;
```

The screenshot shows a code editor with an open file named `create-db-template.sql`. The file contains an SQL script to create two tables, `Customers` and `Orders`, and insert data into them. A specific query at the bottom is highlighted with a green checkmark, indicating it is the solution to the problem. Below the code editor is a database interface window titled "Customer" showing the results of the query.

```
1 config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@330
2   use test;
3   create table Customers
4     (id int primary key,
5      name    varchar(10)
6    );
7   create table Orders
8     (id int primary key,
9      customerId int,
10     foreign key(customerID) references Customers(id)
11   );
12  insert into Customers values
13    (1, 'Joe'),
14    (2, 'Henry'),
15    (3, 'Sam'),
16    (4, 'Max');
17  insert into Orders values
18    (1, 3),
19    (2,1);
20  select c.name
21    from Customers c
22  left join
23    Orders o
24  on c.id = o.customerID
25  where o.id is NULL;
26
```

Customer

		name	Type
<input checked="" type="checkbox"/>	1	Henry	varchar
	2	Max	

Q54.

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.

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:

```
count(team_id) over(partition by team_id) as team_size  
from Employee  
order by employee_id;
```

create-db-template.sql X

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

```
1  create database test;
2  use test;
3  create table Employee
4  (
5    employee_id int primary key,
6    team_id int
7  );
8  insert into Employee values
9  (1, 8),
10 (2, 8),
11 (3, 8),
12 (4, 7),
13 (5, 9),
14 (6, 9);
15 select employee_id,
16 count(team_id) over(partition by team_id) as team_size
17 from Employee
18 order by employee_id;
```

Employee X

select employee_id,
count(team_id) over(partition by team_id) as team_size

Free 1

Input to filter result

employee_id team_size

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

Cost: 5ms < 1 > Total 6

Q55

Table Person:

Column Name	Type
Id	int
Name	varchar
phone_number	varchar

id is the primary key for this table.

Each row of this table contains the name of a person and their phone number.

Phone number will be in the form 'xxx-yyyyyy' where xxx is the country code (3 characters) and yyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
Name	varchar
country_code	varchar

country_code is the primary key for this table.

Each row of this table contains the country name and its code. country_code will be in the form 'xxx' where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
Duration	int

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

Each row of this table contains the caller id, callee id and the duration of the call in minutes. caller_id != callee_id

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Person table:

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

Country table:

Name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251

Calls table:

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

Output:

country
Peru

Explanation:

The average call duration for Peru is $(102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667$

The average call duration for Israel is $(33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500$

The average call duration for Morocco is $(33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000$

Global call duration average = $(2 * (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000$

Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

Solution:

```
select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl.callee_id as id, cl.duration
from Calls cl)) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;
```

QLQuery2.sql - LAP...ARTA.test (sa (65))* ⇐ × SQLQuery1.sql - not connected

```
| ('Morocco', 212),
| ('Germany', 49),
| ('Ethiopia', 251);
insert into Calls values
(1, 9, 33),
(2, 9, 4),
(1, 2, 59),
(3, 12, 102),
(3, 12, 330),
(12, 3, 5),
(7, 9, 13),
(7, 1, 3),
(9, 7, 1),
(1, 7, 7);
select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl.callee_id as id, cl.duration
from Calls cl)) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;
```

00 %

Results Messages

Name
Peru

Query executed successfully.

Q56.

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;
```

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 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.device_id
16 from (select player_id, device_id, 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.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;

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

	player_id	device_id
1	1	2
2	2	3
3	3	1

Q57.

Table: Orders

Column Name	Type
order_number	int
customer_number	int

order_number is the primary key for this table.

This table contains information about the order ID and the customer ID.

Write an SQL query to find the customer_number for the customer who has placed the largest number of orders.

The test cases are generated so that exactly one customer will have placed more orders than any other customer.

The query result format is in the following example.

Input:

Orders table:

order_number	customer_number
1	1
2	2
3	3
4	3

Output:

customer_number
3

Explanation:

The customer with number 3 has two orders, which is greater than either customer 1 or 2 because each of them only has one order.

So the result is customer_number 3.

Solution:

```
select customer_number
from
Orders
group by customer_number
order by count(order_number) desc
limit 1;
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@
  1  create database test;
     ▶ Execute
  2  use test;
     ▶ Execute
  3  create table Orders
  4    (order_number int,
  5     customer_number int,
  6     primary key(order_number)
  7   );
     ▶ Execute
  8  insert into Orders values
  9    (1, 1),
 10   (2, 2),
 11   (3, 3),
 12   (4, 3);
     ▶ Execute
 13  select customer_number
 14  from
 15  Orders
 16  group by customer_number
 17  order by count(order_number) desc
 18  limit 1;

```

Orders	
<pre>select customer_number</pre>	
From	customer_number
Free	1

Follow up: What if more than one customer has the largest number of orders, can you find all the customer_number in this case?

Ans: To find all such customers, we will use dense_rank() order by count(order_number desc). Now, all those customers who have placed the largest number of orders will get rank 1. Then, we select all those customers who are having rank 1.

Solution:

```

select t.customer_number
from
(select customer_number,
dense_rank() over(order by count(order_number) desc) as r
from
Orders
group by customer_number) t
where t.r = 1;

```

Q58.

Table: Cinema

Column Name	Type
seat_id	Int
Free	Bool

seat_id is an auto-increment primary key column for this table.

Each row of this table indicates whether the ith seat is free or not. 1 means free while 0 means occupied.

Write an SQL query to report all the consecutive available seats in the cinema.

Return the result table ordered by seat_id in ascending order.

The test cases are generated so that more than two seats are consecutively available. The query result format is in the following example.

Input:

Cinema table:

seat_id	Free
1	1
2	0
3	1
4	1
5	1

Output:

seat_id
3
4
5

Solution:

```
select t.seat_id
from
(select seat_id, lead(seat_id,1,seat_id) over(order by seat_id) as next
from Cinema
where Free != 0
) t
where next - seat_id in (0,1)
order by seat_id;
```

```
create-db-template.sql ×

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  ↗ Active Connection | ▶ Execute
1   create database test;
  ▷ Execute
2   use test;
  ▷ Execute
3   create table Cinema
4     (seat_id int AUTO_INCREMENT PRIMARY KEY,
5      Free boolean
6    );
  ▷ Execute
7   insert into Cinema values
8     (1, 1),
9     (2, 0),
10    (3, 1),
11    (4, 1),
12    (5, 1);
  ▷ Execute
13  select t.seat_id
14  from
15  (select seat_id, lead(seat_id,1,seat_id) over(order by seat_id) as next
16  from Cinema
17  where Free != 0
18  ) t
19  where next - seat_id in (0,1)
20  order by seat_id;
21
```

```
select t.seat_id
from
Input to filter result
Free 1 🔍 + + 🗑️ 🔍 ↑ ↓ ⏴ Cost: 10ms < 1 > Total
```

	seat_id	int
1	3	
2	4	
3	5	

Q59:

Table: SalesPerson

Column Name	Type
sales_id	Int
Name	varchar
Salary	Int
commission_rate	Int
hire_date	date

sales_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a salesperson alongside their salary, commission rate, and hire date.

Table: Company

Column Name	Type
com_id	Int
Name	varchar
City	varchar

com_id is the primary key column for this table.

Each row of this table indicates the name and the ID of a company and the city in which the company is located.

Table: Orders

Column Name	Type
order_id	Int
order_date	Date
com_id	Int
sales_id	Int
Amount	Int

order_id is the primary key column for this table.

com_id is a foreign key to com_id from the Company table.

sales_id is a foreign key to sales_id from the SalesPerson table.

Each row of this table contains information about one order. This includes the ID of the company, the ID of the salesperson, the date of the order, and the amount paid.

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

Return the result table in any order.

The query result format is in the following example.

Input:

SalesPerson table:

sales_id	Name	salary	commission_rate	hire_date
1	John	100000	6	4/1/2006
2	Amy	12000	5	5/1/2010
3	Mark	65000	12	12/25/2008
4	Pam	25000	25	1/1/2005
5	Alex	5000	10	2/3/2007

Company table:

com_id	Name	City
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin

Orders table:

order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

Output:

Name
Amy
Mark
Alex

Explanation:

According to orders 3 and 4 in the Orders table, it is easy to tell that only salesperson John and Pam have sales to company RED, so we report all the other names in the table salesperson.

Solution:

```
select Name from SalesPerson
where sales_id
not in
(select o.sales_id
from
Orders o
left join
Company c
on o.com_id = c.com_id
where c.Name = 'Red');
```

The screenshot shows a split interface in VS Code. On the left, the code editor displays a file named 'create-db-template.sql' containing SQL scripts for creating tables and inserting data into SalesPerson, Company, and Orders. On the right, a results table titled 'SalesPerson' shows three rows of data: Amy, Mark, and Alex.

	Name
1	Amy
2	Mark
3	Alex

Q60.

Table: Triangle

Column Name	Type
X	Int
Y	Int
Z	Int

(x, y, z) is the primary key column for this table.

Each row of this table contains the lengths of three line segments.

Write an SQL query to report for every three line segments whether they can form a triangle. Return the result table in any order.

The query result format is in the following example.

Input: Triangle

table:

X	Y	Z
13	15	30
10	20	15

Output:

X	Y	Z	triangle
13	15	30	No
10	20	15	Yes

Solution:

```
select X, Y, Z, (case
when X+Y > Z and Y+Z > X and Z+X > Y then 'Yes'
else 'No'
end) as triangle
from Triangle;
```

create-db-template.sql ×

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129

7 ▷ Execute

8 create table Triangle

9 (X int,

10 Y int,

11 Z int,

12 PRIMARY KEY(X,Y,Z)

13);

14 ▷ Execute

15 insert into Triangle values

16 (13, 15, 30),

17 (10, 20, 15);

18 ▷ Execute

19 select X, Y, Z, (case
20 when X+Y > Z and Y+Z > X and Z+X > Y then 'Yes'
21 else 'No'
22 end) as triangle
23 from Triangle;

Triangle ×

select X, Y, Z, (case
when X+Y > Z and Y+Z > X and Z+X > Y then 'Yes'

Free 1

Input to filter result

	X	Y	Z	triangle
1	13	15	30	No
2	10	20	15	Yes

Q61.

Table: Point

Column Name	Type
X	Int

x is the primary key column for this table.

Each row of this table indicates the position of a point on the X-axis.

Write an SQL query to report the shortest distance between any two points from the Point table.
The query result format is in the following example.

Input: Point

table:

X
-1
0
2

Output:

shortest
1

Explanation:

The shortest distance is between points -1 and 0 which is $|(-1) - 0| = 1$.

Follow up: How could you optimise your query if the Point table is ordered in ascending order?

Ans: If we arrange the points in ascending order, then we only have to check the difference between consecutive numbers, this will decrease the number of comparisons we have to check.

Solution:

```
select min(t.diff) as shortest
from
(select lead(X,1) over(order by X) - X as diff
from
Point) t;
```

The screenshot shows the MySQL Workbench interface. At the top, it displays a connection configuration: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306. Below this, a code editor window contains the following SQL script:

```
1 create database test;
2 use test;
3 create table Point
4 (X int);
5 insert into Point values
6 (-1),
7 (0),
8 (2);
9 select min(t.diff) as shortest
10 from
11 (select lead(X,1) over(order by X) - X as diff
12 from
13 Point) t;
```

Below the code editor, the results of the query execution are shown in a table:

shortest
1

At the bottom of the interface, there are various status indicators and a toolbar.

Q62.

Table: ActorDirector

Column Name	Type
actor_id	Int
director_id	Int
timestamp	Int

timestamp is the primary key column for this table.

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

Return the result table in any order.

The query result format is in the following example.

Input:

ActorDirector table:

actor_id	director_id	Timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

Output:

actor_id	director_id
1	1

Explanation:

The only pair is (1, 1) where they cooperated exactly 3 times.

Solution:

```
select actor_id, director_id
from
ActorDirector
group by actor_id, director_id
having count(*) >= 3;
```

create-db-template.sql X

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

```
3  create table ActorDirector
4      (actor_id    Int,
5       director_id Int,
6       timestamp   Int primary key
7   );
8   ▷ Execute
9   insert into ActorDirector values
10  (1, 1, 0),
11  (1, 1, 1),
12  (1, 1, 2),
13  (1, 2, 3),
14  (1, 2, 4),
15  (2, 1, 5),
16  (2, 1, 6);
17   ▷ Execute
18 ✓ 16  select actor_id, director_id
19  from
20  ActorDirector
21  group by actor_id, director_id
22  having count(*) >= 3;
23
```

ActorDirector X

```
select actor_id, director_id
from
```

Free 1

Input to filter result

actor_id int director_id int

	1	1	1

Cost: 5ms <

Q63

Table: Sales

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

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

product_id is a foreign key to the Product table.

Each row of this table shows a sale on the product product_id in a certain year. Note that the price is per unit.

Table: Product

Column Name	Type
product_id	int
product_name	varchar

product_id is the primary key of this table.

Each row of this table indicates the product name of each product.

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

Return the resulting table in any order.

The query result format is in the following example.

Input:

Sales table:

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

Product table:

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Output:

product_name	year	Price
Nokia	2008	5000
Nokia	2009	5000
Apple	2011	9000

Explanation:

From sale_id = 1, we can conclude that Nokia was sold for 5000 in the year 2008.

From sale_id = 2, we can conclude that Nokia was sold for 5000 in the year 2009.

From sale_id = 7, we can conclude that Apple was sold for 9000 in the year 2011.

Solution:

```
select p.product_name, s.year,
sum(price) as price
from
Sales s
left join
Product p
on s.product_id = p.product_id
group by p.product_name, s.year;
```

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 Product
4  (product_id int primary key,
5  product_name varchar(10)
6  );
   ▶ Execute
7  create table Sales
8  (
9  sale_id int,
10 product_id int,
11 year int,
12 quantity int,
13 price int,
14 primary key(sale_id, year),
15 foreign key(product_id) references Product(product_id)
16 );
   ▶ Execute
17 insert into Product values
18 (100, 'Nokia'),
19 (200, 'Apple'),
20 (300, 'Samsung');
   ▶ Execute
21 insert into Sales values
22 (1, 100, 2008, 10, 5000),
23 (2, 100, 2009, 12, 5000),
24 (7, 200, 2011, 15, 9000);
   ▶ Execute
25 select p.product_name, s.year,
26 sum(price) as price
27 from
28 Sales s
29 left join
30 Product p
31 on s.product_id = p.product_id
32 group by p.product_name, s.year;
33
```

ActorDirector X

```
select p.product_name, s.year, sum(price) as price
from
```

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

	product_name	year	price
	varchar	int	newdecimal
1	Nokia	2008	5000
2	Nokia	2009	5000
3	Apple	2011	9000

Q64.

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 average experience years of all the employees for each project, rounded to 2 digits.

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	1
4	Doe	2

Output:

project_id	average_years
1	2
2	2.5

Explanation:

The average experience years for the first project is $(3 + 2 + 1) / 3 = 2.00$ and for the second project is $(3 + 2) / 2 = 2.50$

Solution:

```
select p.project_id, round(avg(e.experience_years),2) as average_years
from
Project p
left join
Employee e
on p.employee_id = e.employee_id
group by p.project_id;
```

```
create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql
  ▷ Execute
  2 use test;
  ▷ Execute
  3 create table Employee
  4   (employee_id int primary key,
  5    name varchar(15),
  6    experience_years int
  7   );
  ▷ Execute
  8 create table Project
  9   (project_id int,
 10    employee_id int,
 11    primary key(project_id, employee_id),
 12    foreign key(employee_id) references Employee(employee_id)
 13   );
  ▷ Execute
 14 insert into Employee values
 15   (1, 'Khaled', 3),
 16   (2, 'Ali', 2),
 17   (3, 'John', 1),
 18   (4, 'Doe', 2);
  ▷ Execute
 19 insert into Project values
 20   (1, 1),
 21   (1, 2),
 22   (1, 3),
 23   (2, 1),
 24   (2, 4);
  ▷ Execute
 25 select p.project_id, round(avg(e.experience_years),2) as average_years
 26 from
 27 Project p
 28 left join
 29 Employee e
 30 on p.employee_id = e.employee_id
 31 group by p.project_id;
 32
ActorDirector ✘
select p.project_id, round(avg(e.experience_years),2) as average_years
from
Project p
left join
Employee e
on p.employee_id = e.employee_id
group by p.project_id;
```

Input to filter result: Free 1

	project_id	average_years
1	1	2.00
2	2	2.50

Cost: 2ms < 1 > Total 2

Q65.

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 best seller by total sales price, If there is a tie, report them all.

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:

seller_id
1
3

Explanation: Both sellers with id 1 and 3 sold products with the most total price of 2800.

Solution:

```
select t.seller_id
from
(select seller_id , sum(price),
dense_rank() over(order by sum(price) desc) as r
from Sales
group by seller_id) t
where t.r = 1;
```

The screenshot shows a MySQL client interface with two tabs: 'create-db-template.sql' and 'Sales'. The 'create-db-template.sql' tab contains the following code:

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  2  use test;
  3  create table Product
  4  (product_id int primary key,
  5  product_name  varchar(10),
  6  unit_price  int
  7  );
  8  create table Sales(
  9  seller_id  int,
 10  product_id  int,
 11  buyer_id  int,
 12  sale_date  date,
 13  quantity  int,
 14  price  int,
 15  foreign key(product_id) references Product(product_id)
 16  );
  17  insert into Product values
 18  (1, 'S8', 1000),
 19  (2, 'G4', 800),
 20  (3, 'iPhone', 1400);
  21  insert into Sales values
 22  (1, 1, 1, '2019-01-21', 2, 2000),
 23  (1, 2, 2, '2019-02-17', 1, 800),
 24  (2, 2, 3, '2019-06-02', 1, 800),
 25  (3, 3, 4, '2019-05-13', 2, 2800);
  26  select t.seller_id
 27  from
 28  (select seller_id , sum(price), dense_rank() over(order by sum(price) desc) as r
 29  from Sales
 30  group by seller_id) t
 31  where t.r = 1;
```

The 'Sales' tab shows the results of the query:

seller_id	1	3
1	1	
2	3	

At the bottom of the interface, there are various status icons and a message bar indicating 'Cost: 4ms'.

Q66.

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 buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

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:

buyer_id
1

Explanation:

The buyer with id 1 bought an S8 but did not buy an iPhone. The buyer with id 3 bought both.

Soltion:

```
select buyer_id
from
(
    select t1.buyer_id,
    sum(case when t1.product_name = 'S8' then 1 else 0 end) as S8_count,
    sum(case when t1.product_name = 'iPhone' then 1 else 0 end) as iphone_count
from
(
    select s.buyer_id, p.product_name
from
Sales s
left join
Product p
on s.product_id = p.product_id
) t1
group by t1.buyer_id
) t2
where t2.S8_count = 1 and t2.iphone_count = 0;
```

create-db-template.sql

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

```
8  create table Sales(
9    seller_id  int,
10   product_id  int,
11   buyer_id  int,
12   sale_date  date,
13   quantity  int,
14   price  int,
15   foreign key(product_id) references Product(product_id)
16 );
16   ▷ Execute
17   insert into Product values
18   (1, 'S8', 1000),
19   (2, 'G4', 800),
20   (3, 'iPhone', 1400);
20   ▷ Execute
21   insert into Sales values
22   (1, 1, 1, '2019-01-21', 2, 2000),
23   (1, 2, 2, '2019-02-17', 1, 800),
24   (2, 2, 3, '2019-06-02', 1, 800),
25   (3, 3, 4, '2019-05-13', 2, 2800);
25   ▷ Execute
26 select buyer_id
27 from
28 (
29   |  select t1.buyer_id,
30   |  sum(case when t1.product_name = 'S8' then 1 else 0 end) as S8_count,
31   |  sum(case when t1.product_name = 'iPhone' then 1 else 0 end) as iphone_count
32   |  from
33   |  (
34   |    |  select s.buyer_id, p.product_name
35   |    |  from
36   |    Sales s
37   |    left join
38   |    Product p
39   |    on s.product_id = p.product_id
40   |  ) t1
41   |  group by t1.buyer_id
42   |  ) t2
43 where t2.S8_count = 1 and t2.iphone_count = 0;
44
```

Data

```
select buyer_id
from

Free 1 + + - ? ↑ ↓ ▷ Cost: 2ms < 1 > Total 1
```

buyer_id	int
1	1

Q67.

Table: Customer

Column Name	Type
customer_id	Int
Name	Varchar
visited_on	Date
Amount	Int

(customer_id, visited_on) is the primary key for this table.

This table contains data about customer transactions in a restaurant.

visited_on is the date on which the customer with ID (customer_id) has visited the restaurant.

amount is the total paid by a customer.

You are the restaurant owner and you want to analyse a possible expansion (there will be at least one customer every day).

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

The query result format is in the following example.

Input:

Customer table:

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

Output:

visited_on	amount	average_amount
2019-01-07	860	122.86

2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

Explanation:

1st moving average from 2019-01-01 to 2019-01-07 has an average_amount of $(100 + 110 + 120 + 130 + 110 + 140 + 150)/7 = 122.86$

2nd moving average from 2019-01-02 to 2019-01-08 has an average_amount of $(110 + 120 + 130 + 110 + 140 + 150 + 80)/7 = 120$

3rd moving average from 2019-01-03 to 2019-01-09 has an average_amount of $(120 + 130 + 110 + 140 + 150 + 80 + 110)/7 = 120$

4th moving average from 2019-01-04 to 2019-01-10 has an average_amount of $(130 + 110 + 140 + 150 + 80 + 110 + 130 + 150)/7 = 142.86$

Solution:

```
select t2.visited_on, t2.amount, t2.average_amount
from
(select t1.visited_on, t1.prev_date_interval_6,
round(sum(amount) over(order by visited_on range between interval '6' day
preceding and current row),2) as amount,
round(avg(amount) over(order by visited_on range between interval '6' day
preceding and current row),2) as average_amount
from
(select visited_on, sum(amount) as amount,
lag(visited_on,6) over(order by visited_on) as prev_date_interval_6
from Customer
group by visited_on
order by visited_on) t1
) t2
where prev_date_interval_6 is not null;
```

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 Customer
4    (customer_id  int,
5     name  varchar(15),
6     visited_on date,
7     amount  int,
8     primary key(customer_id, visited_on)
9   );
▷ Execute
10 insert into Customer values
11   (1, 'Jhon', '2019-01-01', 100),
12   (2, 'Daniel', '2019-01-02', 110),
13   (3, 'Jade', '2019-01-03', 120),
14   (4, 'Khaled', '2019-01-04', 130),
15   (5, 'Winston', '2019-01-05', 110),
16   (6, 'Elvis', '2019-01-06', 140),
17   (7, 'Anna', '2019-01-07', 150),
18   (8, 'Maria', '2019-01-08', 80),
19   (9, 'Jaze', '2019-01-09', 110),
20   (1, 'Jhon' , '2019-01-10', 130),
21   (3, 'Jade', '2019-01-10', 150);
▷ Execute
22 select t2.visited_on, t2.amount, t2.average_amount
23 from
24 (select t1.visited_on, t1.prev_date_interval_6,
25 round(sum(amount) over(order by visited_on range between interval '6' day preceding and current row),2) as amount,
26 round(avg(amount) over(order by visited_on range between interval '6' day preceding and current row),2) as average_amount
27 from
28 (select visited_on, sum(amount) as amount,
29 lag(visited_on,6) over(order by visited_on) as prev_date_interval_6
30 from Customer
31 group by visited_on
32 order by visited_on) t1
33 ) t2
34 where prev_date_interval_6 is not null;
35

```

Customer X

select t2.visited_on, t2.amount, t2.average_amount
from

	visited_on date	amount newdecimal	average_amount newdecimal
<input checked="" type="checkbox"/>	1 2019-01-07	860	122.86
<input type="checkbox"/>	2 2019-01-08	840	120.00
<input type="checkbox"/>	3 2019-01-09	840	120.00
<input type="checkbox"/>	4 2019-01-10	1000	142.86

Free 1 Cost: 7ms < 1 > Total 4

Q68.

Table: Scores

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day) is the primary key for this table.

A competition is held between the female team and the male team.

Each row of this table indicates that a player_name and with gender has scored score_point in someday.

Gender is 'F' if the player is in the female team and 'M' if the player is in the male team.

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

Return the result table ordered by gender and day in ascending order.

The query result format is in the following example.

Input:

Scores table:

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

Output:

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

M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

Explanation:

For the female team:

The first day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.

The second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.

The third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

The fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For the male team:

The first day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

The second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.

The third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.

The fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

The fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

Solution:

```
select gender, day,
sum(score_points) over(partition by gender order by day) as total
from Scores
group by gender, day
order by gender, day;
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Execute
3   create table Scores
4   (
5     player_name varchar(15),
6     gender  varchar(2),
7     day date,
8     score_points  int,
9     primary key(gender, day)
10  );
  ↗ Execute
11  insert into Scores values
12  ('Aron', 'F', '2020-01-01', 17),
13  ('Alice', 'F', '2020-01-07', 23),
14  ('Bajrang', 'M', '2020-01-07', 7),
15  ('Khali', 'M', '2019-12-25', 11),
16  ('Slaman', 'M', '2019-12-30', 13),
17  ('Joe', 'M', '2019-12-31', 3),
18  ('Jose', 'M', '2019-12-18', 2),
19  ('Priya', 'F', '2019-12-31', 23),
20  ('Priyanka', 'F', '2019-12-30', 17);
  ↗ Execute
✓ 21  select gender, day, sum(score_points) over(partition by gender order by day) as total
22  from Scores
23  group by gender, day
24  order by gender, day;

```

Scores

```

select gender, day, sum(score_points) over(partition by gender order by day) as total
from Scores

Free 1

|   | gender  | day        | total      |
|---|---------|------------|------------|
|   | varchar | date       | newdecimal |
| 1 | F       | 2019-12-30 | 17         |
| 2 | F       | 2019-12-31 | 40         |
| 3 | F       | 2020-01-01 | 57         |
| 4 | F       | 2020-01-07 | 80         |
| 5 | M       | 2019-12-18 | 2          |
| 6 | M       | 2019-12-25 | 13         |
| 7 | M       | 2019-12-30 | 26         |
| 8 | M       | 2019-12-31 | 29         |
| 9 | M       | 2020-01-07 | 36         |


```

Q69.

Table: Logs

Column Name	Type
log_id	int

log_id is the primary key for this table.

Each row of this table contains the ID in a log Table.

Write an SQL query to find the start and end number of continuous ranges in the table Logs.
Return the result table ordered by start_id.
The query result format is in the following example.

Input: Logs table:

log_id
1
2
3
7
8
10

Output:

start_id	end_id
1	3
7	8
10	10

Explanation:

The result table should contain all ranges in table Logs.

From 1 to 3 is contained in the table.

From 4 to 6 is missing in the table

From 7 to 8 is contained in the table.

Number 9 is missing from the table.

Number 10 is contained in the table.

Solution:

```
select distinct start.log_id as start_id,
min(end.log_id) over(partition by start.log_id) as end_id
from
(select log_id from Logs where log_id - 1 not in (select * from Logs)) start,
(select log_id from Logs where log_id + 1 not in (select * from Logs)) end
where start.log_id <= end.log_id;
```

create-db-template.sql ×

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

```
1  use test;
   ▶ Execute
2  create table Logs
3  (
4    log_id int primary key
5  );
   ▶ Execute
6  insert into Logs values
7  (1),
8  (2),
9  (3),
10 (7),
11 (8),
12 (10);
   ▶ Execute
13 select distinct start.log_id as start_id,
14 min(end.log_id) over(partition by start.log_id) as end_id
15 from
16 (select log_id from Logs where log_id - 1 not in (select * from Logs)) start,
17 (select log_id from Logs where log_id + 1 not in (select * from Logs)) end
18 where start.log_id <= end.log_id;
```

Data ×

select distinct start.log_id as start_id,
min(end.log_id) over(partition by start.log_id) as end_id

Free 1

Input to filter result

	start_id	end_id
1	1	3
2	7	8
3	10	10

Cost: 5ms < 1 >

Q70.

Table: Students

Column Name	Type
student_id	Int
student_name	Varchar

student_id is the primary key for this table.

Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

Column Name	Type
subject_name	Varchar

subject_name is the primary key for this table.

Each row of this table contains the name of one subject in the school.

Table: Examinations

Column Name	Type
student_id	Int
subject_name	Varchar

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

Each student from the Students table takes every course from the Subjects table.

Each row of this table indicates that a student with ID student_id attended the exam of subject_name.

Write an SQL query to find the number of times each student attended each exam.
Return the result table ordered by student_id and subject_name.

The query result format is in the following example.

Input: Students

table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

Subjects table:

subject_name
Math
Physics
Programming

Examinations table:

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

Output:

student_id	student_name	subject_name	attended_exams
1	Alice	Math	3
1	Alice	Physics	2
1	Alice	Programming	1
2	Bob	Math	1
2	Bob	Physics	0

2	Bob	Programming	1
6	Alex	Math	0
6	Alex	Physics	0
6	Alex	Programming	0
13	John	Math	1
13	John	Physics	1
13	John	Programming	1

Explanation:

The result table should contain all students and all subjects.

Alice attended the Math exam 3 times, the Physics exam 2 times, and the Programming exam 1 time.

Bob attended the Math exam 1 time, the Programming exam 1 time, and did not attend the Physics exam.

Alex did not attend any exams.

John attended the Math exam 1 time, the Physics exam 1 time, and the Programming exam 1 time.

Solution:

```
select t.student_id, t.student_name , t.subject_name,
count(e.subject_name) as attended_exams
from
(select student_id, student_name, subject_name
from Students, Subjects) t
left join
Examinations e
on t.student_id = e.student_id and t.subject_name = e.subject_name
group by t.student_id, t.subject_name
order by t.student_id, t.subject_name;
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 >
  ↻ execute
2  create table Subjects
3  (subject_name varchar(30),
4  primary key(subject_name));
  ↻ Execute
5  create table Students
6  (student_id int primary key,
7  student_name varchar(20));
  ↻ Execute
8  create table Examinations
9  (student_id Int,
10 subject_name  Varchar(30)
11 );
  ↻ Execute
12 insert into Students values
13 (1, 'Alice'),
14 (2, 'Bob'),
15 (13, 'John'),
16 (6, 'Alex');
  ↻ Execute
17 insert into Subjects values
18 ('Math'),
19 ('Physics'),
20 ('Programming');
  ↻ Execute
21 insert into Examinations values
22 (1, 'Math'),
23 (1, 'Physics'),
24 (1, 'Programming'),
25 (2, 'Programming'),
26 (1, 'Physics'),
27 (1, 'Math'),
28 (13, 'Math'),
29 (13, 'Programming'),
30 (13, 'Physics'),
31 (2, 'Math'),
32 (1, "Math");
  ↻ Execute
33 select t.student_id, t.student_name , t.subject_name,
34 count(e.subject_name) as attended_exams
35 from
36 (select student_id, student_name, subject_name
37 from Students, Subjects) t
38 left join
39 Examinations e
40 on t.student_id = e.student_id and t.subject_name = e.subject_name
41 group by t.student_id, t.subject_name
42 order by t.student_id, t.subject_name;
43
44

```

Free 1

	student_id	student_name	subject_name	attended_exams
1	1	Alice	Math	3
2	1	Alice	Physics	2
3	1	Alice	Programming	1
4	2	Bob	Math	1
5	2	Bob	Physics	0
6	2	Bob	Programming	1
7	6	Alex	Math	0
8	6	Alex	Physics	0
9	6	Alex	Programming	0
10	13	John	Math	1
11	13	John	Physics	1
12	13	John	Programming	1

Q71.

Table: Employees

Column Name	Type
employee_id	Int
employee_name	Varchar
manager_id	int

employee_id is the primary key for this table.

Each row of this table indicates that the employee with ID employee_id and name employee_name reports his work to his/her direct manager with manager_id

The head of the company is the employee with employee_id = 1.

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

The indirect relation between managers will not exceed three managers as the company is small.

Return the result table in any order.

The query result format is in the following example.

Input:

Employees table:

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

Output:

employee_id
2
77
4
7

Explanation:

The head of the company is the employee with employee_id 1.

The employees with employee_id 2 and 77 report their work directly to the head of the company.

The employee with employee_id 4 reports their work indirectly to the head of the company 4 → 2 → 1. The employee with employee_id 7 reports their work indirectly to the head of the company 7 → 4 → 2 → 1.

The employees with employee_id 3, 8, and 9 do not report their work to the head of the company directly or indirectly.

Solution:

```
with recursive new as
(
    select employee_id from Employees where employee_id = 1
    union
    select e2.employee_id from new e1
    inner join
        Employees e2
    on e1.employee_id = e2.manager_id
)
select * from new where employee_id <> 1;
```

The screenshot shows the MySQL Workbench interface with two tabs: 'create-db-template.sql' and 'Employees'.

The 'create-db-template.sql' tab contains the following SQL code:

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
1 create database test;
2 use test;
3 create table Employees
4 (employee_id Int primary key,
5 employee_name Varchar(15),
6 manager_id int
7 );
8 insert into Employees value(
9 | 1, 'Boss', 1),
10 | (3, 'Alice', 3),
11 | (2, 'Bob', 1),
12 | (4, 'Daniel', 2),
13 | (7, 'Luis', 4),
14 | (8, 'Jhon', 3),
15 | (9, 'Angela', 8),
16 | (77, 'Robert', 1);
17
18 with recursive new as
19 (
20     select employee_id from Employees where employee_id = 1
21     union
22     select e2.employee_id from new e1
23     inner join
24         Employees e2
25     on e1.employee_id = e2.manager_id
26 )
27 select * from new where employee_id <> 1;
28
```

The 'Employees' tab shows the data from the 'Employees' table:

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

Q72.

Table: Transactions

Column Name	Type
Id	Int
Country	Varchar
State	Enum
Amount	Int
trans_date	Date

id is the primary key of this table.

The table has information about incoming transactions.

The state column is an enum of type ["approved", "declined"].

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

Return the result table in any order.

The query result format is in the following example.

Input: Transactions

table:

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

Output:

month	Country	trans_count	approved_count	trans_total_amount	approved_total_amount
12	US	2	1	3000	1000
1	US	1	1	2000	2000
1	DE	1	1	2000	2000

Solution:

```
select month(trans_date) as Month,
       Country, count(Id) as trans_count,
       sum(case when State = 'approved' then 1 else 0 end) as approved_count,
       sum(amount) as trans_total_amount,
       sum(case when State = 'approved' then amount else 0 end) as
approved_total_amount
from Transactions
group by Month, Country;
```

```
create-db-template.sql X [Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
    ⚡ Active Connection | ▶ Execute
1  create database test;
    ▶ Execute
2  use test;
    ▶ Execute
3  create table Transactions
4      (Id Int primary key,
5       Country varchar(15),
6       State  varchar(15),
7       Amount  Int,
8       trans_date Date
9   );
    ▶ Execute
10 insert into Transactions values
11     (121, 'US', 'approved', 1000, '2018-12-18'),
12     (122, 'US', 'declined', 2000, '2018-12-19'),
13     (123, 'US', 'approved', 2000, '2019-01-01'),
14     (124, 'DE', 'approved', 2000, '2019-01-07');
    ▶ Execute
15 select month(trans_date) as Month,
16       Country, count(Id) as trans_count,
17       sum(case when State = 'approved' then 1 else 0 end) as approved_count,
18       sum(amount) as trans_total_amount,
19       sum(case when State = 'approved' then amount else 0 end) as approved_total_amount
20 from Transactions
21 group by Month, Country;
22
```

Transactions X

```
select month(trans_date) as Month,
       Country, count(Id) as trans_count

```

Free 1 + + + ↑ ↓ Cost: 3ms < 1 > Total 3

	Month	Country	trans_count	approved_count	trans_total_amount	approved_total_amount
	int	varchar	bigint	newdecimal	newdecimal	newdecimal
1	12	US	2	1	3000	1000
2	1	US	1	1	2000	2000
3	1	DE	1	1	2000	2000

Q73.

Table: Actions

Column Name	Type
user_id	Int
post_id	Int
action_date	Date
action	Enum
extra	Varchar

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

The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report', 'share').

The extra column has optional information about the action, such as a reason for the report or a type of reaction.

Table: Removals

Column Name	Type
post_id	Int
remove_date	Date

post_id is the primary key of this table.

Each row in this table indicates that some post was removed due to being reported or as a result of an admin review.

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

The query result format is in the following example.

Input:

Actions table:

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

5	2	2019-07-03	view	null
5	2	2019-07-03	report	racism
5	5	2019-07-03	view	null
5	5	2019-07-03	report	racism

Removals table:

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

Output:

average_daily_percent
75

Explanation:

The percentage for 2019-07-04 is 50% because only one post of two spam reported posts were removed.

The percentage for 2019-07-02 is 100% because one post was reported as spam and it was removed. The other days had no spam reports so the average is $(50 + 100) / 2 = 75\%$

Note that the output is only one number and that we do not care about the remove dates.

Solution:

```
select round(avg(t.daily_percent), 2) as average_daily_percent
from
(
    select
sum(case when remove_date > action_date then 1 else 0 end)/
count(tmp.action_date)*100 as daily_percent
from
(
    select post_id, action_date, extra
from Actions where extra = 'spam') tmp
left join Removals r
on tmp.post_id = r.post_id
group by action_date
) t;
```

create-db-template.sql X [Preview] README.md

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql >
    ↗ Active Connection | ⌂ Execute
1   create database test;
    ⌂ Execute
2   use test;
    ⌂ Execute
3   create table Actions
4     (user_id      Int,
5      post_id Int,
6      action_date Date,
7      action Varchar(10),
8      extra  varchar(10)
9    );
    ⌂ Execute
10  create table Removals
11    (post_id      int primary key,
12     remove_date date
13   );
    ⌂ Execute
14  insert into Actions values
15    (1, 1, '2019-07-01', 'view', null),
16    (1, 1, '2019-07-01', 'like', null),
17    (1, 1, '2019-07-01', 'share', null),
18    (2, 2, '2019-07-04', 'view', null),
19    (2, 2, '2019-07-04', 'report', 'spam'),
20    (3, 4, '2019-07-04', 'view', null),
21    (3, 4, '2019-07-04', 'report', 'spam'),
22    (4, 3, '2019-07-02', 'view', null),
23    (4, 3, '2019-07-02', 'report', 'spam'),
24    (5, 2, '2019-07-03', 'view', null),
25    (5, 2, '2019-07-03', 'report', 'racism'),
26    (5, 5, '2019-07-03', 'view', null),
27    (5, 5, '2019-07-03', 'report', 'racism');
    ⌂ Execute
28  insert into Removals values
29    (2, '2019-07-20'),
30    (3, '2019-07-18');
31
    ⌂ Execute
32  select round(avg(t.daily_percent), 2) as average_daily_percent
33  from
34  (
35    select
36      sum(case when remove_date > action_date then 1 else 0 end)/
37      count(tmp.action_date)*100 as daily_percent
38  from
39  (
40    select post_id, action_date, extra
41    from Actions where extra = 'spam') tmp
42  left join Removals r
43  on tmp.post_id = r.post_id
44  group by action_date
45  ) t;
46

```

Data X

```

select round(avg(t.daily_percent), 2) as average_daily_percent
from
    ↗ Input to filter result 1
    ↗ Average daily percent newdecimal
    ↗ Cost: 6ms < 1 > Total 1
    ↗ 1 75.00

```

Q74.

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 > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...

```
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;
23
```

activity X

select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction

from

Free 1

Input to filter result

Cost: 3ms < 1 > Total 1

	fraction	newdecimal
1	0.33	

Q75.

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 > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...

```
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;
23
```

activity X

```
select round(t.player_id/(select count(distinct player_id) from activity),2) as fraction
from
```

Free 1 ↕ + ⊕ ⊖ 🔍 Input to filter result Cost: 3ms < 1 > Total 1

✓ Q fraction newdecimal ↕

	1	0.33
--	---	------

Q76.

Table Salaries:

Column Name	Type
company_id	int
employee_id	int
employee_name	varchar
salary	int

(company_id, employee_id) is the primary key for this table.

This table contains the company id, the id, the name, and the salary for an employee.

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

The tax rate is calculated for each company based on the following criteria:

- 0% If the max salary of any employee in the company is less than \$1000.
- 24% If the max salary of any employee in the company is in the range [1000, 10000] inclusive.
- 49% If the max salary of any employee in the company is greater than \$10000.

Return the result table in any order.

The query result format is in the following example.

Input: Salaries

table:

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

Output:

company_id	employee_id	employee_name	Salary
1	1	Tony	1020
1	2	Pronub	10863
1	3	Tyrrox	5508
2	1	Pam	300
2	7	Bassem	450
2	9	Hermione	700
3	7	Bocaben	76
3	2	Ognjen	1672
3	13	Nyan Cat	2508
3	15	Morning Cat	5911

Explanation:

For company 1, Max salary is 21300. Employees in company 1 have taxes = 49%

For company 2, Max salary is 700. Employees in company 2 have taxes = 0%

For company 3, Max salary is 7777. Employees in company 3 have taxes = 24%

The salary after taxes = salary - (taxes percentage / 100) * salary

For example, Salary for Morning Cat (3, 15) after taxes = $7777 - 7777 * (24 / 100) = 7777 - 1866.48 = 5910.52$, which is rounded to 5911.

Solution:

```
select company_id, employee_id, employee_name,
(case when max(salary) over(partition by company_id) < 1000 then salary
      when max(salary) over(partition by company_id) < 10000 then
round(0.76*salary)
      else round(0.51*salary)
end) as Salary
from Salaries;
```

The screenshot shows a dark-themed interface of the MySQL extension in VS Code. At the top, there's a navigation bar with tabs for 'create-db-template.sql' (active), 'README.md', and other database connections. Below the navigation is a code editor window containing SQL code. The code creates a database named 'test', creates a table 'Salaries' with columns 'company_id', 'employee_id', 'employee_name', and 'salary', and inserts 15 rows of sample data. It then executes a query that uses a CASE WHEN statement with window functions to calculate a new salary column based on company_id and salary.

```
config > data > User > globalStorage > cwejan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Active Connection | ⌂ Execute
1  create database test;
2  use test;
3  create table Salaries
4  (
5    company_id int,
6    employee_id int,
7    employee_name varchar(15),
8    salary int,
9    primary key(company_id, employee_id)
10 );
11 insert into Salaries values
12 (1, 1, 'Tony', 2000),
13 (1, 2, 'Pronub', 21300),
14 (1, 3, 'Tyrrox', 10800),
15 (2, 1, 'Pam', 300),
16 (2, 7, 'Bassem', 450),
17 (2, 9, 'Hermione', 700),
18 (3, 7, 'Bocaben', 100),
19 (3, 2, 'Ognjen', 2200),
20 (3, 13, 'Nyan Cat', 3300),
21 (3, 15, 'Morning Cat', 7777);
22
23 select company_id, employee_id, employee_name,
24   (case when max(salary) over(partition by company_id) < 1000 then salary
25       when max(salary) over(partition by company_id) < 10000 then round(0.76*salary)
26       else round(0.51*salary)
27   end) as Salary
28 from Salaries;
29
```

Below the code editor is a results table titled 'Salaries'. The table has columns: company_id, employee_id, employee_name, and Salary. The data is as follows:

	company_id	employee_id	employee_name	Salary
1	1	1	Tony	1020
2	1	2	Pronub	10863
3	1	3	Tyrrox	5508
4	2	1	Pam	300
5	2	7	Bassem	450
6	2	9	Hermione	700
7	3	2	Ognjen	1672
8	3	7	Bocaben	76
9	3	13	Nyan Cat	2508
10	3	15	Morning Cat	5911

Q77.

Table Variables:

Column Name	Type
name	varchar
value	int

name is the primary key for this table.

This table contains the stored variables and their values.

Table Expressions:

Column Name	Type
left_operand	varchar
operator	enum
right_operand	varchar

(left_operand, operator, right_operand) is the primary key for this table.

This table contains a boolean expression that should be evaluated.

operator is an enum that takes one of the values ('<', '>', '=')

The values of left_operand and right_operand are guaranteed to be in the Variables table.

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

Return the result table in any order.

The query result format is in the following example.

Input: Variables

table:

name	value
x	66
y	77

Expressions table:

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

Output:

left_operand	operator	right_operand	value
x	>	y	false
x	<	y	true
x	=	y	false
y	>	x	true
y	<	x	false
x	=	x	true

Explanation:

As shown, you need to find the value of each boolean expression in the table using the variables table.

Solution:

```
select t.left_operand, t.operator, t.right_operand, (case
when t.value > v2.value and operator = '>' then "true"
when t.value < v2.value and operator = '<' then "true"
when t.value = v2.value and operator = '=' then "true"
else "false"
end) as value
from(select e.*, v1.value
from
Expressions e
inner join
Variables v1
on e.left_operand = v1.name) t
inner join
Variables v2
on t.right_operand = v2.name;
```

```

create-db-template.sql X
config > data > User > globalStorage > cwejan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
  8 create table Expressions(
  9   left_operand  varchar(5),
10   operator      varchar(5),
11   right_operand varchar(5)
12 );
13 insert into Variables values(
14   'x',    66),
15   ('y',    77
16 );
17 insert into Expressions values(
18   'x',    '>',    'y'),
19   ('x',    '<',    'y'),
20   ('x',    '=',    'y'),
21   ('y',    '>',    'x'),
22   ('y',    '<',    'x'),
23   ('x',    '=',    'x'
24 );
25 select t.left_operand, t.operator, t.right_operand, (case
26 when t.value > v2.value and operator = '>' then "true"
27 when t.value < v2.value and operator = '<' then "true"
28 when t.value = v2.value and operator = '=' then "true"
29 else "false"
30 end) as value
31 from(select e.* , v1.value
32 from
33 Expressions e
34 inner join
35 Variables v1
36 on e.left_operand = v1.name) t
37 inner join
38 Variables v2
39 on t.right_operand = v2.name;
40
41

```

	left_operand	operator	right_operand	value
1	x	>	y	false
2	x	<	y	true
3	x	=	y	false
4	y	>	x	true
5	y	<	x	false
6	x	=	x	true

Q78.

Table Person:

Column Name	Type
id	int
name	varchar
phone_number	varchar

id is the primary key for this table.

Each row of this table contains the name of a person and their phone number.

Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
name	varchar
country_code	varchar

country_code is the primary key for this table.

Each row of this table contains the country name and its code. country_code will be in the form 'xxx' where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
duration	int

There is no primary key for this table, it may contain duplicates.
Each row of this table contains the caller id, callee id and the duration of the call in minutes. $\text{caller_id} \neq \text{callee_id}$

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Person table:

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

Country table:

name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251

Calls table:

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

7	1	3
9	7	1
1	7	7

Output:

country
Peru

Explanation:

The average call duration for Peru is $(102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667$

The average call duration for Israel is $(33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500$

The average call duration for Morocco is $(33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000$

Global call duration average = $(2 * (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000$

Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

Solution:

```

select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl.callee_id as id, cl.duration
from Calls cl)) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;

```

SQLQuery2.sql - LAP...ARTA.test (sa (65)) * X SQLQuery1.sql - not connected

```

('Morocco', 212),
('Germany', 49),
('Ethiopia', 251);
insert into Calls values
(1, 9, 33),
(2, 9, 4),
(1, 2, 59),
(3, 12, 102),
(3, 12, 330),
(12, 3, 5),
(7, 9, 13),
(7, 1, 3),
(9, 7, 1),
(1, 7, 7);
select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl.callee_id as id, cl.duration
from Calls cl)) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;

```

00 %

Results Messages

Name
Peru

Query executed successfully.

Q79.

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

Level - Easy

Hint - Use ORDER BY

Input Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

Sample Input

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

Sample Output

Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd

Solution:

```
select name
from
Employee
order by name;
```

The screenshot shows a MySQL Workbench interface. The top part is a script editor with the file name 'create-db-template.sql'. It contains SQL code for creating a 'Employee' table and inserting 10 rows of data. The last row is a SELECT query. A green checkmark is next to the SELECT query, indicating it has been run successfully. The bottom part is a results grid titled 'Employee' showing the names of the employees. The results are:

	name
	varchar
1	Angela
2	Bonnie
3	Frank
4	Joe
5	Kimberly
6	Lisa
7	Michael
8	Patrick
9	Rose
10	Todd

Q80.

Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

Output the year (in ascending order) partitioned by product id, current year's spend, previous year's spend and year-on-year growth rate (percentage rounded to 2 decimal places).

Level - Hard

Hint - Use extract function

user_transactions Table:

Column Name	Type
transaction_id	integer
product_id	integer
spend	decimal
transaction_date	datetime

user_transactions Example Input:

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

Example Output:

y	product_id	curr_year_spend	prev_year_spend	yoy_rate
2	123424	1500.60		
2	123424	1000.20	1500.60	-33.35

2	123424	1246.44	1000.20	24.62
2	123424	2145.32	1246.44	72.12

Solution:

```

select year, product_id, curr_year_spend, coalesce(prev_year_spend, '') as
prev_year_spend,
coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100,2), '') as
yoY_rate
from
(
    select year(transaction_date) as year, product_id, spend as curr_year_spend,
round(lag(spend,1) over(partition by product_id order by transaction_date),2) as
prev_year_spend
from user_transactions
) t;

```

create-db-template.sql

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
  ✨ Active Connection | ▶ Execute
1   create database test;
  ▶ Execute
2   use test;
  ▶ Execute
3   create table user_transactions
4     (transaction_id Int,
5      product_id  Int,
6      Spend      float,
7      transaction_date DATETIME
8    );
  ▶ Execute
9   insert into user_transactions values
10  (1341, 123424, 1500.60, '2019/12/31 12:00:00'),
11  (1423, 123424, 1000.20, '2020/12/31 12:00:00'),
12  (1623, 123424, 1246.44, '2021/12/31 12:00:00'),
13  (1322, 123424, 2145.32, '2022/12/31 12:00:00');
  ▶ Execute
14  select year, product_id, curr_year_spend, coalesce(prev_year_spend,'') as prev_year_spend,
15  coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100,2),'') as yoy_rate
16  from
17  (
18    |  select year(transaction_date) as year, product_id, spend as curr_year_spend,
19    round(lag(spend,1) over(partition by product_id order by transaction_date),2) as prev_year_spend
20    from user_transactions
21  ) t;
  |

```

user_transactions

	year	product_id	curr_year_spend	prev_year_spend	yoy_rate
1	2019	123424	1500.6		
2	2020	123424	1000.2	1500.6	-33.35
3	2021	123424	1246.44	1000.2	24.62
4	2022	123424	2145.32	1246.44	72.12

Q81.

Amazon wants to maximise the number of items it can stock in a 500,000 square feet warehouse. It wants to stock as many prime items as possible, and afterwards use the remaining square footage to stock the most number of non-prime items.

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

Hint - create a table containing a summary of the necessary fields such as item type ('prime_eligible', 'not_prime'), SUM of square footage, and COUNT of items grouped by the item type.

inventory table:

Column Name	Type
item_id	integer
item_type	string
item_category	string
square_footage	decimal

inventory Example Input:

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

Example Output:

item_type	item_count
prime_eligible	9285
not_prime	6

Solution:

```
select item_type, (case
    when item_type = 'prime_eligible' then floor(500000/sum(square_footage)) *
count(item_type)
    when item_type = 'not_prime' then floor((500000 -(select
floor(500000/sum(square_footage)) * sum(square_footage) from inventory where
item_type = 'prime_eligible'))/sum(square_footage)) * count(item_type)
end) as item_count
from inventory
group by item_type
order by count(item_type) desc;
```

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 inventory
4  (
5      item_id int,
6      item_type varchar(20),
7      item_category varchar(20),
8      square_footage float
9  );
  > Execute
10 insert into inventory values
11 (1374, 'prime_eligible', 'mini refrigerator', 68.00),
12 (4245, 'not_prime', 'standing lamp', 26.40),
13 (2452, 'prime_eligible', 'television', 85.00),
14 (3255, 'not_prime', 'side table', 22.60),
15 (1672, 'prime_eligible', 'Laptop', 8.50);
  > Execute
16 select item_type, (case
17     when item_type = 'prime_eligible' then floor(500000/sum(square_footage)) * count(item_type)
18     when item_type = 'not_prime' then floor((500000 - [select floor(500000/sum(square_footage))
19     * sum(square_footage) from inventory where item_type = 'prime_eligible'])/sum(square_footage)) * count(item_type)
20 end) as item_count
21 from inventory
22 group by item_type
23 order by count(item_type) desc;
24
```

inventory X

select item_type, case
when item_tvne = 'prime_eligible' then floor(500000/sum(square_footage)) * count(item_tvne)

Free 1 Input to filter result

	item_type	item_count
1	prime_eligible	9285
2	not_prime	6

Q82.

Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month and last month.

Hint- Use generic correlated subquery

user_actions Table:

Column Name	Type
user_id	integer
event_id	integer
event_type	string ("sign-in", "like", "comment")
event_date	datetime

user_actionsExample Input:

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

Example Output for June 2022:

month	monthly_active_users
6	1

Solution: For July Month

```
select month(a.event_date) as month, count(distinct a.user_id) as
monthly_active_users
from
user_actions a
inner join
user_actions b
on concat(month(a.event_date),year(a.event_date)) =
concat(1+month(b.event_date),year(b.event_date))
and a.user_id = b.user_id
where a.event_type in ('sign-in', 'like', 'comment')
and b.event_type in ('sign-in', 'like', 'comment')
and concat(month(a.event_date), '/', year(a.event_date)) = '7/2022'
and concat(1+month(b.event_date), '/', year(b.event_date)) = '7/2022'
group by month(a.event_date);
```

Solution: For June Month

```
select month(a.event_date) as month, count(distinct a.user_id) as
monthly_active_users
from
user_actions a
inner join
user_actions b
on concat(month(a.event_date),year(a.event_date)) =
concat(1+month(b.event_date),year(b.event_date))
and a.user_id = b.user_id
where a.event_type in ('sign-in', 'like', 'comment')
and b.event_type in ('sign-in', 'like', 'comment')
and concat(month(a.event_date), '/', year(a.event_date)) = '6/2022'
and concat(1+month(b.event_date), '/', year(b.event_date)) = '6/2022'
group by month(a.event_date);
```

The screenshot shows a MySQL Workbench interface. The top part is a code editor with a dark theme containing SQL code for creating a table and inserting data. The bottom part is a results viewer titled "Data" showing the output of a query.

```

1 create-db-template.sql X
2 config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
3   ▷ Execute
4 2 use test;
5  ▷ Execute
6 3 create table user_actions
7 4 (user_id    Int,
8 5 event_id    Int,
9 6 event_type  varchar(10),
10 7 event_date  datetime
11 8 );
12  ▷ Execute
13 9 insert into user_actions values
14 10 (445,    7765,    'sign-in',    '2022/05/31 12:00:00'),
15 11 (742,    6458,    'sign-in',    '2022/06/03 12:00:00'),
16 12 (445,    3634,    'like',      '2022/06/05 12:00:00'),
17 13 (742,    1374,    'comment',   '2022/06/05 12:00:00'),
18 14 (648,    3124,    'like',      '2022/06/10 12:00:00');
19  ▷ Execute
20 15 select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users
21 16 from
22 17 user_actions a
23 18 inner join
24 19 user_actions b
25 20 on concat(month(a.event_date),year(a.event_date)) = concat(1+month(b.event_date),year(b.event_date))
26 21 and a.user_id = b.user_id
27 22 where a.event_type in ('sign-in', 'like', 'comment')
28 23 and b.event_type in ('sign-in', 'like', 'comment')
29 24 and concat(month(a.event_date), '/', year(a.event_date)) = '6/2022'
30 25 and concat(1+month(b.event_date), '/', year(b.event_date)) = '6/2022'
31 26 group by month(a.event_date);
32 27
33 28

```

Data

```

select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users
from user_actions a

```

Free 1

	month	monthly_active_users
<input checked="" type="checkbox"/>	int	bigint
1	6	1

Cost: 3ms < 1 > Total 1

Q83.

Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year.

However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the number of searches made last year and how many Google users fall into that bucket.

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

Hint- Write a subquery or common table expression (CTE) to generate a series of data (that's keyword for column) starting at the first search and ending at some point with an optional incremental value.

`search_frequency` Table:

Column Name	Type
searches	integer
num_users	integer

`search_frequency` Example Input:

searches	num_users
1	2
2	2
3	3
4	1

Example Output:

median
2.5

```
Solution: using recursive cte
with recursive seq as
(
    select searches, num_users, 1 as c from search_frequency
    union
    select searches, num_users, c+1 from seq where c < num_users
)
select round(avg(t.searches),1) as median from
(select searches, row_number() over(order by searches, c) as r1,
row_number() over(order by searches desc, c desc) as r2 from seq order by
searches) t
where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);
```

create-db-template.sql X [Preview] README.md

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

```
1      ▷ Execute
2  create database test;
3      ▷ Execute
4  use test;
5      ▷ Execute
6  create table search_frequency
7    (searches int,
8     num_users int);
9      ▷ Execute
10 insert into search_frequency values
11   (1, 2),
12   (2, 2),
13   (3, 3),
14   (4, 1);
15      ▷ Execute
16 ✓ 12 with recursive seq as
17  (
18    select searches, num_users, 1 as c from search_frequency
19    union
20    select searches, num_users, c+1 from seq where c < num_users
21  )
22  select round(avg(t.searches),1) as median from
23  (select searches, row_number() over(order by searches, c) as r1,
24   row_number() over(order by searches desc, c desc) as r2 from seq order by searches) t
25  where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);
26
```

Data X

with recursive seq as

Free 1 Input to filter result

Cost: 4ms < 1 Total 1

	median	newdecimal
	1	2.5

```

Solution: using cumulative sum
select round(avg(t1.searches),1) as median
from
(select t.searches, t.cumm_sum,
lag(cumm_sum,1,0) over(order by searches) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select searches, num_users,
sum(num_users) over(order by searches rows between unbounded preceding and current
row) as cumm_sum,
sum(num_users) over(order by searches rows between unbounded preceding and
unbounded following) as total
from search_frequency) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 >
t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);

```

The screenshot shows the MySQL Workbench interface. At the top, there's a breadcrumb navigation: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > Below this, the 'create-db-template.sql' tab is active, displaying the provided SQL code.

Below the code editor, the 'search_frequency' table is shown with the following data:

	searches	num_users
1	1	2
2	2	2
3	3	3
4	4	1

At the bottom, the results of the executed query are displayed:

	median
1	2.5

Below the results, there are various status indicators and performance metrics: Free, 1, Input to filter result, Cost: 7ms, Total 1.

Write a query to update the Facebook advertiser's status using the daily_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and daily_pay table has current information about their payment. Only advertisers who paid will show up in this table. Output the user id and current payment status sorted by the user id.

Hint- Query the daily_pay table and check through the advertisers in this table..

advertiser Table:

Column Name	Type
user_id	string
status	string

advertiser Example Input:

user_id	status
bing	NEW
yahoo	NEW
alibaba	EXISTING

daily_pay Table:

Column Name	Type
user_id	string
paid	decimal

daily_pay Example Input:

user_id	paid
yahoo	45.00

alibaba	100.00
target	13.00

Definition of advertiser status:

- New: users registered and made their first payment.
- Existing: users who paid previously and recently made a current payment.
- Churn: users who paid previously, but have yet to make any recent payment.
- Resurrect: users who did not pay recently but may have made a previous payment and have made payment again recently.

Example Output:

user_id	new_status
Bing	CHURN
Yahoo	EXISTING
alibaba	EXISTING

Bing's updated status is CHURN because no payment was made in the daily_pay table whereas Yahoo which made a payment is updated as EXISTING.

The dataset you are querying against may have different input & output - this is just an example!

Read this before proceeding to solve the question

For better understanding of the advertiser's status, we're sharing with you a table of possible transitions based on the payment status.

#	Start	End	Condition
1	NEW	EXISTING	Paid on day T
2	NEW	CHURN	No pay on day T
3	EXISTING	EXISTING	Paid on day T
4	EXISTING	CHURN	No pay on day T
5	CHURN	RESURRECT	Paid on day T
6	CHURN	CHURN	No pay on day T
7	RESURRECT	EXISTING	Paid on day T

8	RESURRECT	CHURN	No pay on day T
---	-----------	-------	-----------------

1. Row 2, 4, 6, 8: As long as the user has not paid on day T, the end status is updated to CHURN regardless of the previous status.
2. Row 1, 3, 5, 7: When the user paid on day T, the end status is updated to either EXISTING or RESURRECT, depending on their previous state. RESURRECT is only possible when the previous state is CHURN. When the previous state is anything else, the status is updated to EXISTING.

Solution:

Conditions used in case when:

Previous Status	Condition	Next Status
New, Existing, Churn, Resurrect	Didn't pay on day T	Churn
New, Existing, Resurrect	Paid on day T	Existing
Churn	Paid on day T	Resurrect

```
select user_id, case
when status in ('NEW','EXISTING','CHURN','RESURRECT') and user_id not in (select user_id from daily_pay) then 'CHURN'
when status in ('NEW','EXISTING','RESURRECT') and user_id in (select user_id from daily_pay) then 'EXISTING'
when status = 'CHURN' and user_id in (select user_id from daily_pay) then
'RESURRECT'
end as new_status
from advertiser
order by user_id;
```

```

create table advertiser
(user_id varchar(15),
status varchar(10));
insert into advertiser values
('Bing', 'NEW'),
('Yahoo', 'NEW'),
('Alibaba', 'EXISTING');

create table daily_pay
(user_id varchar(15),
paid float);
insert into daily_pay values
('Yahoo', 45.00),
('Alibaba', 100.00),
('Target', 13.00);

select user_id, case
when status in ('NEW', 'EXISTING', 'CHURN', 'RESURRECT') and user_id not in (select user_id from daily_pay) then 'CHURN'
when status in ('NEW', 'EXISTING', 'RESURRECT') and user_id in (select user_id from daily_pay) then 'EXISTING'
when status = 'CHURN' and user_id in (select user_id from daily_pay) then 'RESURRECT'
end as new_status
from advertiser
order by user_id;

```

	user_id	new_status
1	Alibaba	EXISTING
2	Bing	CHURN
3	Yahoo	EXISTING

Q85.

Amazon Web Services (AWS) is powered by fleets of servers. Senior management has requested data-driven solutions to optimise server usage.

Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

Level - Hard

Hint-

1. Calculate individual uptimes

2. Sum those up to obtain the uptime of the whole fleet, keeping in mind that the result must be output in units of full days

Assumptions:

- Each server might start and stop several times.
- The total time in which the server fleet is running can be calculated as the sum of each server's uptime.

server_utilization Table:

Column Name	Type
server_id	Integer
status_time	timestamp
session_status	String

server_utilization Example Input:

server_id	status_time	session_status
1	08/02/2022 10:00:00	Start
1	08/04/2022 10:00:00	Stop
2	08/17/2022 10:00:00	Start
2	08/24/2022 10:00:00	stop

Solution:

```
select sum(t.individual_uptime) as total_uptime_days
from
(
    select case when session_status = 'stop'
then
timestampdiff(day, lag(status_time) over(partition by server_id order by
status_time), status_time) end as individual_uptime
from server_utilization
) t;
```

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

☆ Active Connection | ▷ Execute

```
1 create database test;
  ▷ Execute
2 use test;
  ▷ Execute
3 create table server_utilization
4 (server_id Int,
5 status_time timestamp,
6 session_status varchar(10)
7 );
  ▷ Execute
8 insert into server_utilization values
9 (1, '2022/08/02 10:00:00', 'start'),
10 (1, '2022/08/04 10:00:00', 'stop'),
11 (2, '2022/08/17 10:00:00', 'start'),
12 (2, '2022/08/24 10:00:00', 'stop');
  ▷ Execute
13 select sum(t.individual_uptime) as total_uptime_days
14 from
15 (
16   | select case when session_status = 'stop'
17   then
18     timestampdiff(day, lag(status_time) over(partition by server_id order by status_time), status_time) end as individual_uptime
19   from server_utilization
20 ) t;
21
```

```
server_utilization X

select sum(individual_uptime) as total_uptime_days
from
    
Free 1
total_uptime_days newdecimal
     Q

|   |   |
|---|---|
| 1 | 9 |
|---|---|

Cost: 7ms < 1 > Total 1
```

Q86.

Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or a retry error that causes a credit card to be charged twice.

Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other. Count such repeated payments.

Level - Hard

Hint- Use Partition and order by

Assumptions:

- The first transaction of such payments should not be counted as a repeated payment. This means, if there are two transactions performed by a merchant with the same credit card and for the same amount within 10 minutes, there will only be 1 repeated payment.

transactions Table:

Column Name	Type
transaction_id	integer
merchant_id	integer
credit_card_id	integer
amount	integer

transaction_timestamp	datetime
-----------------------	----------

transactions Example Input:

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

Example Output:

payment_count
1

Solution:

```

select sum(case when (unix_timestamp(t.next_transaction) -
unix_timestamp(t.transaction_timestamp))/60 <= 10 then 1 else 0 end) as
payment_count
from
(select transaction_timestamp,
lead(transaction_timestamp,1) over(partition by merchant_id, credit_card_id,
Amount order by transaction_timestamp) as next_transaction
from transactions)t;

```

The screenshot shows a MySQL Workbench interface. The top part is a query editor with the following SQL code:

```

1 config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
2 use test;
3 create table transactions
4 (transaction_id Int,
5 merchant_id Int,
6 credit_card_id Int,
7 Amount Int,
8 transaction_timestamp datetime
9 );
10 insert into transactions values
11 (1, 101, 1, 100, '2022/09/25 12:00:00'),
12 (2, 101, 1, 100, '2022/09/25 12:08:00'),
13 (3, 101, 1, 100, '2022/09/25 12:28:00'),
14 (4, 102, 2, 300, '2022/09/25 12:00:00'),
15 (6, 102, 2, 400, '2022/09/25 14:00:00');
16
17 select sum(case when (unix_timestamp(t.next_transaction) - unix_timestamp(t.transaction_timestamp))/60 <= 10 then 1 else 0 end) as payment_count
18 from
19 (select transaction_timestamp,
20 lead(transaction_timestamp,1) over(partition by merchant_id, credit_card_id, Amount order by transaction_timestamp) as next_transaction
21 from transactions)t;
22
23
24

```

The bottom part is a results grid titled "transactions" showing one row of data:

	payment_count
	1

Q87.

DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because:

- the orders are being completed incorrectly (missing items, wrong order, etc.)
- the orders aren't being received (wrong address, wrong drop off spot)
- the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated_delivery_timestamp is automatically set to 30 minutes after the order_timestamp.

Hint- Use Where Clause and joins

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

orders Table:

Column Name	Type
order_id	Integer

customer_id	Integer
trip_id	Integer
status	string ('completed successfully', 'completed incorrectly', 'never received')
order_timestamp	Timestamp

orders Example Input:

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

trips Table:

Column Name	Type
dasher_id	integer
trip_id	integer
estimated_delivery_timestamp	timestamp
actual_delivery_timestamp	timestamp

trips Example Input:

dasher_id	trip_id	estimated_delivery_timestamp	actual_delivery_timestamp
101	100463	06/05/2022 09:42:00	06/05/2022 09:38:00
102	100482	06/05/2022 15:10:00	06/05/2022 15:46:00

101	100362	06/07/2022 15:33:00	06/07/2022 16:45:00
102	100657	07/07/2022 15:52:00	-
103	100213	06/12/2022 14:13:00	06/12/2022 14:10:00

customers Table:

Column Name	Type
customer_id	integer
signup_timestamp	timestamp

customers Example Input:

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

Example Output:

bad_experience_pct
75.00

Solution:

```
select round(avg(t1.bad_exp_pct_per_cust),2) as bad_exp_pct
from
(
    select t.customer_id, 100*sum(case when o.status <> 'completed successfully'
then 1 else 0 end)/count(*) as bad_exp_pct_per_cust
    from
    (
        select customer_id, signup_timestamp from customers where
month(signup_timestamp) = 6
    ) t
    inner join
    orders o
    on o.customer_id = t.customer_id
    where timestampdiff(day, t.signup_timestamp, o.order_timestamp) <= 13
    group by t.customer_id
) t1;
```


Q88

Table: Scores

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day) is the primary key for this table.

A competition is held between the female team and the male team.

Each row of this table indicates that a player_name and with gender has scored score_point in someday.

Gender is 'F' if the player is in the female team and 'M' if the player is in the male team.

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

Return the result table ordered by gender and day in ascending order.

The query result format is in the following example.

Input:

Scores table:

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

Output:

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

M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

Explanation:

For the female team:

The first day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.

The second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.

The third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

The fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For the male team:

The first day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

The second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.

The third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.

The fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

The fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

Solution:

```
select gender, day,
sum(score_points) over(partition by gender order by day) as total
from Scores
group by gender, day
order by gender, day;
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Execute
3   create table Scores
4   (
5     player_name varchar(15),
6     gender varchar(2),
7     day date,
8     score_points int,
9     primary key(gender, day)
10  );
  ↗ Execute
11  insert into Scores values
12  ('Aron', 'F', '2020-01-01', 17),
13  ('Alice', 'F', '2020-01-07', 23),
14  ('Bajrang', 'M', '2020-01-07', 7),
15  ('Khali', 'M', '2019-12-25', 11),
16  ('Slaman', 'M', '2019-12-30', 13),
17  ('Joe', 'M', '2019-12-31', 3),
18  ('Jose', 'M', '2019-12-18', 2),
19  ('Priya', 'F', '2019-12-31', 23),
20  ('Priyanka', 'F', '2019-12-30', 17);
  ↗ Execute
21  select gender, day, sum(score_points) over(partition by gender order by day) as total
22  from Scores
23  group by gender, day
24  order by gender, day;

```

Scores ×

```

select gender, day, sum(score_points) over(partition by gender order by day) as total
from Scores

Free 1 ✉️ ➕ ✖️ ✖️ ⬆️ ⬇️ ▶️ Cost: 4ms < 1 > Total 9

```

	gender	day	total
	varchar	date	newdecimal
1	F	2019-12-30	17
2	F	2019-12-31	40
3	F	2020-01-01	57
4	F	2020-01-07	80
5	M	2019-12-18	2
6	M	2019-12-25	13
7	M	2019-12-30	26
8	M	2019-12-31	29
9	M	2020-01-07	36

Q89.

Table Person:

Column Name	Type
id	int
name	varchar
phone_number	varchar

id is the primary key for this table.

Each row of this table contains the name of a person and their phone number.

Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
name	varchar
country_code	varchar

country_code is the primary key for this table.

Each row of this table contains the country name and its code. country_code will be in the form 'xxx'

where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
duration	int

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

Each row of this table contains the caller id, callee id and the duration of the call in minutes. $\text{caller_id} \neq \text{callee_id}$

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Person table:

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

Country table:

name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251
Ethiopia	251

Calls table:

caller_id	callee_id	duration
1	9	33
2	9	4

1	2	59
3	12	102
3	12	330
12	3	5
7	9	13
7	1	3
9	7	1
1	7	7

Output:

country
Peru

Explanation:

The average call duration for Peru is $(102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667$

The average call duration for Israel is $(33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500$

The average call duration for Morocco is $(33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000$

Global call duration average = $(2 * (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000$

Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

Solution:

```

select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl.callee_id as id, cl.duration
from Calls cl)) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;

```

SQLQuery2.sql - LAP...ARTA.test (sa (65))*

```

('Morocco', 212),
('Germany', 49),
('Ethiopia', 251);
insert into Calls values
(1, 9, 33),
(2, 9, 4),
(1, 2, 59),
(3, 12, 102),
(3, 12, 330),
(12, 3, 5),
(7, 9, 13),
(7, 1, 3),
(9, 7, 1),
(1, 7, 7);
select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl.callee_id as id, cl.duration
from Calls cl))) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;

```

00 %

Results Messages

Name
Peru

Query executed successfully.

Q90.

Table: Numbers

Column Name	Type
num	int
frequency	int

num is the primary key for this table.

Each row of this table shows the frequency of a number in the database.

The median is the value separating the higher half from the lower half of a data sample.

Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point.

The query result format is in the following example.

num	frequency
0	7
1	1
2	3
3	1

Output:

median
0

Explanation:

If we decompose the Numbers table, we will get [0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3], so the median is $(0 + 0) / 2 = 0$.

```

Solution: using recursive cte
with recursive seq as
(
    select num, frequency, 1 as c from Numbers
    union
    select num, frequency, c+1 from seq where c < frequency
)
select round(avg(t.num),1) as median
from
(
    select num, row_number() over(order by num, c) as r1,
    row_number() over(order by num desc, c desc) as r2 from seq order by num
) t
where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);

```

create-db-template.sql X [Preview] README.md

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

```

1      ▷ Execute
2  create database test;
3      ▷ Execute
4  use test;
5      ▷ Execute
6  create table Numbers
7    (num int primary key,
8     frequency int);
9      ▷ Execute
10 insert into Numbers values
11   (0, 7),
12   (1, 1),
13   (2, 3),
14   (3, 1);
15      ▷ Execute
16 with recursive seq as
17 (
18   select num, frequency, 1 as c from Numbers
19   union
20   select num, frequency, c+1 from seq where c < frequency
21 )
22 select round(avg(t.num),1) as median
23 from
24 (
25   select num, row_number() over(order by num, c) as r1,
26         row_number() over(order by num desc, c desc) as r2 from seq order by num
27 ) t
28 where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);
29

```

Data X

with recursive seq as

<input checked="" type="checkbox"/>	<input type="checkbox"/>	Input to filter result
		Cost: 2ms < 1 > Total 1

<input checked="" type="checkbox"/>	<input type="checkbox"/>	median newdecimal
1	0.0	

Solution: using cumulative sum

```

select round(avg(t1.num),1) as median
from
(select t.num, t.cumm_sum,
lag(cumm_sum,1,0) over(order by num) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select num, frequency,
sum(frequency) over(order by num rows between unbounded preceding and current row)
as cumm_sum,
sum(frequency) over(order by num rows between unbounded preceding and unbounded
following) as total

```

```

from Numbers) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 >
t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);

```

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 Numbers
4 (num Int,
5 frequency Int,
6 primary key(num)
7);
 ▷ Execute
8 insert into Numbers values
9 (0, 7),
10 (1, 1),
11 (2, 3),
12 (3, 1);
13
 ▷ Execute
✓ 14 select round(avg(t1.num),1) as median
15 from
16 (select t.num, t.cumm_sum,
17 lag(cumm_sum,1,0) over(order by num) as prev_cumm_sum,
18 case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
19 case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
20 from
21 (select num, frequency,
22 sum(frequency) over(order by num rows between unbounded preceding and current row) as cumm_sum,
23 sum(frequency) over(order by num rows between unbounded preceding and unbounded following) as total
24 from Numbers) t
25) t1
26 where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 > t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
27

Numbers X

select round(avg(t1.num),1) as median
from
 Free 1 Cost: 5ms < 1 Total 1

	median	newdecimal
<input checked="" type="checkbox"/>	1	0.0

Q91.

Table: Salary

Column Name	Type
id	int
employee_id	int
amount	int
pay_date	date

id is the primary key column for this table.

Each row of this table indicates the salary of an employee in one month.

employee_id is a foreign key from the Employee table.

Table: Employee

Column Name	Type
employee_id	int
department_id	int

employee_id is the primary key column for this table.

Each row of this table indicates the department of an employee.

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

Return the result table in any order.

The query result format is in the following example.

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

Employee table:

employee_id	department_id
1	1
2	2
3	2

Output:

pay_month	department_id	comparison
2017-02	1	same
2017-03	1	higher
2017-02	2	same
2017-03	2	lower

Explanation:

In March, the company's average salary is $(9000+6000+10000)/3 = 8333.33\dots$

The average salary for department '1' is 9000, which is the salary of employee_id '1' since there is only one employee in this department. So the comparison result is 'higher' since $9000 > 8333.33$ obviously.

The average salary of department '2' is $(6000 + 10000)/2 = 8000$, which is the average of employee_id '2' and '3'. So the comparison result is 'lower' since $8000 < 8333.33$.

With the same formula for the average salary comparison in February, the result is 'same' since both the departments '1' and '2' have the same average salary with the company, which is 7000.

Solution:

```
select distinct concat(year(t.pay_date), '-', month(t.pay_date)) as pay_month,
t.department_id,
case
when monthly_department_avg_salary > monthly_average_salary then 'higher'
when monthly_department_avg_salary < monthly_average_salary then 'lower'
else 'same'
end as Comparison
from
(select s.pay_date, e.department_id,
avg(s.amount) over(partition by month(s.pay_date), e.department_id) as
monthly_department_avg_salary,
avg(s.amount) over(partition by month(s.pay_date)) as monthly_average_salary
from Salary s
left join
Employee e
on s.employee_id = e.employee_id) t
order by t.department_id;
```

The screenshot shows a MySQL client interface with a code editor and a results table.

Code Editor (create-db-template.sql):

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
10  create table Employee
11  (employee_id  Int,
12  department_id  Int,
13  primary key(employee_id)
14 );
15  insert into Salary values
16  (1, 1, 9000, '2017/03/31'),
17  (2, 2, 6000, '2017/03/31'),
18  (3, 3, 10000, '2017/03/31'),
19  (4, 1, 7000, '2017/02/28'),
20  (5, 2, 6000, '2017/02/28'),
21  (6, 3, 8000, '2017/02/28');
22  insert into Employee values
23  (1, 1),
24  (2, 2),
25  (3, 2);
26  select distinct concat(year(t.pay_date), '-', month(t.pay_date)) as pay_month,
27  t.department_id,
28  case
29  when monthly_department_avg_salary > monthly_average_salary then 'higher'
30  when monthly_department_avg_salary < monthly_average_salary then 'lower'
31  else 'same'
32  end as Comparison
33  from
34  (select s.pay_date, e.department_id,
35  avg(s.amount) over(partition by month(s.pay_date), e.department_id) as monthly_department_avg_salary,
36  avg(s.amount) over(partition by month(s.pay_date)) as monthly_average_salary
37  from Salary s
38  left join
39  Employee e
40  on s.employee_id = e.employee_id) t
41  order by t.department_id;
42
```

Results Table:

	pay_month	department_id	Comparison
1	2017-2	1	same
2	2017-3	1	higher
3	2017-2	2	same
4	2017-3	2	lower

Q92

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.

The install date of a player is the first login day of that player.

We define day one retention of some date x to be the number of players whose install date is x and they logged back in on the day right after x, divided by the number of players whose install date is x, rounded to 2 decimal places.

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

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	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-01	0
3	4	2016-07-03	5

Output:

install_dt	installs	Day1_retention
2016-03-01	2	0.5
2017-06-25	1	0

Explanation:

Player 1 and 3 installed the game on 2016-03-01 but only player 1 logged back in on 2016-03-02 so the day 1 retention of 2016-03-01 is $1 / 2 = 0.50$

Player 2 installed the game on 2017-06-25 but didn't log back in on 2017-06-26 so the day 1 retention of 2017-06-25 is $0 / 1 = 0.00$

Solution:

```
select t1.install_dt, count(player_id) as installs,
round(count(t1.next_install)/count(t1.player_id),2) as Day1_retention
from
(
    select t.player_id, t.install_dt, a.event_date as next_install
    from
        (
            select player_id, min(event_date) as install_dt
            from Activity
            group by player_id
        ) t
        left join
        Activity a
        on t.player_id = a.player_id and a.event_date = t.install_dt + 1
) t1
group by install_dt;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** The code block above is pasted into the query editor.
- Execution Result:** The result set is displayed in a table:

install_dt	installs	Day1_retention
2016-03-01	2	0.50
2017-06-25	1	0.00
- Performance Metrics:** The execution cost is shown as 6ms.
- Session Information:** The session path is visible at the top: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

Q93.

Table: Players

Column Name	Type
player_id	Int
group_id	Int

player_id is the primary key of this table.

Each row of this table indicates the group of each player.

Table: Matches

Column Name	Type
match_id	Int
first_player	Int
second_player	Int
first_score	Int
second_score	Int

match_id is the primary key of this table.

Each row is a record of a match, first_player and second_player contain the player_id of each match. first_score and second_score contain the number of points of the first_player and second_player respectively.

You may assume that, in each match, players belong to the same group.

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;

```

```

create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
    ▷ Execute
4  use test;
    ▷ Execute
5  create table Players
6  (
7  player_id  Int primary key,
8  group_id   Int
9  );
    ▷ Execute
10 create table Matches
11 (
12     match_id   Int primary key,
13     first_player  Int,
14     second_player Int,
15     first_score  Int,
16     second_score Int
17 );
    ▷ Execute
18 insert into Players values
19 (15,      '1'),
20 (25,      '1'),
21 (30,      '1'),
22 (45,      '1'),
23 (10,      '2'),
24 (35,      '2'),
25 (50,      '2'),
26 (20,      '3'),
27 (40,      '3');
    ▷ Execute
28 insert into Matches values
29 (1, 15, 45, 3, 0),
30 (2, 30, 25, 1, 2),
31 (3, 30, 15, 2, 0),
32 (4, 40, 20, 5, 2),
33 (5, 35, 50, 1, 1);
    ▷ Execute
✓ 34 select t2.group_id, t2.player_id from
35 (
36     select t1.group_id, t1.player_id,
37     dense_rank() over(partition by group_id order by score desc, player_id) as r
38     from
39     (
40         select p.*, case when p.player_id = m.first_player then m.first_score
41         when p.player_id = m.second_player then m.second_score
42         end as score
43         from
44         Players p, Matches m
45         where player_id in (first_player, second_player)
46         | t1
47     ) t2
48     where r = 1;

```

Players ✘

```

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;

```

Free 1

Input to filter result

Cost 12ms < 1 Total 3

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

Q94.

Table: Student

Column Name	Type
student_id	Int
student_name	varchar

student_id is the primary key for this table. student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	Int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.

Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam.

Return the result table ordered by student_id.

The query result format is in the following example.

Input:

Student table:

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

Exam table:

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

Output:

student_id	student_name
2	Jade

Explanation:

For exam 1: Student 1 and 3 hold the lowest and high scores respectively.

For exam 2: Student 1 holds both the highest and lowest score.

For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively.

Students 2 and 5 have never got the highest or lowest in any of the exams.

Since student 5 is not taking any exam, he is excluded from the result. So, we only return the information of Student 2.

Solution:

```
select t.student_id, t.student_name from
(select s.student_name, s.student_id, count(e.student_id) over(partition by
student_name) as exams_given,
case when e.score > min(e.score) over(partition by e.exam_id) and e.score <
max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
# 1 means student is quiet, 0 means student is not quiet
from Exam e
left join
Student s
on e.student_id = s.student_id)t
group by t.student_name, t.student_id, t.exams_given
having sum(t.quiet) = t.exams_given
# sum(quiet) will give the total number of exams in which student is quiet
```

create-db-template.sql

```
config > data > User > globalStorage > cwejan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
5  student_name      varchar(15),
6  primary key(student_id)
7  );
   > Execute
8  create table Exam
9  (exam_id    int,
10 student_id  int,
11 score      int,
12 primary key(exam_id, student_id)
13 );
   > Execute
14 insert into Student values
15 (1, 'Daniel'),
16 (2, 'Jade'),
17 (3, 'Stella'),
18 (4, 'Jonathan'),
19 (5, 'Will');
   > Execute
20 insert into Exam values
21 (10, 1, 70),
22 (10, 2, 80),
23 (10, 3, 90),
24 (20, 1, 80),
25 (30, 1, 70),
26 (30, 3, 80),
27 (30, 4, 90),
28 (40, 1, 60),
29 (40, 2, 70),
30 (40, 4, 80);
   > Execute
✓ 31 select t.student_id, t.student_name from
32 (select s.student_name, s.student_id, count(e.student_id) over(partition by student_name) as exams_given,
33 case when e.score > min(e.score) over(partition by e.exam_id) and e.score < max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
34 # 1 means student is quiet, 0 means student is not quiet
35 from Exam e
36 left join
37 Student s
38 on e.student_id = s.student_id
39 group by t.student_name, t.student_id, t.exams_given
40 having sum(t.quiet) = t.exams_given
41 # sum(quiet) will give the total number of exams in which student is quiet
42 ;
```

Data

student_id	student_name
1	Jade

Q95.

Table: Student

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table. student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.

Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

A quiet student is the one who took at least one exam and did not score the high or the low score. Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam.

Return the result table ordered by student_id.

The query result format is in the following example.

Input: Student

table:

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

Exam table:

exam_id	student_id	score
10	1	70
10	2	80
10	3	90
20	1	80
30	1	70

30	3	80
30	4	90
40	1	60
40	2	70
40	4	80

Output:

student_id	student_name
2	Jade

Explanation:

For exam 1: Student 1 and 3 hold the lowest and high scores respectively.

For exam 2: Student 1 holds both the highest and lowest score.

For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively.

Students 2 and 5 have never got the highest or lowest in any of the exams.

Since student 5 is not taking any exam, he is excluded from the result.

So, we only return the information of Student 2.

Solution:

```

select t.student_id, t.student_name from
(select s.student_name, s.student_id, count(e.student_id) over(partition by
student_name) as exams_given,
case when e.score > min(e.score) over(partition by e.exam_id) and e.score <
max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
# 1 means student is quiet, 0 means student is not quiet
from Exam e
left join
Student s
on e.student_id = s.student_id)t
group by t.student_name, t.student_id, t.exams_given
having sum(t.quiet) = t.exams_given
# sum(quiet) will give the total number of exams in which student is quiet

```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
5   student_name  varchar(15),
6   primary key(student_id)
7 );
    ▷ Execute
8   create table Exam
9   (exam_id  int,
10  student_id  int,
11  score  int,
12  primary key(exam_id, student_id)
13 );
    ▷ Execute
14  insert into Student values
15  (1, 'Daniel'),
16  (2, 'Jade'),
17  (3, 'Stella'),
18  (4, 'Jonathan'),
19  (5, 'Will');
    ▷ Execute
20  insert into Exam values
21  (10,  1,  70),
22  (10,  2,  80),
23  (10,  3,  90),
24  (20,  1,  80),
25  (30,  1,  70),
26  (30,  3,  80),
27  (30,  4,  90),
28  (40,  1,  60),
29  (40,  2,  70),
30  (40,  4,  80);
    ▷ Execute
31  select t.student_id, t.student_name from
32  (select s.student_name, s.student_id, count(e.student_id) over(partition by student_name) as exams_given,
33  case when e.score > min(e.score) over(partition by e.exam_id) and e.score < max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
34  # 1 means student is quiet, 0 means student is not quiet
35  from Exam e
36  left join
37  Student s
38  on e.student_id = s.student_id)t
39  group by t.student_name, t.student_id, t.exams_given
40  having sum(t.quiet) = t.exams_given
41  # sum(quiet) will give the total number of exams in which student is quiet
42 ;

```

Data

student_id	student_name	
1	Jade	

Q96.

You're given two tables on Spotify users' streaming data. songs_history table contains the historical streaming data and songs_weekly table contains the current week's streaming data.

Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022 sorted in descending order.

Hint- Use group by

Definitions:

- song_weekly table currently holds data from 1 August 2022 to 7 August 2022.

- songs_history table currently holds data up to 31 July 2022. The output should include the historical data in this table.

Assumption:

- There may be a new user or song in the songs_weekly table not present in the songs_history table.

songs_history Table:

Column Name	Type
history_id	Integer
user_id	Integer
song_id	Integer
song_plays	Integer

songs_history Example Input:

history_id	user_id	song_id	song_plays
10011	777	1238	11
12452	695	4520	1

song_plays: Refers to the historical count of streaming or song plays by the user.

songs_weekly Table:

Column Name	Type
user_id	Integer
song_id	Integer
listen_time	Datetime

songs_weekly Example Input:

user_id	song_id	listen_time
777	1238	08/01/2022 12:00:00
695	4520	08/04/2022 08:00:00

125	9630	08/04/2022 16:00:00
695	9852	08/07/2022 12:00:00

user_id	song_id	song_plays
777	1238	12
695	4520	2
125	9630	1

Solution:

```
select t.user_id, t.song_id, sum(t.song_plays) as song_plays
from
(
    select user_id, song_id, song_plays
from songs_history
union all
select user_id, song_id, 1 as song_plays
from songs_weekly
where date(listen_time) <= '2022/08/04') t
group by user_id, song_id;
```

create-db-template.sql

```
2  create table songs_history
3  (history_id Int,
4  user_id Int,
5  song_id Int,
6  song_plays Int
7  );
8  ▷ Execute
9  create table songs_weekly
10 (
11 | user_id Int,
12 | song_id Int,
13 | listen_time Datetime
14 );
15 ▷ Execute
16 insert into songs_history values
17 (10011, 777, 1238, 11),
18 (12452, 695, 4520, 1);
19 ▷ Execute
20 insert into songs_weekly values
21 (777, 1238, '2022/08/01 12:00:00'),
22 (695, 4520, '2022/08/04 08:00:00'),
23 (125, 9630, '2022/08/04 16:00:00'),
24 (695, 9852, '2022/08/07 12:00:00');
25 ▷ Execute
26 select t.user_id, t.song_id, sum(t.song_plays) as song_plays
27 from
28 (
29 |     select user_id, song_id, song_plays
30 from songs_history
31 union all
32 |     select user_id, song_id, 1 as song_plays
33 from songs_weekly
34 where date(listen_time) <= '2022/08/04' t
35 group by user_id, song_id;
36 
```

songs_history

```
select t.user_id, t.song_id, sum(t.song_plays) as song_plays
from
```

Input to filter result

Free 1

	user_id	song_id	song_plays
1	777	1238	12
2	695	4520	2
3	125	9630	1

Cost: 3ms

Q97.

New TikTok users sign up with their emails, so each signup requires a text confirmation to activate the new user's account.

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

Hint- Use Joins

Assumptions:

- A user may fail to confirm several times with text. Once the signup is confirmed for a user, they will not be able to initiate the signup again.
- A user may not initiate the signup confirmation process at all.

emails Table:

Column Name	Type
email_id	Integer
user_id	Integer
signup_date	Datetime

emails Example Input:

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

Texts Table:

Column Name	Type
text_id	Integer
email_id	Integer
signup_action	varchar

texts Example Input:

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

Example Output:

confirm_rate
0.67

Solution:

```
select round(sum(case when t.signup_action = 'Confirmed' then 1 else 0
end)/count(*),2) as confirm_rate
from
emails e
join
texts t
on e.email_id = t.email_id;
```

```

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 | D Execute
1 use test;
D Execute
2 create table emails
3 (email_id Int,
4 user_id Int,
5 signup_date Datetime
6 );
D Execute
7 create table texts
8 (text_id Int,
9 email_id Int,
10 signup_action varchar(30)
11 );
D Execute
12 insert into emails values
13 (125, 7771, '2022/06/14 00:00:00'),
14 (236, 6950, '2022/07/01 00:00:00'),
15 (433, 1052, '2022/07/09 00:00:00');
D Execute
16 insert into texts values
17 (6878, 125, 'Confirmed'),
18 (6920, 236, 'Not Confirmed'),
19 (6994, 236, 'Confirmed');
D Execute
20 select round(sum(case when t.signup_action = 'Confirmed' then 1 else 0 end)/count(*),2) as confirm_rate
21 from
22 emails e
23 join
24 texts t
25 on e.email_id = t.email_id;
26
27
songs_history X
select round(sum(case when t.signup_action = 'Confirmed' then 1 else 0 end)/count(*),2) as
confirm_rate
Free 1 Cost: 3ms < 1 > Total 1
Input to filter result
confirm_rate newdecimal
1 0.67

```

Q98.

The table below contains information about tweets over a given period of time. Calculate the 3-day rolling average of tweets published by each user for each date that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.

Hint- Use Count and group by

Important Assumptions:

- Rows in this table are consecutive and ordered by date.
- Each row represents a different day
- A day that does not correspond to a row in this table is not counted. The most recent day is the next row above the current row.

Note: Rolling average is a metric that helps us analyze data points by creating a series of averages based on different subsets of a dataset. It is also known as a moving average, running average, moving mean, or rolling mean.

tweets Table:

Column Name	Type
tweet_id	Integer
user_id	Integer
tweet_date	Timestamp

tweets Example Input:

tweet_id	user_id	tweet_date
214252	111	06/01/2022 12:00:00
739252	111	06/01/2022 12:00:00
846402	111	06/02/2022 12:00:00
241425	254	06/02/2022 12:00:00
137374	111	06/04/2022 12:00:00

Example Output:

user_id	tweet_date	rolling_avg_3days
111	06/01/2022 12:00:00	2.00
111	06/02/2022 12:00:00	1.50
111	06/04/2022 12:00:00	1.33
254	06/02/2022 12:00:00	1.00

Solution:

```
select user_id, date_format(tweet_date, '%m/%d/%Y %h.%i:%s') as tweet_date,
round(avg(count(distinct tweet_id)) over(order by tweet_date rows between 2 preceding and current row),2) as rolling_avg_3days
from tweets
group by user_id, tweet_date
```

create-db-template.sql

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template
  Active Connection | Execute
1  create database test;
  Execute
2  use test;
  Execute
3  create table tweets
4    (tweet_id  Int,
5     user_id  Int,
6     tweet_date  Timestamp
7   );
  Execute
8  insert into tweets values
9    (214252,      111,      '2022/06/01 12:00:00'),
10   (739252,      111,      '2022/06/01 12:00:00'),
11   (846402,      111,      '2022/06/02 12:00:00'),
12   (241425,      254,      '2022/06/02 12:00:00'),
13   (137374,      111,      '2022/06/04 12:00:00');
  Execute
14  select user_id, date_format(tweet_date, '%m/%d/%Y %h:%i:%s') as tweet_date,
15  round(avg(count(distinct tweet_id))
16 over(order by tweet_date rows between 2 preceding and current row),2) as rolling_avg_3days
17 from tweets
18 group by user_id, tweet_date
19
20
```

tweets

```
select user_id, date_format(tweet_date, '%m/%d/%Y %h:%i:%s') as tweet_date,
round(avg(count(distinct tweet_id)) over(order by tweet_date rows between 2 preceding and current
  Input to filter result
  Free
  Cost: 4ms < 1 > Total 4
```

	user_id	tweet_date	rolling_avg_3days
1	111	06/01/2022 12:00:00	2.00
2	111	06/02/2022 12:00:00	1.50
3	254	06/02/2022 12:00:00	1.33
4	111	06/04/2022 12:00:00	1.00

Q99.

Assume you are given the tables below containing information on Snapchat users, their ages, and their time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

Hint- Use join and case

Output the age bucket and percentage of sending and opening snaps. Round the percentage to 2 decimal places.

Notes:

- You should calculate these percentages:
 - time sending / (time sending + time opening)
 - time opening / (time sending + time opening)
- To avoid integer division in percentages, multiply by 100.0 and not 100.

activities Table:

Column Name	Type
activity_id	Integer
user_id	Integer
activity_type	string ('send', 'open', 'chat')
time_spent	float
activity_date	Datetime

activities Example Input:

activity_id	user_id	activity_type	time_spent	activity_date
7274	123	Open	4.50	06/22/2022 12:00:00
2425	123	Send	3.50	06/22/2022 12:00:00
1413	456	Send	5.67	06/23/2022 12:00:00
1414	789	Chat	11.00	06/25/2022 12:00:00
2536	456	Open	3.00	06/25/2022 12:00:00

age_breakdown Table:

Column Name	Type
user_id	Integer
age_bucket	string ('21-25', '26-30', '31-25')

age_breakdown Example Input:

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

Example Output:

age_bucket	send_perc	open_perc
26-30	65.40	34.60
31-35	43.75	56.25

Solution:

```

select b.age_bucket,
round(100*sum(case when a.activity_type = 'Send' then a.time_spent else 0
end)/sum(a.time_spent),2) send_perc,
round(100*sum(case when a.activity_type = 'Open' then a.time_spent else 0
end)/sum(a.time_spent),2) open_perc
from
activities a
join
age_breakdown b
on a.user_id = b.user_id
where activity_type in ('Open', 'Send')
group by b.age_bucket
order by b.age_bucket;

```

```

# create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Active Connection | ⌂ Execute
1   create database test;
  ⌂ Execute
2   use test;
  ⌂ Execute
3   create table activities
4     (activity_id    Int,
5      user_id Int,
6      activity_type  varchar(5),
7      time_spent    float,
8      activity_date  Datetime
9    );
  ⌂ Execute
10  create table age_breakdown
11    (user_id    Int,
12      age_bucket  varchar(10)
13    );
  ⌂ Execute
14  insert into activities values
15  (7274, 123, 'Open', 4.58, '2022/06/22 12:00:00'),
16  (2425, 123, 'Send', 3.58, '2022/06/22 12:00:00'),
17  (1413, 456, 'Send', 5.67, '2022/06/23 12:00:00'),
18  (1414, 789, 'Chat', 11.00, '2022/06/25 12:00:00'),
19  (2536, 456, 'Open', 3.00, '2022/06/25 12:00:00');
  ⌂ Execute
20  insert into age_breakdown values
21  (123, '31-35'),
22  (456, '26-30'),
23  (789, '21-25');
  ⌂ Execute
24  select b.age_bucket,
25  round(100*sum(case when a.activity_type = 'Send' then a.time_spent else 0 end)/sum(a.time_spent),2) send_perc,
26  round(100*sum(case when a.activity_type = 'Open' then a.time_spent else 0 end)/sum(a.time_spent),2) open_perc
27  from
28  activities a
29  join
30  age_breakdown b
31  on a.user_id = b.user_id
32  where activity_type in ('Open', 'Send')
33  group by b.age_bucket
34  order by b.age_bucket;
35
36
activities ×
select b.age_bucket,
round(100*sum(case when a.activity_type = 'Send' then a.time_spent else 0 end)/sum(a.time_spent),2) send_perc,
round(100*sum(case when a.activity_type = 'Open' then a.time_spent else 0 end)/sum(a.time_spent),2) open_perc
Input to filter result 1 Free
age_bucket send_perc open_perc
26-30 65.4 34.6
31-35 43.75 56.25

```

Q100 .

The LinkedIn Creator team is looking for power creators who use their personal profile as a company or influencer page. This means that if someone's LinkedIn page has more followers than all the companies they work for, we can safely assume that person is a Power Creator. Keep in mind that if a person works at multiple companies, we should take into account the company with the most followers.

Level - Medium

Hint- Use join and group by

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

Assumptions:

- A person can work at multiple companies.
- In the case of multiple companies, use the one with largest follower base.

personal_profiles Table:

Column Name	Type
profile_id	integer
name	string
followers	integer

personal_profiles Example Input:

profile_id	name	followers
1	Nick Singh	92,000
2	Zach Wilson	199,000
3	Daliana Liu	171,000
4	Ravit Jain	107,000
5	Vin Vashishta	139,000
6	Susan Wojcicki	39,000

employee_company Table:

Column Name	Type
personal_profile_id	integer
company_id	integer

employee_company Example Input:

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

company_pages Table:

Column Name	Type
company_id	integer

name	string
followers	integer

company_pages Example Input:

company_id	Name	followers
1	The Data Science Podcast	8,000
2	Airbnb	700,000
3	The Ravit Show	6,000
4	DataLemur	200
5	YouTube	1,6000,000
6	DataScience.Vin	4,500
9	Ace The Data Science Interview	4479

Example Output:

profile_id
1
3
4
5

Solution:

```
select p.profile_id
from
personal_profiles p
join
employee_company e
on p.profile_id = e.personal_profile_id
join
company_pages c
on e.company_id = c.company_id
group by p.profile_id, p.followers
having p.followers > sum(c.followers)
order by profile_id;
```

```
cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1:3306 > create-db-template.sql > ...
+ Active Connection | > Execute
1  create database test;
> Execute
2  use test;
> Execute
3  create table personal_profiles
4  (profile_id int,
5   name  varchar(30),
6   followers  int
7  );
> Execute
8  create table employee_company
9  (personal_profile_id  int,
10  company_id  int
11 );
> Execute
12 create table company_pages
13 (company_id int,
14  name  varchar(30),
15  followers  int
16 );
> Execute
17 insert into personal_profiles values
18 (1, 'Nick Singh', 92000),
19 (2, 'Zach Wilson', 199000),
20 (3, 'Dalliana Liu', 171000),
21 (4, 'Ravit Jain', 107000),
22 (5, 'Vin Vashishta', 139000),
23 (6, 'Susan Wojcicki', 39000);
> Execute
24 insert into employee_company values
25 (1, 4),
26 (1, 9),
27 (2, 2),
28 (3, 1),
29 (4, 3),
30 (5, 6),
31 (6, 5);
> Execute
32 insert into company_pages values
33 (1, 'The Data Science Podcast', 8000),
34 (2, 'Airbnb', 700000),
35 (3, 'The Ravit Show', 6000),
36 (4, 'DataLemur', 200),
37 (5, 'YouTube', 16000000),
38 (6, 'DataScience.Vin', 4500),
39 (9, 'Ace The Data Science Interview', 4479);
> Execute
40 select p.profile_id
41 from
42 personal_profiles p
43 join
44 employee_company e
45 on p.profile_id = e.personal_profile_id
46 join
47 company_pages c
48 on e.company_id = c.company_id
49 group by p.profile_id, p.followers
50 having p.followers > sum(c.followers)
51 order by profile_id;
52
53
54
```

select p.profile_id
from
personal_profiles p
join
employee_company e
on p.profile_id = e.personal_profile_id
join
company_pages c
on e.company_id = c.company_id
group by p.profile_id, p.followers
having p.followers > sum(c.followers)
order by profile_id;

profile_id	profile_id
1	1
2	3
3	4
4	5

Q 101.

Table: UserActivity

Column Name	Type
username	varchar
activity	varchar
startDate	Date
endDate	Date

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

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input:

UserActivity table:

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

Output:

username	activity	startDate	endDate
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

```
Solution:  
with new as  
(select t.username, t.activity, t.startDate, t.endDate  
from(  
    select username, activity, startDate, endDate,  
dense_rank() over(partition by username order by endDate desc) as r  
from UserActivity)t  
where r = 2  
)  
select * from new  
union  
select n.username, n.activity, n.startDate, n.endDate  
from(  
    select username, activity, startDate, endDate,  
dense_rank() over(partition by username order by endDate desc) as r  
from UserActivity)n  
where r = 1 and username not in (select username from new);
```

```

create-db-template.sql ▾
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template
    ▶ Execute
4  create table UserActivity
5      (username  varchar(15),
6       activity   varchar(15),
7       startDate  Date,
8       endDate   Date
9   );
    ▶ Execute
10 insert into UserActivity values
11     ('Alice',    'Travel',    '2020-02-12',    '2020-02-20'),
12     ('Alice',    'Dancing',   '2020-02-21',    '2020-02-23'),
13     ('Alice',    'Travel',    '2020-02-24',    '2020-02-28'),
14     ('Bob',      'Travel',    '2020-02-11',    '2020-02-18');
    ▶ Execute
✓ 15 with new as
16     (select t.username, t.activity, t.startDate, t.endDate
17      from(
18          |  select username, activity, startDate, endDate,
19          |  dense_rank() over(partition by username order by endDate desc) as r
20      from UserActivity)t
21     where r = 2
22   )
23     select * from new
24   union
25     select n.username, n.activity, n.startDate, n.endDate
26      from(
27          |  select username, activity, startDate, endDate,
28          |  dense_rank() over(partition by username order by endDate desc) as r
29      from UserActivity)n
30     where r = 1 and username not in (select username from new);
31
Data X
with new as
(select t.username, t.activity, t.startDate, t.endDate
Free 1
+ 🔒 Q Input to filter result
username varchar activity varchar startDate date endDate date
1 Alice Dancing 2020-02-21 2020-02-23
2 Bob Travel 2020-02-11 2020-02-18
Cost 5ms < 1 > Total 2

```

Q102.

Table: UserActivity

Column Name	Type
username	Varchar
activity	Varchar
startDate	Date

endDate	Date
---------	------

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

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input:

UserActivity table:

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

Output:

username	activity	startDate	endDate
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

Solution:

```

with new as
(select t.username, t.activity, t.startDate, t.endDate
from(
    select username, activity, startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r
from UserActivity)t
where r = 2
)
select * from new
union
select n.username, n.activity, n.startDate, n.endDate
from(
    select username, activity, startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r
from UserActivity)n
where r = 1 and username not in (select username from new);

```

```
create-db-template.sql ^

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template
    ▶ Execute
4   create table UserActivity
5     (username  varchar(15),
6      activity   varchar(15),
7      startDate  Date,
8      endDate   Date
9    );
    ▶ Execute
10  insert into UserActivity values
11    ('Alice',  'Travel',  '2020-02-12',  '2020-02-20'),
12    ('Alice',  'Dancing', '2020-02-21',  '2020-02-23'),
13    ('Alice',  'Travel',  '2020-02-24',  '2020-02-28'),
14    ('Bob',    'Travel',  '2020-02-11',  '2020-02-18');
    ▶ Execute
✓ 15 with new as
16   (select t.username, t.activity, t.startDate, t.endDate
17    from(
18      | | select username, activity, startDate, endDate,
19      | | dense_rank() over(partition by username order by endDate desc) as r
20    from UserActivity)t
21   where r = 2
22   )
23   select * from new
24   union
25   select n.username, n.activity, n.startDate, n.endDate
26   from(
27      | | select username, activity, startDate, endDate,
28      | | dense_rank() over(partition by username order by endDate desc) as r
29    from UserActivity)n
30   where r = 1 and username not in (select username from new);
31
```

Data X

with new as
(select t.username, t.activity, t.startDate, t.endDate
Free 1

Input to filter result Cost: 5ms < 1 > Total 2

	username	activity	startDate	endDate
1	Alice	Dancing	2020-02-21	2020-02-23
2	Bob	Travel	2020-02-11	2020-02-18

Q103.

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

Input Format

The STUDENTS table is described as follows:

Column	Type
<i>ID</i>	<i>Integer</i>
<i>Name</i>	<i>String</i>
<i>Marks</i>	<i>Integer</i>

The Name column only contains uppercase (A-Z) and lowercase (a-z) letters.

Sample Input

<i>ID</i>	<i>Name</i>	<i>Marks</i>
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Sample Output

Ashley

Julia

Belvet

Explanation

Only Ashley, Julia, and Belvet have Marks > 75 . If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

Solution:

```
select name from Students
where marks > 75
order by right(name, 3), id;
```

create-db-template.sql X [Preview] README.md

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@0
    ▶ Execute
2   use test;
    ▶ Execute
3   create table Students
4     (id int,
5      name varchar(15),
6      marks int);
    ▶ Execute
7   insert into Students values
8     (1, 'Ashley', 81),
9     (2, 'Samantha', 75),
10    (4, 'Julia', 76),
11    (3, 'Belvet', 84);
    ▶ Execute
12  select name from Students
13  where marks > 75
14  order by right(name, 3), id;
15
```

Students X

```
select name from Students
where marks > 75
```

Free 1

	name	varchar
1	Ashley	
2	Julia	
3	Belvet	

Q104.

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

Input Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input

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

Sample Output

Angela

Michael

Todd

Joe

Explanation

Angela has been an employee for 1 month and earns \$3443 per month.

Michael has been an employee for 6 months and earns \$2017 per month.Todd has been an employee for 5 months and earns \$3396 per month.

Joe has been an employee for 9 months and earns \$3573 per month.We order our output by ascending employee_id.

Solution:

```
select name
from
Employee
where salary > 2000 and months < 10
order by employee_id;
```

create-db-template.sql

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0
  3  create table Employee
  4    (employee_id int,
  5     name varchar(12),
  6     months int,
  7     salary int);
    ▷ Execute
  8  insert into Employee values
  9    (12228, 'Rose', 15, 1968),
 10    (33645, 'Angela', 1, 3443),
 11    (45692, 'Frank', 17, 1608),
 12    (56118, 'Patrick', 7, 1345),
 13    (59725, 'Lisa', 11, 2330),
 14    (74197, 'Kimberly', 16, 4372),
 15    (78454, 'Bonnie', 8, 1771),
 16    (83565, 'Michael', 6, 2017),
 17    (98607, 'Todd', 5, 3396),
 18    (99989, 'Joe', 9, 3573);
    ▷ Execute
✓ 19  select name
 20  from
 21  Employee
 22  where salary > 2000 and months < 10
 23  order by employee_id;
 24
```

Employee

```
select name
from Employee
Input to filter result
Free 1
Cost:
```

	name
1	Angela
2	Michael
3	Todd
4	Joe

Q105

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.
Output one of the following statements for each record in the table:

- Equilateral: It's a triangle with sides of equal length.
- Isosceles: It's a triangle with sides of equal length.
- Scalene: It's a triangle with sides of differing lengths.
- Not A Triangle: The given values of A, B, and C don't form a triangle.

Input Format

The TRIANGLES table is described as follows:

<i>Column</i>	<i>Type</i>
A	<i>Integer</i>
B	<i>Integer</i>
C	<i>Integer</i>

Each row in the table denotes the lengths of each of a triangle's three sides.

Sample Input

A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

Sample Output

Isosceles

Equilateral

Scalene

Not A Triangle

Explanation

Values in the tuple(20,20,23) form an Isosceles triangle, because $A \equiv B$.

Values in the tuple(20,20,20) form an Equilateral triangle, because $A \equiv B \equiv C$. Values in the tuple(20,21,22) form a Scalene triangle, because $A \neq B \neq C$.

Values in the tuple (13,14,30) cannot form a triangle because the combined value of sides A and B is not larger than that of side C .

```
Solution:  
select case  
when A+B > C and B+C > A and C+A > B then  
(  
    case  
        when A != B and B != C then 'Scalene'  
        when A = B and B = C then 'Equilateral'  
        else 'Isosceles'  
    end  
)  
else 'Not A Triangle'  
end as Result  
from Triangles;
```

create-db-template.sql

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
  1  create database test;
     ▶ Execute
  2  use test;
     ▶ Execute
  3  create table Triangles
  4    (A Int,
  5     B Int,
  6     C Int);
     ▶ Execute
  7  insert into Triangles values
  8    (20, 20, 23),
  9    (20, 20, 20),
 10   (20, 21, 22),
 11   (13, 14, 30);
 12
     ▶ Execute
 13  select case
 14    when A+B > C and B+C > A and C+A > B then
 15      (
 16        case
 17          when A != B and B != C then 'Scalene'
 18          when A = B and B = C then 'Equilateral'
 19          else 'Isosceles'
 20        end
 21      )
 22    else 'Not A Triangle'
 23  end as Result
 24  from Triangles;

```

Triangles

		Result
		varchar
1	Isosceles	
2	Equilateral	
3	Scalene	
4	Not A Triangle	

Q106.

Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

Input Format

The EMPLOYEES table is described as follows:

<i>Column</i>	<i>Type</i>
<i>ID</i>	<i>Integer</i>
<i>Name</i>	<i>String</i>
<i>Salary</i>	<i>Integer</i>

Note: Salary is per month.

Constraints

$1000 < \text{salary} < 10^5$

Sample Input

<i>ID</i>	<i>Name</i>	<i>Salary</i>
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

Sample Output

2061

Explanation

The table below shows the salaries without zeros as they were entered by Samantha:

<i>ID</i>	<i>Name</i>	<i>Salary</i>
1	<i>Kristeen</i>	142
2	<i>Ashley</i>	26
3	<i>Julia</i>	221
4	<i>Maria</i>	3

Samantha computes an average salary of 98.00 . The actual average salary is 2159.00.

The resulting error between the two calculations is $2159.00 - 98.00 = 2061.00$. Since it is equal to the integer 2061, it does not get rounded up.

Solution:

```
select ceil(avg(salary) - avg(replace(salary, 0, '')))
as calculation_difference
from Employees;
```

The screenshot shows the MySQL Workbench interface. The top window is titled "create-db-template.sql" and contains a SQL script with numbered lines. Lines 1 through 11 are standard SQL statements for creating a database named "test", selecting it, creating a table "Employees" with columns "id", "name", and "salary", and inserting four rows of data. Line 12 is a query that has been executed successfully, indicated by a green checkmark. This query calculates the difference between the average salary and the average salary after removing the first digit of each salary value. The result is displayed in the bottom window, which is titled "Employees". The results grid shows one row with the value 2061.

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1
    ⚡ Active Connection | ▶ Execute
1   create database test;
    ▶ Execute
2   use test;
    ▶ Execute
3   create table Employees
4       (id int,
5        name varchar(15),
6        salary int);
    ▶ Execute
7   insert into Employees values
8       (1, 'Kristeen', 1420),
9       (2, 'Ashley', 2006),
10      (3, 'Julia', 2210),
11      (4, 'Maria', 3000);
    ▶ Execute
12  select ceil(avg(salary) - avg(replace(salary, 0, '')))
13  as calculation_difference
14  from Employees;
15
16
17
18

```

		calculation_difference
	1	2061

Q107.

We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

Level - Easy

Hint - Use Aggregation functions

Input Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input

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

Sample Output

69952 1

Explanation:

The table and earnings data is depicted in the following diagram:

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

The maximum earnings value is 69952. The only employee with earnings= 69952 is Kimberly, so we print the maximum earnings value (69952) and a count of the number of employees who have earned \$69952 (which is 1) as two space-separated values.

```
select concat(max(t.earnings), ' ',  
sum(case  
    when earnings = max_salary then 1  
    else 0  
end)) as Output  
from  
(  
    select max(salary*months) over() as max_salary,  
salary*months as earnings  
from  
Employee) t;
```

```
create-db-template.sql X [Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > ■

2  use test;
   ▷ Execute
3  create table Employee
4    (employee_id int,
5     name varchar(12),
6     months int,
7     salary int);
   ▷ Execute
8  insert into Employee values
9    (12228, 'Rose', 15, 1968),
10   (33645, 'Angela', 1, 3443),
11   (45692, 'Frank', 17, 1608),
12   (56118, 'Patrick', 7, 1345),
13   (59725, 'Lisa', 11, 2330),
14   (74197, 'Kimberly', 16, 4372),
15   (78454, 'Bonnie', 8, 1771),
16   (83565, 'Michael', 6, 2017),
17   (98607, 'Todd', 5, 3396),
18   (99989, 'Joe', 9, 3573);
19
20  ▷ Execute
21  select concat(max(t.earnings), ' ',
22  sum(case
23    | when earnings = max_salary then 1
24    | else 0
25    | end)) as Output
26  from
27  (
28    select max(salary*months) over() as max_salary,
29    salary*months as earnings
30    from
31    Employee) t;
```

Employee X

```
select concat(max(t.earnings), ' ',
sum(case
when earnings = max_salary then 1
else 0
end)) as Output
from
(
select max(salary*months) over() as max_salary,
salary*months as earnings
from
Employee) t;
```

Free 1

	Output	varchar
1	69952	1

Cost: 4ms < 1

Q108.

Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).

Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

Level - Medium

There are a total of [occupation_count] [occupation]s.

2. where [occupation_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.

Note: There will be at least two entries in the table for each type of occupation.

Input Format

The OCCUPATIONS table is described as follows:

<i>Column</i>	<i>Type</i>
<i>Name</i>	<i>String</i>
<i>Occupation</i>	<i>String</i>

Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor.

Sample Input

An OCCUPATIONS table that contains the following records:

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

Sample Output

Ashely(P)

Christeen(P)

Jane(A)

Jenny(D)

Julia(A)

Ketty(P)

Maria(A)

Meera(S)

Priya(S)

Samantha(D)

There are a total of 2 doctors.

There are a total of 2 singers.

There are a total of 3 actors.

There are a total of 3 professors.

Hint -

The results of the first query are formatted to the problem description's specifications.

The results of the second query are ascendingly ordered first by number of names corresponding to each profession ($2 \leq 2 \leq 3 \leq 3$), and then alphabetically by profession (doctor \leq singer , and actor \leq professor).

```

Solution:
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
order by name;
select
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation),
's.') as occupation_count
from Occupations
group by occupation
order by count(occupation), occupation;

```

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

```

3  create table Occupations
4    (name varchar(15),
5     occupation varchar(15));
  ↴ Execute
6  insert into Occupations values
7    ('Samantha', 'Doctor'),
8    ('Julia', 'Actor'),
9    ('Maria', 'Actor'),
10   ('Meera', 'Singer'),
11   ('Ashley', 'Professor'),
12   ('Ketty', 'Professor'),
13   ('Christeen', 'Professor'),
14   ('Jane', 'Actor'),
15   ('Jenny', 'Doctor'),
16   ('Priya', 'Singer');
  ↴ Execute
✓ 17  select concat(name, '(', left(occupation,1),')') as name_occupation
18  from Occupations
19  order by name;
  ↴ Execute
20  select
21    concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's') as occupation_count
22  from Occupations
23  group by occupation
24  order by count(occupation), occupation;
  ↴ Execute

```

Occupations X

		name_occupation
		varchar
1		Ashley(P)
2		Christeen(P)
3		Jane(A)
4		Jenny(D)
5		Julia(A)
6		Ketty(P)
7		Maria(A)
8		Meera(S)
9		Priya(S)
10		Samantha(D)

Free 1

1

+

-

↑

↓

⌂

⌂

⌂

⌂

⌂

⌂

⌂

⌂

⌂

⌂

⌂

⌂

⌂

Cost: 4ms

1

Total 10

create-db-template.sql ×

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

```
3  create table Occupations
4  (name varchar(15),
5  occupation varchar(15));
    ▷ Execute
6  insert into Occupations values
7  ('Samantha', 'Doctor'),
8  ('Julia', 'Actor'),
9  ('Maria', 'Actor'),
10 ('Meera', 'Singer'),
11 ('Ashley', 'Professor'),
12 ('Ketty', 'Professor'),
13 ('Christeen', 'Professor'),
14 ('Jane', 'Actor'),
15 ('Jenny', 'Doctor'),
16 ('Priya', 'Singer');
    ▷ Execute
17 select concat(name, '(', left(occupation,1),')') as name_occupation
18 from Occupations
19 order by name;
    ▷ Execute
✓ 20 select
21 concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count
22 from Occupations
23 group by occupation
24 order by count(occupation), occupation;
    ▷ Execute
```

Occupations X

select
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count

Free 1 Input to filter result ↻ 1 + + - ⚡ ⬆ ⬇ ⏪ ⏩ Cost: 5ms < 1 > Total 4

	occupation_count
1	There are a total of 2 doctors.
2	There are a total of 2 singers.
3	There are a total of 3 actors.
4	There are a total of 3 professors.

Q109 .

Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print NULL when there are no more names corresponding to an occupation.

Input Format

The OCCUPATIONS table is described as follows:

Column	Type
Name	String
Occupation	String

Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor.

Sample Input

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

Sample Output

Jenny Ashley Meera Jane

Samantha Christeen Priya Julia

NULL Ketty NULL Maria

Hint -

The first column is an alphabetically ordered list of Doctor names.
The second column is an alphabetically ordered list of Professor names. The
third column is an alphabetically ordered list of Singer names.
The fourth column is an alphabetically ordered list of Actor names.
The empty cell data for columns with less than the maximum number of names per occupation (in
this case, the Professor and Actor columns) are filled with NULL values.

Solution:

```
select max(case Occupation when 'Doctor' then Name end) as Doctors,
       max(case Occupation when 'Professor' then Name end) as Professors,
       max(case Occupation when 'Singer' then Name end) as Singers,
       max(case Occupation when 'Actor' then Name end) as Actors
  from (
    select occupation, name,
           row_number() over(partition by Occupation order by name) as r
      from Occupations
     ) t
 group by r;
```

The screenshot shows a MySQL Workbench interface. The top window displays a SQL script named 'create-db-template.sql' with several lines of code. Line 17 is highlighted with a green checkmark, indicating it has been executed successfully. The bottom window shows the results of the query, titled 'Occupations'. The results grid contains three rows of data:

	Doctors	Professors	Singers	Actors
1	Jenny	Ashley	Meera	Jane
2	Samantha	Christeen	Priya	Julia
3	(NULL)	Ketty	(NULL)	Maria

Q110.

You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.

Column	Type
N	Integer
P	Integer

Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

Sample Input

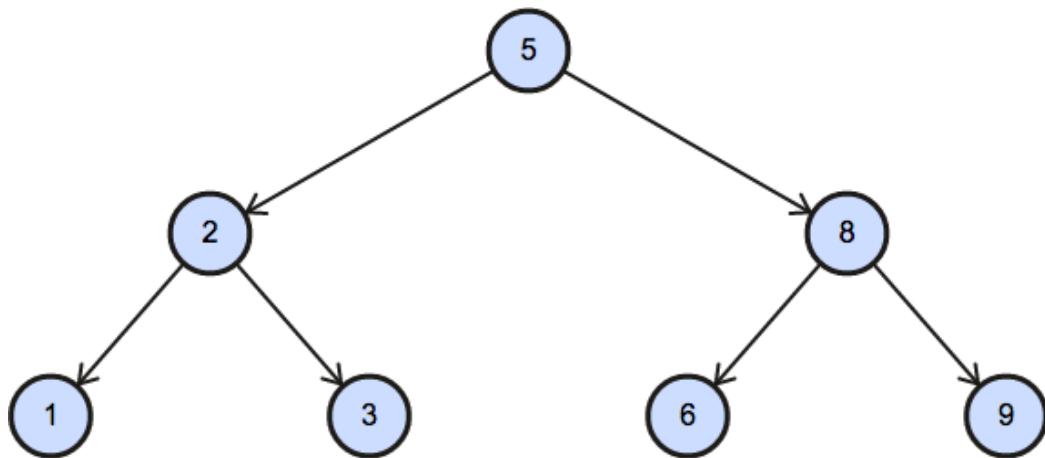
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	<i>null</i>

Sample Output

1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
8 Inner
9 Leaf

Explanation

The Binary Tree below illustrates the sample:



Solution:

```
select
(
    case
        when P is NULL then 'Root'
        when N not in (select distinct P from BST where P is not null) then 'Leaf'
    else 'Inner'
    end
) as Node_Type
from BST
order by N;
```

The screenshot shows a MySQL Workbench interface. At the top, there's a tab bar with 'create-db-template.sql' (highlighted), 'Preview', and 'README.md'. Below the tabs, the connection path is listed: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template. A tooltip indicates an 'Active Connection | Execute'.

```

1  create database test;
2  use test;
3  create table BST
4  (N int,
5  P int);
6  insert into BST values
7  (1,2),
8  (3,2),(6,8),(9,8),(2,5),(8,5),(5, null);
9  select
10 (
11   case
12   when P is NULL then 'Root'
13   when N not in (select distinct P from BST where P is not null) then 'Leaf'
14   else 'Inner'
15   end
16 ) as Node_Type
17 from BST
18 order by N;
19

```

The results of the query are displayed in a table titled 'BST' with one row:

	N	Node_Type
	1	Leaf
	2	Inner
	3	Leaf
	4	Root
	5	Leaf
	6	Inner
	7	Leaf

Below the table, the status bar shows 'Cost: 3ms' and 'Total 7'.

Q111 .

Amber's conglomerate corporation just acquired some new companies. Each of the companies

Founder



Lead Manager



Senior Manager



Manager



Employee

follows this hierarchy:

Given the table schemas below, write a query to print the company_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code.

Level - Medium

Note:

The tables may contain duplicate records.

- The company_code is string, so the sorting should not be numeric. For example, if the company_codes are C_1, C_2, and C_10, then the ascending company_codes will be C_1, C_10, and C_2.

Input Format

The following tables contain company data:

- Company: The company_code is the code of the company and founder is the founder of the

Column	Type
company_code	String
founder	String

company.

- Lead_Manager: The lead_manager_code is the code of the lead manager, and the

Column	Type
lead_manager_code	String
company_code	String

company_code is the code of the working company.

- Senior_Manager: The senior_manager_code is the code of the senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the

Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

working company.

- Manager: The manager_code is the code of the manager, the senior_manager_code is the code of its senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the working company.

Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

- Employee: The employee_code is the code of the employee, the manager_code is the code of its manager, the senior_manager_code is the code of its senior manager, the

`lead_manager_code` is the code of its lead manager, and the `company_code` is the code of the

Column	Type
<code>employee_code</code>	String
<code>manager_code</code>	String
<code>senior_manager_code</code>	String
<code>lead_manager_code</code>	String
<code>company_code</code>	String

working company.

Sample Input

company_code	founder
C1	Monika
C2	Samantha

Company Table:

lead_manager_code	company_code
LM1	C1
LM2	C2

Lead_Manager Table:

Senior_Manager Table:

senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

Manager Table:

manager_code	senior_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

Employee Table:

employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

Sample Output

C1 Monika 1 2 1 2

C2 Samantha 1 1 2 2

Hint -

In company C1, the only lead manager is LM1. There are two senior managers, SM1 and SM2, under LM1. There is one manager, M1, under senior manager SM1. There are two employees, E1 and E2, under manager M1.

In company C2, the only lead manager is LM2. There is one senior manager, SM3, under LM2. There are two managers, M2 and M3, under senior manager SM3. There is one employee, E3, under manager M2, and another employee, E4, under manager, M3.

Solution:

```
select concat(c.company_code, ' ', c.founder, ' ',  
count(distinct l.lead_manager_code), ' ',  
count(distinct s.senior_manager_code), ' ',  
count(distinct m.manager_code), ' ',  
count(distinct e.employee_code)) as Output  
from Company c  
left outer join  
Lead_Manager l  
on c.company_code = l.company_code  
left join  
Senior_Manager s  
on l.lead_manager_code = s.lead_manager_code  
left join  
Manager m  
on s.senior_manager_code = m.senior_manager_code  
left join  
Employee e  
on m.manager_code = e.manager_code  
group by c.company_code, c.founder  
order by c.company_code;
```

The screenshot shows a MySQL Workbench interface. On the left, a script editor displays a SQL file named 'create-db-template.sql'. The code creates four tables: Company, Lead_Manager, Senior_Manager, and Manager, and inserts data into them. It then defines a query to select concatenated company codes, founders, and counts of distinct manager levels (Lead, Senior, Manager, Employee) into an 'Employee' table. The right side shows the results of this query in a grid:

	Output
1	C1 Monika 1 2 1 2
2	C2 Samantha 1 1 2 2

Q112.

Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand () character as your separator (instead of a space).

For example, the output for all prime numbers ≤ 10 would be: 2&3&5&7

Hint - Firstly, select L Prime_Number from (select Level L from Dual connect Level ≤ 1000) and then do the same thing to create Level M, and then filter by M \leq L and then group by L having count(case when L/M = trunc(L/M) then 'Y' end) = 2 order by L

Solution:

```
with recursive cte as
(
    select 2 as num
```

```

        union
        select num+1 from cte
        where num+1 <= 1000
    )
select GROUP_CONCAT(num SEPARATOR "&") as prime
from
(
    select 2 as num
    union
    select c1.num from cte c1
    inner join
    cte c2 on c2.num <= round(c1.num/2)
    group by num
    having min(c1.num % c2.num) > 0
    order by num
)t;

```

create-db-template.sql

```

1 create database test;
2 use test;
3 with recursive cte as
4 (
5     select 2 as num
6     union
7     select num+1 from cte
8     where num+1 <= 1000
9 )
10 select GROUP_CONCAT(num SEPARATOR "&") as prime
11 from
12 (
13 select 2 as num
14 union
15 select c1.num from cte c1
16 inner join
17 cte c2 on c2.num <= round(c1.num/2)
18 group by num
19 having min(c1.num % c2.num) > 0
20 order by num
21 )t;
22

```

Q113.

$P(R)$ represents a pattern drawn by Julia in R rows. The following pattern represents $P(5)$:

```

*
**
***
```

Write a query to print the pattern P(20).Level

- Easy

Source - Hackerrank

Hint - Use SYS_CONNECT_BY_PATH(NULL, '*') FROM DUAL

Solution:

```
with recursive num(n) as
(
  select 1
    union
    select n + 1
      from num
     where n + 1 <= 20
)
select lpad(' ', num.n, '*') as 'P(20)'
from num;
```

The screenshot shows the Oracle SQL Developer interface. On the left, a code editor window titled "create-db-template.sql" displays the following SQL code:

```
1 create database test;
2 use test;
3 with recursive num(n) as
4 (
5   select 1
6     union
7     select n + 1
8       from num
9      where n + 1 <= 20
10 )
11 select lpad(' ', num.n, '**') as 'P(20)'
12 from num;
```

On the right, a results window titled "Data" shows the output of the query. The results are displayed in a grid format, with each row containing a number from 1 to 20 followed by a varying number of asterisks. The column headers are "P(20)" and "long_blob". The results are as follows:

P(20)	long_blob
1	*
2	**
3	***
4	****
5	*****
6	*****
7	*****
8	*****
9	*****
10	*****
11	*****
12	*****
13	*****
14	*****
15	*****
16	*****
17	*****
18	*****
19	*****
20	*****

Q114.

P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

```
*****
****
 ***
 **
 *
```

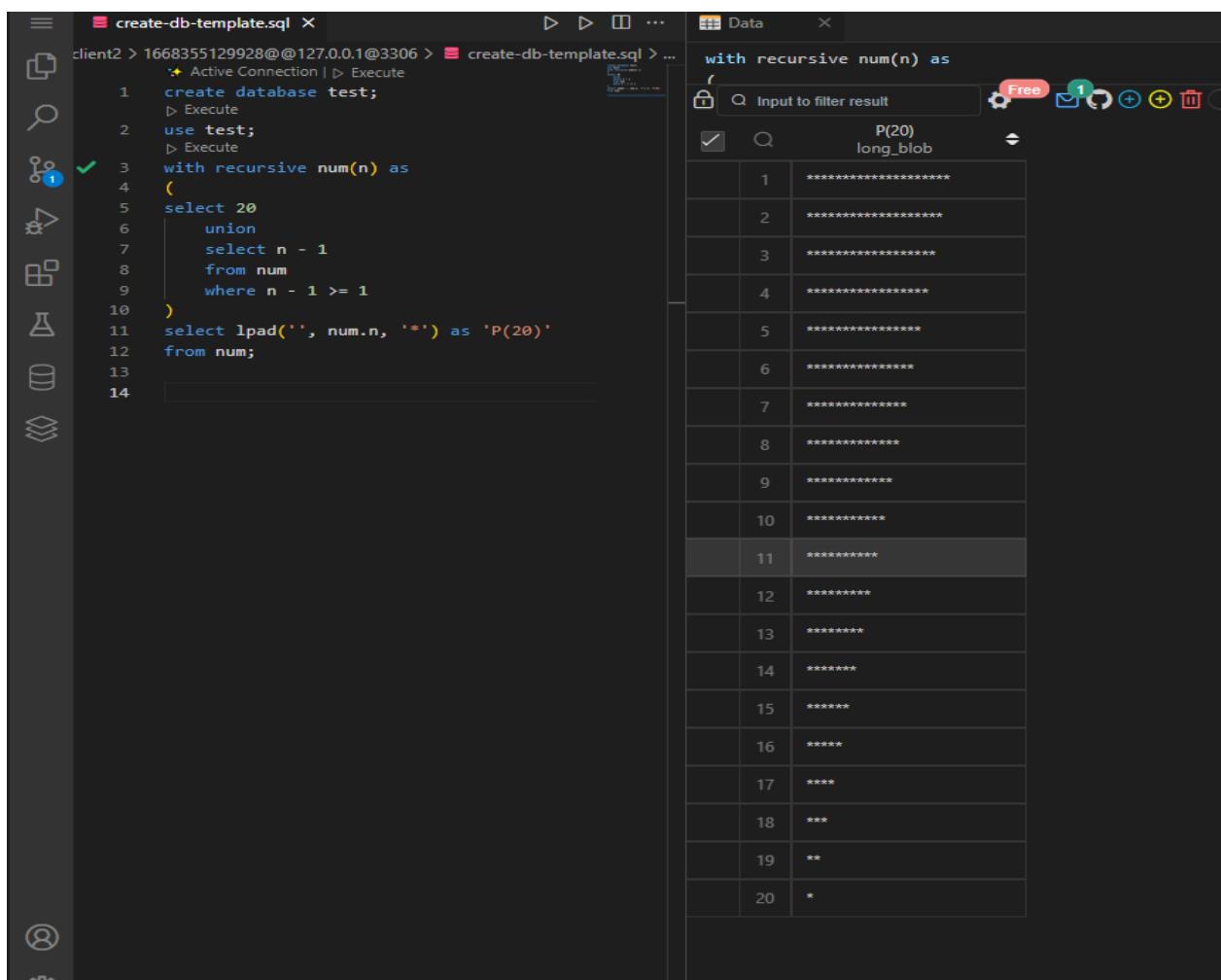
Write a query to print the pattern P(20).Level

- Easy

Hint - Use SYS_CONNECT_BY_PATH(NULL, '*') FROM DUAL

Solution:

```
with recursive num(n) as
(
select 20
    union
    select n - 1
    from num
    where n - 1 >= 1
)
select lpad(' ', num.n, '*') as 'P(20)'
from num;
```



The screenshot shows the Oracle SQL Developer interface. On the left, the 'Script Editor' pane displays the SQL script 'create-db-template.sql' with the following content:

```
client2 > 1668355129928@127.0.0.1:3306 > create-db-template.sql > ...
 1  create database test;
 2  use test;
 3  with recursive num(n) as
 4  (
 5    select 20
 6    union
 7    select n - 1
 8    from num
 9    where n - 1 >= 1
10  )
11  select lpad(' ', num.n, '*') as 'P(20)'
12  from num;
13
14
```

The 'Data' pane on the right shows the resulting output for 'P(20)' as a long blob:

Row	Content
1	*****
2	****
3	***
4	**
5	*

Q116. You are given a table, Functions, containing two columns: X and Y.

Column	Type
X	Integer
Y	Integer

Two pairs (X_1, Y_1) and (X_2, Y_2) are said to be symmetric pairs if $X_1 = Y_2$ and $X_2 = Y_1$.

Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that $X_1 \leq Y_1$.

Sample Input

X	Y
20	20
20	20
20	21
23	22
22	23
21	20

Sample Output

```
20 20  
20 21  
22 23
```

Solution:

```
select distinct a.X, a.Y from  
(select *, row_number() over(order by X) as r1 from Functions) a  
inner join  
(select *,row_number() over(order by X) as r2 from Functions) b  
on a.X = b.Y and b.X = a.Y  
where a.X <= a.Y and a.r1 <> b.r2  
order by a.X
```

The screenshot shows a MySQL client interface with two tabs: "create-db-template.sql" and "Employee".

The "create-db-template.sql" tab contains the following SQL code:

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >  
  ↗ Active Connection | ▶ Execute  
1  create database test;  
  ▶ Execute  
2  use test;  
  ▶ Execute  
3  create table Functions  
4  (  
5    X int,  
6    Y int  
7  );  
  ▶ Execute  
8  insert into Functions values  
9  (20, 20),  
10 (20, 20),  
11 (20, 21),  
12 (23, 22),  
13 (22, 23),  
14 (21, 20);  
  ▶ Execute  
15 select distinct a.X, a.Y from  
16 (select *, row_number() over(order by X) as r1 from Functions) a  
17 inner join  
18 (select *,row_number() over(order by X) as r2 from Functions) b  
19 on a.X = b.Y and b.X = a.Y  
20 where a.X <= a.Y and a.r1 <> b.r2  
21 order by a.X  
22  
23
```

The "Employee" tab displays the results of the query:

```
select distinct a.X, a.Y from  
(select *, row_number() over(order by X) as r1 from Functions) a
```

Input to filter result: Free 1

	X	Y
1	20	20
2	20	21
3	22	23

Cost: 3ms

Q115.

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

Level - Easy

Hint - Use Like

Input Format

Column	Type
<i>ID</i>	<i>Integer</i>
<i>Name</i>	<i>String</i>
<i>Marks</i>	<i>Integer</i>

The STUDENTS table is described as follows:

The Name column only contains uppercase (A-Z) and lowercase (a-z) letters.

Sample Input

<i>ID</i>	<i>Name</i>	<i>Marks</i>
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

Sample Output

Ashley

Julia

Belvet

Explanation

Only Ashley, Julia, and Belvet have Marks > 75 . If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

Solution:

```
select name from Students
where marks > 75
order by right(name, 3), id;
```

The screenshot shows a VS Code interface with two tabs: 'create-db-template.sql' and '[Preview] README.md'. The 'create-db-template.sql' tab contains the following MySQL script:

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@0
    ▷ Execute
2   use test;
    ▷ Execute
3   create table Students
4     (id int,
5      name varchar(15),
6      marks int);
    ▷ Execute
7   insert into Students values
8     (1, 'Ashley', 81),
9     (2, 'Samantha', 75),
10    (4, 'Julia', 76),
11    (3, 'Belvet', 84);
    ▷ Execute
12  select name from Students
13  where marks > 75
14  order by right(name, 3), id;
15
```

The '[Preview] README.md' tab is currently empty.

Below the tabs, there is a preview pane titled 'Students' showing the results of the query:

```
select name from Students
where marks > 75
```

The preview pane includes a toolbar with icons for refresh, free, filter, and other database operations. The table data is as follows:

		name
	1	Ashley
	2	Julia
	3	Belvet

Q116.

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

Level - Easy

Hint - Use ORDER BY

Input Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

Sample Input

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

Sample Output

```
Angela
Bonnie
Frank
Joe
Kimberly
Lisa
Michael
Patrick
Rose
Todd
```

Solution:

```
select name
from
Employee
order by name;
```

The screenshot shows a VS Code interface with a dark theme. On the left, there's a sidebar with a tree view showing a 'create-db-template.sql' file, a 'User' section, and a 'globalStorage' section. Below that is a 'Employee' table viewer. At the bottom, there's a status bar showing 'Cost: 5ms'.

Employee

	name	varchar
1	Angela	
2	Bonnie	
3	Frank	
4	Joe	
5	Kimberly	
6	Lisa	
7	Michael	
8	Patrick	
9	Rose	
10	Todd	

Q117. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than \$2000 per month who have been employees for less than 10 months. Sort your result by ascending employee_id.

Level - Easy

Hint - Use AscendingInput

Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input

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

Sample Output

Angela

Michael

Todd

Joe

Explanation

Angela has been an employee for 1 month and earns \$3443 per month.

Michael has been an employee for 6 months and earns \$2017 per month.

Todd has been an employee for 5 months and earns \$3396 per month.
 Joe has been an employee for 9 months and earns \$3573 per month.
 We order our output by ascending employee_id.

Q118. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths. Output one of the following statements for each record in the table:

- Equilateral: It's a triangle with sides of equal length.
- Isosceles: It's a triangle with sides of equal length.
- Scalene: It's a triangle with sides of differing lengths.
- Not A Triangle: The given values of A, B, and C don't form a triangle.

Level - Easy

Hint - Use predefined functions for calculation.

Input Format

The TRIANGLES table is described as follows:

Column	Type
A	Integer
B	Integer
C	Integer

Each row in the table denotes the lengths of each of a triangle's three sides.

Sample Input

A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

Sample Output

Isosceles

Equilateral

Scalene

Not A Triangle

Explanation

Values in the tuple(20,20,23) form an Isosceles triangle, because $A \equiv B$.

Values in the tuple(20,20,20) form an Equilateral triangle, because $A \equiv B \equiv C$. Values in the tuple(20,21,22) form a Scalene triangle, because $A \neq B \neq C$.

Values in the tuple (13,14,30) cannot form a triangle because the combined value of sides A and B is not larger than that of side C .

```

Solution:
select case
when A+B > C and B+C > A and C+A > B then
(
    case
        when A != B and B != C then 'Scalene'
        when A = B and B = C then 'Equilateral'
        else 'Isosceles'
    end
)
else 'Not A Triangle'
end as Result
from Triangles;

```

create-db-template.sql

```

1 create database test;
2 use test;
3 create table Triangles
4 (A Int,
5 B Int,
6 C Int);
7 insert into Triangles values
8 (20, 20, 23),
9 (20, 20, 20),
10 (20, 21, 22),
11 (13, 14, 30);
12
13 select case
14 when A+B > C and B+C > A and C+A > B then
15 (
16     case
17         when A != B and B != C then 'Scalene'
18         when A = B and B = C then 'Equilateral'
19         else 'Isosceles'
20     end
21 )
22 else 'Not A Triangle'
23 end as Result
24 from Triangles;

```

Triangles

	Result
<input checked="" type="checkbox"/>	varchar
1	Isosceles
2	Equilateral
3	Scalene
4	Not A Triangle

Q119. Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the year-on-year growth rate for the total spend of each product for each year.

Output the year (in ascending order) partitioned by product id, current year's spend, previous year's spend and year-on-year growth rate (percentage rounded to 2 decimal places).

Level - Hard

Hint - Use extract function

user_transactions Table:

Column Name	Type
transaction_id	Integer
product_id	Integer
spend	decimal
transaction_date	datetime

user_transactions Example Input:

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

Example Output:

y	product_id	curr_year_spend	prev_year_spend	yoym_rate
2	123424	1500.60		
2	123424	1000.20	1500.60	-33.35
2	123424	1246.44	1000.20	24.62
2	123424	2145.32	1246.44	72.12

Solution:

```
select year, product_id, curr_year_spend, coalesce(prev_year_spend, '') as
prev_year_spend,
coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100,2), '') as
yoym_rate
from
(
    select year(transaction_date) as year, product_id, spend as curr_year_spend,
round(lag(spend,1) over(partition by product_id order by transaction_date),2)
as prev_year_spend
from user_transactions
) t;
```

```

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   create database test;
  ▶ Execute
2   use test;
  ▶ Execute
3   create table user_transactions
4     (transaction_id Int,
5      product_id Int,
6      Spend float,
7      transaction_date DATETIME
8    );
  ▶ Execute
9   insert into user_transactions values
10  (1341, 123424, 1500.60, '2019/12/31 12:00:00'),
11  (1423, 123424, 1000.20, '2020/12/31 12:00:00'),
12  (1623, 123424, 1246.44, '2021/12/31 12:00:00'),
13  (1322, 123424, 2145.32, '2022/12/31 12:00:00');
  ▶ Execute
14  select year, product_id, curr_year_spend, coalesce(prev_year_spend,'') as prev_year_spend,
15  coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100,2),'') as yoy_rate
16  from
17  (
18    select year(transaction_date) as year, product_id, spend as curr_year_spend,
19    round(lag(spend,1) over(partition by product_id order by transaction_date),2) as prev_year_spend
20  from user_transactions
21 ) t;
22

```

user_transactions X

```

select year, product_id, curr_year_spend, coalesce(prev_year_spend,'') as prev_year_spend,
coalesce(round((curr_year_spend - prev_year_spend)/prev_year_spend *100.2),'') as yoy_rate

```

Free 1 ↻ 🔍 + 🗑️ ⏷ ⏵ ⏴ Cost: 6ms < 1 > Total 4

Input to filter result

	year	product_id	curr_year_spend	prev_year_spend	yoy_rate
	int	int	float	varchar	varchar
1	2019	123424	1500.6		
2	2020	123424	1000.2	1500.6	-33.35
3	2021	123424	1246.44	1000.2	24.62
4	2022	123424	2145.32	1246.44	72.12

Q120. Amazon wants to maximise the number of items it can stock in a 500,000 square feet warehouse. It wants to stock as many prime items as possible, and afterwards use the remaining square footage to stock the most number of non-prime items.

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

Hint - create a table containing a summary of the necessary fields such as item type ('prime_eligible', 'not_prime'), SUM of square footage, and COUNT of items grouped by the item type.

inventory table:

Column Name	Type
item_id	integer
item_type	string
item_category	string

square_footage	decimal
----------------	---------

inventory Example Input:

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

Example Output:

item_type	item_count
prime_eligible	9285
not_prime	6

Solution:

```
select item_type, (case
    when item_type = 'prime_eligible' then floor(500000/sum(square_footage)) *
count(item_type)
    when item_type = 'not_prime' then floor((500000 -(select
floor(500000/sum(square_footage)) * sum(square_footage) from inventory where
item_type = 'prime_eligible'))/sum(square_footage)) * count(item_type)
end) as item_count
from inventory
group by item_type
order by count(item_type) desc;
```

The screenshot shows a MySQL Workbench interface. The top part is a query editor with the following SQL code:

```

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 inventory
4  (
5    item_id int,
6    item_type varchar(20),
7    item_category varchar(20),
8    square_footage float
9  );
  ▷ Execute
10 insert into inventory values
11 (1374, 'prime_eligible', 'mini refrigerator', 68.00),
12 (4245, 'not_prime', 'standing lamp', 26.40),
13 (2452, 'prime_eligible', 'television', 85.00),
14 (3255, 'not_prime', 'side table', 22.60),
15 (1672, 'prime_eligible', 'Laptop', 8.50);
  ▷ Execute
16 select item_type, (case
17   when item_type = 'prime_eligible' then floor(500000/sum(square_footage)) * count(item_type)
18   when item_type = 'not_prime' then floor((500000 - (select floor(500000/sum(square_footage))
19   * sum(square_footage) from inventory where item_type = 'prime_eligible'))) / sum(square_footage) * count(item_type)
20 end) as item_count
21 from inventory
22 group by item_type
23 order by count(item_type) desc;
24

```

The bottom part shows the results of the last query in a grid:

	item_type	item_count
1	prime_eligible	9285
2	not_prime	6

Q121. Assume you have the table below containing information on Facebook user actions. Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the number of monthly active users (MAUs).

Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month and last month.

Hint- Use generic correlated subquery

user_actions Table:

Column Name	Type
user_id	Integer
event_id	Integer
event_type	string ("sign-in", "like", "comment")
event_date	Datetime

user_actionsExample Input:

user_id	event_id	event_type	event_date
445	7765	sign-in	05/31/2022 12:00:00
742	6458	sign-in	06/03/2022 12:00:00
445	3634	Like	06/05/2022 12:00:00
742	1374	Comment	06/05/2022 12:00:00
648	3124	Like	06/18/2022 12:00:00

Example Output for June 2022:

month	monthly_active_users
6	1

Solution: For July Month

```
select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users
from
user_actions a
inner join
user_actions b
on concat(month(a.event_date),year(a.event_date)) =
concat(1+month(b.event_date),year(b.event_date))
and a.user_id = b.user_id
where a.event_type in ('sign-in', 'like', 'comment')
and b.event_type in ('sign-in', 'like', 'comment')
and concat(month(a.event_date), '/', year(a.event_date)) = '7/2022'
and concat(1+month(b.event_date), '/', year(b.event_date)) = '7/2022'
group by month(a.event_date);
```

Solution: For June Month

```
select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users
from
user_actions a
inner join
user_actions b
on concat(month(a.event_date),year(a.event_date)) =
concat(1+month(b.event_date),year(b.event_date))
and a.user_id = b.user_id
where a.event_type in ('sign-in', 'like', 'comment')
and b.event_type in ('sign-in', 'like', 'comment')
and concat(month(a.event_date), '/', year(a.event_date)) = '6/2022'
```

```
and concat(1+month(b.event_date), '/', year(b.event_date)) = '6/2022'  
group by month(a.event_date);
```

create-db-template.sql X

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

 ▷ Execute

2 use test;

 ▷ Execute

3 create table user_actions

4 (user_id Int,

5 event_id Int,

6 event_type varchar(10),

7 event_date datetime

8);

 ▷ Execute

9 insert into user_actions values

10 (445, 7765, 'sign-in', '2022/05/31 12:00:00'),

11 (742, 6458, 'sign-in', '2022/06/03 12:00:00'),

12 (445, 3634, 'like', '2022/06/05 12:00:00'),

13 (742, 1374, 'comment', '2022/06/05 12:00:00'),

14 (648, 3124, 'like', '2022/06/10 12:00:00');

 ▷ Execute

15 select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users

16 from

17 user_actions a

18 inner join

19 user_actions b

20 on concat(month(a.event_date),year(a.event_date)) = concat(1+month(b.event_date),year(b.event_date))

21 and a.user_id = b.user_id

22 where a.event_type in ('sign-in', 'like', 'comment')

23 and b.event_type in ('sign-in', 'like', 'comment')

24 and concat(month(a.event_date), '/', year(a.event_date)) = '6/2022'

25 and concat(1+month(b.event_date), '/', year(b.event_date)) = '6/2022'

26 group by month(a.event_date);

27

28

Data X

select month(a.event_date) as month, count(distinct a.user_id) as monthly_active_users

from user_actions a

Free 1

Input to filter result

month monthly_active_users

int bigint

	1	6	1

Q122. Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year.

However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the number of searches made last year and how many Google users fall into that bucket.

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

Hint- Write a subquery or common table expression (CTE) to generate a series of data (that's keyword for column) starting at the first search and ending at some point with an optional incremental value.

search_frequency Table:

Column Name	Type
searches	integer
num_users	integer

search_frequency Example Input:

searches	num_users
1	2
2	2
3	3
4	1

Example Output:

median
2.5

Solution:

```
with recursive seq as
(
    select searches, num_users, 1 as c from search_frequency
    union
    select searches, num_users, c+1 from seq where c < num_users
)
select round(avg(t.searches),1) as median from
(select searches, row_number() over(order by searches, c) as r1,
row_number() over(order by searches desc, c desc) as r2 from seq order by
searches) t
where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);
```

create-db-template.sql X [Preview] README.md

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

```
1      ▷ Execute
2  create database test;
3      ▷ Execute
4  use test;
5      ▷ Execute
6  create table search_frequency
7    (searches int,
8     num_users int);
9      ▷ Execute
10 insert into search_frequency values
11   (1, 2),
12   (2, 2),
13   (3, 3),
14   (4, 1);
15      ▷ Execute
16 ✓ 12 with recursive seq as
17  (
18    select searches, num_users, 1 as c from search_frequency
19    union
20    select searches, num_users, c+1 from seq where c < num_users
21  )
22  select round(avg(t.searches),1) as median from
23  (select searches, row_number() over(order by searches, c) as r1,
24   row_number() over(order by searches desc, c desc) as r2 from seq order by searches) t
25  where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);
```

Data X

with recursive seq as

Free 1 Input to filter result

Cost: 4ms < 1 Total 1

	median	newdecimal
	1	2.5

```

Solution: using cumulative sum
select round(avg(t1.searches),1) as median
from
(select t.searches, t.cumm_sum,
lag(cumm_sum,1,0) over(order by searches) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select searches, num_users,
sum(num_users) over(order by searches rows between unbounded preceding and
current row) as cumm_sum,
sum(num_users) over(order by searches rows between unbounded preceding and
unbounded following) as total
from search_frequency) t
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 >
t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);

```

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 search_frequency
4     (searches  int,
5      num_users  int
6    );
  ↓ Execute
7   insert into search_frequency values
8     (1, 2),
9     (2, 2),
10    (3, 3),
11    (4, 1);
  ↓ Execute
✓ 12  select round(avg(t1.searches),1) as median
13  from
14  (select t.searches, t.cumm_sum,
15    lag(cumm_sum,1,0) over(order by searches) as prev_cumm_sum,
16    case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
17    case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
18  from
19  (select searches, num_users,
20    sum(num_users) over(order by searches rows between unbounded preceding and current row) as cumm_sum,
21    sum(num_users) over(order by searches rows between unbounded preceding and unbounded following) as total
22  from search_frequency) t
23 ) t1
24 where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 > t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
25

```

search_frequency X

```

select round(avg(t1.searches),1) as median
from

 Input to filter result









































































































































































































































<input type="button" value="
```

Q123. Write a query to update the Facebook advertiser's status using the daily_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and daily_pay table has current information about their payment. Only advertisers who paid will show up in this table.

Output the user id and current payment status sorted by the user id.

Hint- Query the daily_pay table and check through the advertisers in this table. .

advertiser Table:

Column Name	Type
user_id	string
status	string

advertiser Example Input:

user_id	status
bing	NEW
yahoo	NEW
alibaba	EXISTING

`daily_pay` Table:

Column Name	Type
<code>user_id</code>	string
<code>paid</code>	decimal

`daily_pay` Example Input:

<code>user_id</code>	<code>paid</code>
yahoo	45.00
alibaba	100.00
target	13.00

Definition of advertiser status:

- New: users registered and made their first payment.
- Existing: users who paid previously and recently made a current payment.
- Churn: users who paid previously, but have yet to make any recent payment.
- Resurrect: users who did not pay recently but may have made a previous payment and have made payment again recently.

Example Output:

<code>user_id</code>	<code>new_status</code>
bing	CHURN
yahoo	EXISTING
alibaba	EXISTING

Bing's updated status is CHURN because no payment was made in the `daily_pay` table whereas Yahoo which made a payment is updated as EXISTING.

The dataset you are querying against may have different input & output - this is just an example! Read this before proceeding to solve the question

For better understanding of the advertiser's status, we're sharing with you a table of possible transitions based on the payment status.

#	Start	End	Condition
1	NEW	EXISTING	Paid on day T
2	NEW	CHURN	No pay on day T
3	EXISTING	EXISTING	Paid on day T
4	EXISTING	CHURN	No pay on day T
5	CHURN	RESURRECT	Paid on day T
6	CHURN	CHURN	No pay on day T
7	RESURRECT	EXISTING	Paid on day T
8	RESURRECT	CHURN	No pay on day T

- Row 2, 4, 6, 8: As long as the user has not paid on day T, the end status is updated to CHURN regardless of the previous status.
- Row 1, 3, 5, 7: When the user paid on day T, the end status is updated to either EXISTING or RESURRECT, depending on their previous state. RESURRECT is only possible when the previous state is CHURN. When the previous state is anything else, the status is updated to EXISTING.

Solution:

Conditions used in case when:

Previous Status	Condition	Next Status
New, Existing, Churn, Resurrect	Didn't pay on day T	Churn
New, Existing, Resurrect	Paid on day T	Existing
Churn	Paid on day T	Resurrect

```
select user_id, case
when status in ('NEW', 'EXISTING', 'CHURN', 'RESURRECT') and user_id not in (select user_id from daily_pay) then 'CHURN'
when status in ('NEW', 'EXISTING', 'RESURRECT') and user_id in (select user_id from daily_pay) then 'EXISTING'
when status = 'CHURN' and user_id in (select user_id from daily_pay) then
'RESURRECT'
end as new_status
from advertiser
order by user_id;
```

The screenshot shows a code editor with a MySQL script named 'create-db-template.sql'. The script creates two tables: 'advertiser' and 'daily_pay', and inserts data into them. It then performs a self-join on 'advertiser' and 'daily_pay' to calculate a new status column ('new_status') based on the user's previous status and their presence in the 'daily_pay' table. The output shows three rows: Alibaba (EXISTING), Bing (CHURN), and Yahoo (EXISTING).

```

1 create table advertiser
2 (user_id varchar(15),
3  status varchar(10)
4 );
5
6  create table daily_pay
7  (user_id varchar(15),
8   paid float
9 );
10
11 insert into advertiser values
12 ('Bing', 'NEW'),
13 ('Yahoo', 'NEW'),
14 ('Alibaba', 'EXISTING');
15
16 insert into daily_pay values
17 ('Yahoo', 45.00),
18 ('Alibaba', 100.00),
19 ('Target', 13.00);
20
21 select user_id, case
22 when status in ('NEW', 'EXISTING', 'CHURN', 'RESURRECT') and user_id not in (select user_id from daily_pay) then 'CHURN'
23 when status in ('NEW', 'EXISTING', 'RESURRECT') and user_id in (select user_id from daily_pay) then 'EXISTING'
24 when status = 'CHURN' and user_id in (select user_id from daily_pay) then 'RESURRECT'
25 end as new_status
26 from advertiser
27 order by user_id;

```

	user_id	new_status
1	Alibaba	EXISTING
2	Bing	CHURN
3	Yahoo	EXISTING

Q124. Amazon Web Services (AWS) is powered by fleets of servers. Senior management has requested data-driven solutions to optimise server usage.

Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

Level - Hard

Hint-

1. Calculate individual uptimes

2. Sum those up to obtain the uptime of the whole fleet, keeping in mind that the result must be output in units of full days

Assumptions:

- Each server might start and stop several times.
- The total time in which the server fleet is running can be calculated as the sum of each server's uptime.

server_utilization Table:

Column Name	Type
server_id	integer
status_time	timestamp
session_status	string

server_utilization Example Input:

server_id	status_time	session_status
1	08/02/2022 10:00:00	start
1	08/04/2022 10:00:00	stop
2	08/17/2022 10:00:00	start
2	08/24/2022 10:00:00	stop

Solution:

```
select sum(t.individual_uptime) as total_uptime_days
from
(
    select case when session_status = 'stop'
then
    timestampdiff(day, lag(status_time) over(partition by server_id order by
status_time), status_time) end as individual_uptime
from server_utilization
) t;
```

Q125. Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or a retry error that causes a credit card to be charged twice.

Using the transactions table, identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other. Count such repeated payments.

Level - Hard

Hint- Use Partition and order by

Assumptions:

- The first transaction of such payments should not be counted as a repeated payment. This means, if there are two transactions performed by a merchant with the same credit card and for the same amount within 10 minutes, there will only be 1 repeated payment.

transactions Table:

Column Name	Type
transaction_id	Integer
merchant_id	Integer
credit_card_id	Integer
amount	Integer
transaction_timestamp	datetime

transactions Example Input:

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

Example Output:

payment_count
1

Solution:

```
select sum(case when (unix_timestamp(t.next_transaction) -  
unix_timestamp(t.transaction_timestamp))/60 <= 10 then 1 else 0 end) as  
payment_count  
from  
(select transaction_timestamp,  
lead(transaction_timestamp,1) over(partition by merchant_id, credit_card_id,  
Amount order by transaction_timestamp) as next_transaction  
from transactions)t;
```

create-db-template.sql

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...  
    ▶ execute  
2  use test;  
   ▶ Execute  
3  create table transactions  
4  (transaction_id Int,  
5  merchant_id Int,  
6  credit_card_id Int,  
7  Amount Int,  
8  transaction_timestamp datetime  
9 );  
   ▶ Execute  
10 insert into transactions values  
11 (1, 101, 1, 100, '2022/09/25 12:00:00'),  
12 (2, 101, 1, 100, '2022/09/25 12:08:00'),  
13 (3, 101, 1, 100, '2022/09/25 12:28:00'),  
14 (4, 102, 2, 300, '2022/09/25 12:00:00'),  
15 (6, 102, 2, 400, '2022/09/25 14:00:00');  
16  
   ▶ Execute  
✓ 17 select sum(case when (unix_timestamp(t.next_transaction) - unix_timestamp(t.transaction_timestamp))/60 <= 10 then 1 else 0 end) as payment_count  
18 from  
19 (select transaction_timestamp,  
20 lead(transaction_timestamp,1) over(partition by merchant_id, credit_card_id, Amount order by transaction_timestamp) as next_transaction  
21 from transactions)t;  
22  
23  
24
```

transactions

```
select sum(case when (unix_timestamp(t.next_transaction) - unix_timestamp(t.transaction_timestamp))/60 <= 10 then 1 else 0 end) as payment_count
```

Free 1

Input to filter result

Payment count newdecimal

	1
1	1

Q126. DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because:

- the orders are being completed incorrectly (missing items, wrong order, etc.)
 - the orders aren't being received (wrong address, wrong drop off spot)
 - the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated_delivery_timestamp is automatically set to 30 minutes after the order_timestamp.

Hint- Use Where Clause and joins

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

orders Table:

Column Name	Type
order_id	integer
customer_id	integer
trip_id	integer
status	string ('completed successfully', 'completed incorrectly', 'never received')
order_timestamp	timestamp

orders Example Input:

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

trips Table:

Column Name	Type
dasher_id	integer
trip_id	integer
estimated_delivery_timestamp	timestamp
actual_delivery_timestamp	timestamp

trips Example Input:

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

customers Table:

Column Name	Type
customer_id	integer
signup_timestamp	timestamp

customers Example Input:

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

Example Output:

bad_experience_pct
75.00

Solution:

```
select round(avg(t1.bad_exp_pct_per_cust),2) as bad_exp_pct
from
(
    select t.customer_id, 100*sum(case when o.status <> 'completed successfully'
then 1 else 0 end)/count(*) as bad_exp_pct_per_cust
    from
    (
        select customer_id, signup_timestamp from customers where
month(signup_timestamp) = 6
    ) t
    inner join
    orders o
    on o.customer_id = t.customer_id
    where timestampdiff(day, t.signup_timestamp, o.order_timestamp) <= 13
    group by t.customer_id
) t1;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Editor Area:** Displays the SQL query for calculating the percentage of bad experiences per customer.
- Execution Log:** Shows the command history: "config > data > User > globalStorage > cweijan.mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...".
- Results Tab:** Shows the output of the query:

	bad_exp_pct
1	75.00
- Performance Metrics:** Below the results, it shows "Cost: 4ms" and "Total 1".

Q127.

Table: Scores

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day) is the primary key for this table.

A competition is held between the female team and the male team.

Each row of this table indicates that a player_name and with gender has scored score_point in someday.

Gender is 'F' if the player is in the female team and 'M' if the player is in the male team.

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

Return the result table ordered by gender and day in ascending order.

The query result format is in the following example.

Input:

Scores table:

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

Output:

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

M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

Explanation:

For the female team:

The first day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.

The second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.

The third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

The fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For the male team:

The first day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

The second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.

The third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.

The fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

The fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

Solution:

```
select gender, day,
sum(score_points) over(partition by gender order by day) as total
from Scores
group by gender, day
order by gender, day;
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Execute
3   create table Scores
4   (
5     player_name varchar(15),
6     gender  varchar(2),
7     day date,
8     score_points  Int,
9     primary key(gender, day)
10 );
  ↗ Execute
11  insert into Scores values
12  ('Aron', 'F', '2020-01-01', 17),
13  ('Alice', 'F', '2020-01-07', 23),
14  ('Bajrang', 'M', '2020-01-07', 7),
15  ('Khalil', 'M', '2019-12-25', 11),
16  ('Slaman', 'M', '2019-12-30', 13),
17  ('Joe', 'M', '2019-12-31', 3),
18  ('Jose', 'M', '2019-12-18', 2),
19  ('Priya', 'F', '2019-12-31', 23),
20  ('Priyanka', 'F', '2019-12-30', 17);
  ↗ Execute
21  select gender, day, sum(score_points) over(partition by gender order by day) as total
22  from Scores
23  group by gender, day
24  order by gender, day;

```

Scores

	gender	day	total
	varchar	date	newdecimal
1	F	2019-12-30	17
2	F	2019-12-31	40
3	F	2020-01-01	57
4	F	2020-01-07	80
5	M	2019-12-18	2
6	M	2019-12-25	13
7	M	2019-12-30	26
8	M	2019-12-31	29
9	M	2020-01-07	36

Q128.

Table Person:

Column Name	Type
id	int
name	varchar
phone_number	varchar

id is the primary key for this table.

Each row of this table contains the name of a person and their phone number.

Phone number will be in the form 'xxx-yyyyyyy' where xxx is the country code (3 characters) and yyyyyyy is the phone number (7 characters) where x and y are digits. Both can contain leading zeros.

Table Country:

Column Name	Type
name	varchar
country_code	varchar

country_code is the primary key for this table.

Each row of this table contains the country name and its code. country_code will be in the form 'xxx' where x is digits.

Table Calls:

Column Name	Type
caller_id	int
callee_id	int
duration	int

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

Each row of this table contains the caller id, callee id and the duration of the call in minutes. $\text{caller_id} \neq \text{callee_id}$

A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Person table:

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

Country table:

name	country_code
Peru	51
Israel	972
Morocco	212
Germany	49
Ethiopia	251
Ethiopia	251

Calls table:

caller_id	callee_id	duration
1	9	33
2	9	4

1	2	59
3	12	102
3	12	330
12	3	5
7	9	13
7	1	3
9	7	1
1	7	7

Output:

country
Peru

Explanation:

The average call duration for Peru is $(102 + 102 + 330 + 330 + 5 + 5) / 6 = 145.666667$

The average call duration for Israel is $(33 + 4 + 13 + 13 + 3 + 1 + 1 + 7) / 8 = 9.37500$

The average call duration for Morocco is $(33 + 4 + 59 + 59 + 3 + 7) / 6 = 27.5000$

Global call duration average = $(2 * (33 + 4 + 59 + 102 + 330 + 5 + 13 + 3 + 1 + 7)) / 20 = 55.70000$

Since Peru is the only country where the average call duration is greater than the global average, it is the only recommended country.

Solution:

```

select t3.Name from
(
select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
avg(t1.duration) over() as global_average
from
((select cl.caller_id as id, cl.duration
from Calls cl)
union
(select cl callee_id as id, cl.duration
from Calls cl)) t1
left join
(select p.id, c.Name from Person p
left JOIN
Country c
ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
ON t1.id = t2.id) t3
where t3.avg_call_duration > global_average
group by t3.Name;

```

SQLQuery2.sql - LAP...ARTA.test (sa (65))*

```

('Morocco', 212),
('Germany', 49),
('Ethiopia', 251);
insert into Calls values
(1, 9, 33),
(2, 9, 4),
(1, 2, 59),
(3, 12, 102),
(3, 12, 330),
(12, 3, 5),
(7, 9, 13),
(7, 1, 3),
(9, 7, 1),
(1, 7, 7);
select t3.Name from
(
  select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg_call_duration,
  avg(t1.duration) over() as global_average
  from
  ((select cl.caller_id as id, cl.duration
  from Calls cl)
  union
  (select cl.callee_id as id, cl.duration
  from Calls cl)) t1
  left join
  (select p.id, c.Name from Person p
  left JOIN
  Country c
  ON cast(left(p.phone_number,3) as int) = cast(c.country_code as int)) t2
  ON t1.id = t2.id) t3
  where t3.avg_call_duration > global_average
  group by t3.Name;

```

00 %

Results Messages

Name
Peru

Query executed successfully.

Q129.

Table: Numbers

Column Name	Type
num	int
frequency	int

num is the primary key for this table.

Each row of this table shows the frequency of a number in the database.

The median is the value separating the higher half from the lower half of a data sample.

Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point.

The query result format is in the following example.

Input:

Numbers table:

num	frequency
0	7
1	1
2	3
3	1

Output:

median
0

Explanation:

If we decompose the Numbers table, we will get [0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3], so the median is $(0 + 0) / 2 = 0$.

Solution:

```
with recursive seq as
(
    select num, frequency, 1 as c from Numbers
    union
    select num, frequency, c+1 from seq where c < frequency
)
select round(avg(t.num),1) as median
from
(
    select num, row_number() over(order by num, c) as r1,
    row_number() over(order by num desc, c desc) as r2 from seq order by num
) t
where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);
```

create-db-template.sql X [Preview] README.md

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

```

1      ▷ Execute
2  create database test;
3      ▷ Execute
4  use test;
5      ▷ Execute
6  create table Numbers
7  (num int primary key,
8   frequency int);
9      ▷ Execute
10 insert into Numbers values
11 (0, 7),
12 (1, 1),
13 (2, 3),
14 (3, 1);
15      ▷ Execute
16 with recursive seq as
17 (
18     select num, frequency, 1 as c from Numbers
19     union
20     select num, frequency, c+1 from seq where c < frequency
21 )
22 select round(avg(t.num),1) as median
23 from
24 (
25     select num, row_number() over(order by num, c) as r1,
26           row_number() over(order by num desc, c desc) as r2 from seq order by num
27 ) t
28 where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);
29

```

Data X

with recursive seq as

<input checked="" type="checkbox"/>	<input type="checkbox"/>	Input to filter result
		1
	Cost: 2ms	1
	Total	1

median
newdecimal

	1	0.0

Solution: using cumulative sum

```

select round(avg(t1.num),1) as median
from
(select t.num, t.cumm_sum,
lag(cumm_sum,1,0) over(order by num) as prev_cumm_sum,
case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
from
(select num, frequency,
sum(frequency) over(order by num rows between unbounded preceding and current
row) as cumm_sum,
sum(frequency) over(order by num rows between unbounded preceding and unbounded
following) as total

from Numbers) t

```

```
) t1
where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 >
t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
```

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 Numbers
4 (num Int,
5 frequency Int,
6 primary key(num)
7);
 ▷ Execute
8 insert into Numbers values
9 (0, 7),
10 (1, 1),
11 (2, 3),
12 (3, 1);
13
 ▷ Execute
✓ 14 select round(avg(t1.num),1) as median
15 from
16 (select t.num, t.cumm_sum,
17 lag(cumm_sum,1,0) over(order by num) as prev_cumm_sum,
18 case when total % 2 = 0 then total/2 else (total+1)/2 end as pos1,
19 case when total % 2 = 0 then (total/2)+1 else (total+1)/2 end as pos2
20 from
21 (select num, frequency,
22 sum(frequency) over(order by num rows between unbounded preceding and current row) as cumm_sum,
23 sum(frequency) over(order by num rows between unbounded preceding and unbounded following) as total
24 from Numbers) t
25) t1
26 where (t1.pos1 > t1.prev_cumm_sum and t1.pos1 <= t1.cumm_sum) or (t1.pos2 > t1.prev_cumm_sum and t1.pos2 <= t1.cumm_sum);
27

Numbers X

select round(avg(t1.num),1) as median
from

Free 1

Input to filter result

Cost: 5ms < 1 Total 1

	median	newdecimal
1	0.0	

Q130.

Table: Salary

Column Name	Type

id	int
employee_id	int
amount	int
pay_date	date

id is the primary key column for this table.

Each row of this table indicates the salary of an employee in one month.

employee_id is a foreign key from the Employee table.

Table: Employee

Column Name	Type
employee_id	int
department_id	int

employee_id is the primary key column for this table.

Each row of this table indicates the department of an employee.

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

Return the result table in any order.

The query result format is in the following example.

Input:

Salary table:

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

Employee table:

employee_id	department_id
1	1
2	2
3	2

Output:

pay_month	department_id	comparison
2017-02	1	same
2017-03	1	higher
2017-02	2	same
2017-03	2	lower

Explanation:

In March, the company's average salary is $(9000+6000+10000)/3 = 8333.33\dots$

The average salary for department '1' is 9000, which is the salary of employee_id '1' since there is only one employee in this department. So the comparison result is 'higher' since $9000 > 8333.33$ obviously.

The average salary of department '2' is $(6000 + 10000)/2 = 8000$, which is the average of employee_id '2' and '3'. So the comparison result is 'lower' since $8000 < 8333.33$.

With the same formula for the average salary comparison in February, the result is 'same' since both the departments '1' and '2' have the same average salary with the company, which is 7000.

Solution:

```
select distinct concat(year(t.pay_date), ' - ', month(t.pay_date)) as pay_month,
t.department_id,
case
when monthly_department_avg_salary > monthly_average_salary then 'higher'
when monthly_department_avg_salary < monthly_average_salary then 'lower'
else 'same'
end as Comparison
from
(select s.pay_date, e.department_id,
avg(s.amount) over(partition by month(s.pay_date), e.department_id) as
monthly_department_avg_salary,
```

```

avg(s.amount) over(partition by month(s.pay_date)) as monthly_average_salary
from Salary s
left join
Employee e
on s.employee_id = e.employee_id) t
order by t.department_id;

```

create-db-template.sql

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
  ↗ Execute
10  create table Employee
11  (employee_id      Int,
12  department_id     Int,
13  primary key(employee_id)
14  );
  ↗ Execute
15  insert into Salary values
16  (1, 1, 9000, '2017/03/31'),
17  (2, 2, 6000, '2017/03/31'),
18  (3, 3, 10000, '2017/03/31'),
19  (4, 1, 7000, '2017/02/28'),
20  (5, 2, 6000, '2017/02/28'),
21  (6, 3, 8000, '2017/02/28');
  ↗ Execute
22  insert into Employee values
23  (1, 1),
24  (2, 2),
25  (3, 2);
  ↗ Execute
26  select distinct concat(year(t.pay_date), '-', month(t.pay_date)) as pay_month,
27  t.department_id,
28  case
29  when monthly_department_avg_salary > monthly_average_salary then 'higher'
30  when monthly_department_avg_salary < monthly_average_salary then 'lower'
31  else 'same'
32  end as Comparison
33  from
34  (select s.pay_date, e.department_id,
35  avg(s.amount) over(partition by month(s.pay_date), e.department_id) as monthly_department_avg_salary,
36  avg(s.amount) over(partition by month(s.pay_date)) as monthly_average_salary
37  from Salary s
38  left join
39  Employee e
40  on s.employee_id = e.employee_id) t
41  order by t.department_id;
  ↗

```

Person

	pay_month	department_id	Comparison
1	2017-2	1	same
2	2017-3	1	higher
3	2017-2	2	same
4	2017-3	2	lower

Q131.

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.

The install date of a player is the first login day of that player.

We define day one retention of some date x to be the number of players whose install date is x and they logged back in on the day right after x, divided by the number of players whose install date is x, rounded to 2 decimal places.

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

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	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-01	0
3	4	2016-07-03	5

Output:

install_dt	installs	Day1_retention
2016-03-01	2	0.5
2017-06-25	1	0

Explanation:

Player 1 and 3 installed the game on 2016-03-01 but only player 1 logged back in on 2016-03-02 so the day 1 retention of 2016-03-01 is $1 / 2 = 0.50$

Player 2 installed the game on 2017-06-25 but didn't log back in on 2017-06-26 so the day 1 retention of 2017-06-25 is $0 / 1 = 0.00$

Solution:

```
select t1.install_dt, count(player_id) as installs,
round(count(t1.next_install)/count(t1.player_id),2) as Day1_retention
from
(
    select t.player_id, t.install_dt, a.event_date as next_install
    from
        (
            select player_id, min(event_date) as install_dt
            from Activity
            group by player_id
        ) t
        left join
        Activity a
        on t.player_id = a.player_id and a.event_date = t.install_dt + 1
) t1
group by install_dt;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** The code block above is pasted into the query editor.
- Execution Result:** The result set is displayed below the editor:

install_dt	installs	Day1_retention
2016-03-01	2	0.50
2017-06-25	1	0.00
- Performance Metrics:** The bottom status bar indicates a cost of 6ms and a total of 2 rows processed.

Q132.

Table: Players

Column Name	Type
player_id	int
group_id	int

player_id is the primary key of this table.

Each row of this table indicates the group of each player.

Table: Matches

Column Name	Type
match_id	int
first_player	int
second_player	int
first_score	int
second_score	int

match_id is the primary key of this table.

Each row is a record of a match, first_player and second_player contain the player_id of each match. first_score and second_score contain the number of points of the first_player and second_player respectively.

You may assume that, in each match, players belong to the same group.

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;

```

```

create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
    ▷ Execute
4   use test;
    ▷ Execute
5   create table Players
6   (
7     player_id  Int primary key,
8     group_id   Int
9   );
    ▷ Execute
10  create table Matches
11  (
12    |   match_id   Int primary key,
13    first_player  Int,
14    second_player Int,
15    first_score  Int,
16    second_score Int
17  );
    ▷ Execute
18  insert into Players values
19  (15, '1'),
20  (25, '1'),
21  (30, '1'),
22  (45, '1'),
23  (10, '2'),
24  (35, '2'),
25  (50, '2'),
26  (20, '3'),
27  (40, '3');
    ▷ Execute
28  insert into Matches values
29  (1, 15, 45, 3, 0),
30  (2, 30, 25, 1, 2),
31  (3, 30, 15, 2, 0),
32  (4, 40, 20, 5, 2),
33  (5, 35, 50, 1, 1);
    ▷ Execute
34  select t2.group_id, t2.player_id from
35  (
36    |   select t1.group_id, t1.player_id,
37    dense_rank() over(partition by group_id order by score desc, player_id) as r
38    from
39    (
40      |   select p.*, case when p.player_id = m.first_player then m.first_score
41      when p.player_id = m.second_player then m.second_score
42      end as score
43      from
44      Players p, Matches m
45      where player_id in (first_player, second_player)
46      |   ) t1
47    ) t2
48  where r = 1;

```

Players X

```

select t2.group_id, t2.player_id from

```

Input to filter result Free 1 Cost: 12ms < 1 > Total 3

	group_id	player_id
1	1	15
2	2	35
3	3	40

Table: Student

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.
student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

A quiet student is the one who took at least one exam and did not score the high or the low score.

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

Return the result table ordered by student_id.

The query result format is in the following example.

Input:

Student table:

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

Exam table:

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

Output:

student_id	student_name
2	Jade

Explanation:

For exam 1: Student 1 and 3 hold the lowest and high scores respectively.

For exam 2: Student 1 holds both the highest and lowest score.

For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively. Students 2 and 5 have never got the highest or lowest in any of the exams.

Since student 5 is not taking any exam, he is excluded from the result. So, we only return the information of Student 2.

Solution:

```
select t.student_id, t.student_name from
(select s.student_name, s.student_id, count(e.student_id) over(partition by
student_name) as exams_given,
case when e.score > min(e.score) over(partition by e.exam_id) and e.score <
max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
# 1 means student is quiet, 0 means student is not quiet
from Exam e
left join
Student s
on e.student_id = s.student_id)t
group by t.student_name, t.student_id, t.exams_given
having sum(t.quiet) = t.exams_given
# sum(quiet) will give the total number of exams in which student is quiet
```

The screenshot shows the MySQL Workbench interface with two tabs: 'create-db-template.sql' and 'Data'.

In the 'create-db-template.sql' tab, the code for creating the 'Student' and 'Exam' tables and inserting data is shown. The 'Exam' table has columns (exam_id, student_id, score). The 'Student' table has columns (student_id, student_name).

In the 'Data' tab, the solution SQL query is executed. The results show the student names and their counts of exams taken, along with a column for quiet status (1 for quiet, 0 for not quiet). The output table includes columns: student_id, student_name, exams_given, and quiet.

student_id	student_name	exams_given	quiet
1	Jade	2	1

Q134.

Table: Student

Column Name	Type
student_id	int
student_name	varchar

student_id is the primary key for this table.
student_name is the name of the student.

Table: Exam

Column Name	Type
exam_id	int
student_id	int
score	int

(exam_id, student_id) is the primary key for this table.
Each row of this table indicates that the student with student_id had a score points in the exam with id exam_id.

A quiet student is the one who took at least one exam and did not score the high or the low score.
Write an SQL query to report the students (student_id, student_name) being quiet in all exams. Do not return the student who has never taken any exam.
Return the result table ordered by student_id.
The query result format is in the following example.

Input: Student

table:

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

Exam table:

exam_id	student_id	score
10	1	70
10	2	80
10	3	90
20	1	80

30	1	70
30	3	80
30	4	90
40	1	60
40	2	70
40	4	80

Output:

student_id	student_name
2	Jade

Explanation:

For exam 1: Student 1 and 3 hold the lowest and high scores respectively. For exam 2: Student 1 holds both the highest and lowest score.

For exam 3 and 4: Student 1 and 4 hold the lowest and high scores respectively. Students 2 and 5 have never got the highest or lowest in any of the exams.

Since student 5 is not taking any exam, he is excluded from the result. So, we only return the information of Student 2.

```
select s.student_name, s.student_id, count(e.student_id) over(partition by
student_name) as exams_given,
case when e.score > min(e.score) over(partition by e.exam_id) and e.score <
max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
# 1 means student is quiet, 0 means student is not quiet
from Exam e
left join
Student s
on e.student_id = s.student_id)t
group by t.student_name, t.student_id, t.exams_given
having sum(t.quiet) = t.exams_given
# sum(quiet) will give the total number of exams in which student is quiet
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
5   student_name  varchar(15),
6   primary key(student_id)
7 );
  ↴ Execute
8   create table Exam
9   (exam_id  int,
10  student_id  int,
11  score  int,
12  primary key(exam_id, student_id)
13 );
  ↴ Execute
14  insert into Student values
15  (1, 'Daniel'),
16  (2, 'Jade'),
17  (3, 'Stella'),
18  (4, 'Jonathan'),
19  (5, 'Will');
  ↴ Execute
20  insert into Exam values
21  (10, 1, 70),
22  (10, 2, 80),
23  (10, 3, 90),
24  (20, 1, 80),
25  (30, 1, 70),
26  (30, 3, 80),
27  (30, 4, 90),
28  (40, 1, 60),
29  (40, 2, 70),
30  (40, 4, 80);
  ↴ Execute
✓ 31  select t.student_id, t.student_name from
32  (select s.student_name, s.student_id, count(e.student_id) over(partition by student_name) as exams_given,
33  case when e.score > min(e.score) over(partition by e.exam_id) and e.score < max(e.score) over(partition by e.exam_id) then 1 else 0 end as quiet
34  # 1 means student is quiet, 0 means student is not quiet
35  from Exam e
36  left join
37  Student s
38  on e.student_id = s.student_id)t
39  group by t.student_name, t.student_id, t.exams_given
40  having sum(t.quiet) = t.exams_given
41  # sum(quiet) will give the total number of exams in which student is quiet
42 ;

```

Data

student_id	student_name	
1	Jade	

Q135.

Table: UserActivity

Column Name	Type
username	varchar
activity	varchar
startDate	Date
endDate	Date

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

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input: UserActivity

table:

username	activity	startDate	endDate
Alice	Travel	2020-02-12	2020-02-20

Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

Output:

username	activity	startDate	endDate
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

Solution:

```
with new as
(select t.username, t.activity, t.startDate, t.endDate
from(
    select username, activity, startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r
from UserActivity)t
where r = 2
)
select * from new
union
select n.username, n.activity, n.startDate, n.endDate
from(
    select username, activity, startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r
from UserActivity)n
where r = 1 and username not in (select username from new);
```

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
    ▷ Execute
4   create table UserActivity
5     (username  varchar(15),
6      activity   varchar(15),
7      startDate  Date,
8      endDate   Date
9    );
    ▷ Execute
10  insert into UserActivity values
11  ('Alice',  'Travel',  '2020-02-12',  '2020-02-20'),
12  ('Alice',  'Dancing', '2020-02-21',  '2020-02-23'),
13  ('Alice',  'Travel',  '2020-02-24',  '2020-02-28'),
14  ('Bob',    'Travel',  '2020-02-11',  '2020-02-18');
    ▷ Execute
✓ 15  with new as
16    (select t.username, t.activity, t.startDate, t.endDate
17     from(
18       |  | select username, activity, startDate, endDate,
19       |  | dense_rank() over(partition by username order by endDate desc) as r
20       |  | from UserActivity)t
21     where r = 2
22   )
23   select * from new
24   union
25   select n.username, n.activity, n.startDate, n.endDate
26   from(
27     |  | select username, activity, startDate, endDate,
28     |  | dense_rank() over(partition by username order by endDate desc) as r
29     |  | from UserActivity)n
30   where r = 1 and username not in (select username from new);
31
Data X
with new as
(select t.username, t.activity, t.startDate, t.endDate
Free 1
+ 🔒 Q Input to filter result
username varchar activity varchar startDate date endDate date
1 Alice Dancing 2020-02-21 2020-02-23
2 Bob Travel 2020-02-11 2020-02-18
Cost: 5ms < 1 > Total 2

```

Q136.

Table: UserActivity

Column Name	Type
username	varchar
activity	varchar
startDate	Date

endDate	Date
---------	------

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

This table contains information about the activity performed by each user in a period of time. A person with a username performed an activity from startDate to endDate.

Write an SQL query to show the second most recent activity of each user.

If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order.

The query result format is in the following example.

Input: UserActivity

table:

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

Output:

username	activity	startDate	endDate
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

Explanation:

The most recent activity of Alice is Travel from 2020-02-24 to 2020-02-28, before that she was dancing from 2020-02-21 to 2020-02-23.

Bob only has one record, we just take that one.

Solution:

```
with new as
(select t.username, t.activity, t.startDate, t.endDate
from(
    select username, activity, startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r
from UserActivity)t
where r = 2
)
select * from new
union
select n.username, n.activity, n.startDate, n.endDate
from(
    select username, activity, startDate, endDate,
dense_rank() over(partition by username order by endDate desc) as r
from UserActivity)n
where r = 1 and username not in (select username from new);
```

```

create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template
    ▷ Execute
4   create table UserActivity
5     (username  varchar(15),
6      activity   varchar(15),
7      startDate  Date,
8      endDate Date
9    );
    ▷ Execute
10  insert into UserActivity values
11  ('Alice',  'Travel',  '2020-02-12',  '2020-02-20'),
12  ('Alice',  'Dancing', '2020-02-21',  '2020-02-23'),
13  ('Alice',  'Travel',  '2020-02-24',  '2020-02-28'),
14  ('Bob',    'Travel',  '2020-02-11',  '2020-02-18');
    ▷ Execute
✓ 15  with new as
16    (select t.username, t.activity, t.startDate, t.endDate
17     from(
18       |  select username, activity, startDate, endDate,
19       |  dense_rank() over(partition by username order by endDate desc) as r
20     from UserActivity)t
21    where r = 2
22  )
23  select * from new
24  union
25  select n.username, n.activity, n.startDate, n.endDate
26  from(
27    |  select username, activity, startDate, endDate,
28    |  dense_rank() over(partition by username order by endDate desc) as r
29    |  from UserActivity)n
30  where r = 1 and username not in (select username from new);
31
Data X
with new as
(select t.username, t.activity, t.startDate, t.endDate
Free 1

⊕ 🔒 Q Cost: 5ms < 1 > Total 2
Filter Sort Reset Columns Rows
Check Q username activity startDate endDate

|  | 1 | Alice | Dancing | 2020-02-21 | 2020-02-23 |
|--|---|-------|---------|------------|------------|
|  | 2 | Bob   | Travel  | 2020-02-11 | 2020-02-18 |


```

Q137.

Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

Input Format

The EMPLOYEES table is described as follows:

Column	Type
<i>ID</i>	<i>Integer</i>
<i>Name</i>	<i>String</i>
<i>Salary</i>	<i>Integer</i>

Note: Salary is per month.

Constraints

$1000 < \text{salary} < 10^5$

Sample Input

<i>ID</i>	<i>Name</i>	<i>Salary</i>
1	Kristeen	1420
2	Ashley	2006
3	Julia	2210
4	Maria	3000

Sample Output

2061

Explanation

The table below shows the salaries without zeros as they were entered by Samantha:

<i>ID</i>	<i>Name</i>	<i>Salary</i>
1	<i>Kristeen</i>	142
2	<i>Ashley</i>	26
3	<i>Julia</i>	221
4	<i>Maria</i>	3

Samantha computes an average salary of 98.00 . The actual average salary is 2159.00.

The resulting error between the two calculations is $2159.00 - 98.00 = 2061.00$. Since it is equal to the integer 2061, it does not get rounded up.

Solution:

```
select ceil(avg(salary) - avg(replace(salary, 0, '')))  
as calculation_difference  
from Employees;
```

The screenshot shows a code editor with a MySQL script and a results table.

create-db-template.sql

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1
    ⚡ Active Connection | ▶ Execute
1   create database test;
    ▶ Execute
2   use test;
    ▶ Execute
3   create table Employees
4       (id int,
5        name varchar(15),
6        salary int);
    ▶ Execute
7   insert into Employees values
8       (1, 'Kristeen', 1420),
9       (2, 'Ashley', 2006),
10      (3, 'Julia', 2210),
11      (4, 'Maria', 3000);
    ▶ Execute
12  select ceil(avg(salary) - avg(replace(salary, 0, '')))
13  as calculation_difference
14  from Employees;
15
16
17
18

```

Employees

		calculation_difference
	1	2061

Q138.

We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

Level - Easy

Hint - Use Aggregation functions

Input Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is the their monthly salary.

Sample Input

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

Sample Output

69952 1

Explanation:

The table and earnings data is depicted in the following diagram:

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

The maximum earnings value is 69952. The only employee with earnings= 69952 is Kimberly, so we print the maximum earnings value (69952) and a count of the number of employees who have earned \$69952 (which is 1) as two space-separated values.

```
select concat(max(t.earnings), ' ',  
sum(case  
    when earnings = max_salary then 1  
    else 0  
end)) as Output  
from  
(  
    select max(salary*months) over() as max_salary,  
salary*months as earnings  
from  
Employee) t;
```

```
create-db-template.sql X [Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > ■

2  use test;
   ▷ Execute
3  create table Employee
4    (employee_id int,
5     name varchar(12),
6     months int,
7     salary int);
   ▷ Execute
8  insert into Employee values
9    (12228, 'Rose', 15, 1968),
10   (33645, 'Angela', 1, 3443),
11   (45692, 'Frank', 17, 1608),
12   (56118, 'Patrick', 7, 1345),
13   (59725, 'Lisa', 11, 2330),
14   (74197, 'Kimberly', 16, 4372),
15   (78454, 'Bonnie', 8, 1771),
16   (83565, 'Michael', 6, 2017),
17   (98607, 'Todd', 5, 3396),
18   (99989, 'Joe', 9, 3573);
19
20  ▷ Execute
21  select concat(max(t.earnings), ' ',
22  sum(case
23    | when earnings = max_salary then 1
24    | else 0
25    | end)) as Output
26  from
27  (
28    select max(salary*months) over() as max_salary,
29    salary*months as earnings
30    from
31    Employee) t;
```

Employee X

```
select concat(max(t.earnings), ' ',
sum(case
when earnings = max_salary then 1
else 0
end)) as Output
from
(
select max(salary*months) over() as max_salary,
salary*months as earnings
from
Employee) t;
```

Free 1

	Output	varchar
1	69952	1

Cost: 4ms < 1

Q139.

Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).

Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

Level - Medium

There are a total of [occupation_count] [occupation]s.

2. where [occupation_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.

Note: There will be at least two entries in the table for each type of occupation.

Input Format

The OCCUPATIONS table is described as follows:

<i>Column</i>	<i>Type</i>
<i>Name</i>	<i>String</i>
<i>Occupation</i>	<i>String</i>

Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor.

Sample Input

An OCCUPATIONS table that contains the following records:

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

Sample Output

Ashely(P)

Christeen(P)

Jane(A)

Jenny(D)

Julia(A)

Ketty(P)

Maria(A)

Meera(S)

Priya(S)

Samantha(D)

There are a total of 2 doctors.

There are a total of 2 singers.

There are a total of 3 actors.

There are a total of 3 professors.

Hint -

The results of the first query are formatted to the problem description's specifications.

The results of the second query are ascendingly ordered first by number of names corresponding to each profession ($2 \leq 2 \leq 3 \leq 3$), and then alphabetically by profession (doctor \leq singer , and actor \leq professor).

```

Solution:
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
order by name;
select
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation),
's.') as occupation_count
from Occupations
group by occupation
order by count(occupation), occupation;

```

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

```

3  create table Occupations
4    (name varchar(15),
5     occupation varchar(15));
  ↴ Execute
6  insert into Occupations values
7    ('Samantha', 'Doctor'),
8    ('Julia', 'Actor'),
9    ('Maria', 'Actor'),
10   ('Meera', 'Singer'),
11   ('Ashley', 'Professor'),
12   ('Ketty', 'Professor'),
13   ('Christeen', 'Professor'),
14   ('Jane', 'Actor'),
15   ('Jenny', 'Doctor'),
16   ('Priya', 'Singer');
  ↴ Execute
✓ 17  select concat(name, '(', left(occupation,1),')') as name_occupation
18  from Occupations
19  order by name;
  ↴ Execute
20  select
21    concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's') as occupation_count
22  from Occupations
23  group by occupation
24  order by count(occupation), occupation;
  ↴ Execute

```

Occupations X

```
select concat(name, '(', left(occupation,1),')') as name_occupation
from Occupations
```

Q Input to filter result 1 Cost: 4ms < 1 > Total 10

		name_occupation
		varchar
1		Ashley(P)
2		Christeen(P)
3		Jane(A)
4		Jenny(D)
5		Julia(A)
6		Ketty(P)
7		Maria(A)
8		Meera(S)
9		Priya(S)
10		Samantha(D)

create-db-template.sql ×

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

```
3  create table Occupations
4  (name varchar(15),
5  occupation varchar(15));
    ▷ Execute
6  insert into Occupations values
7  ('Samantha', 'Doctor'),
8  ('Julia', 'Actor'),
9  ('Maria', 'Actor'),
10 ('Meera', 'Singer'),
11 ('Ashley', 'Professor'),
12 ('Ketty', 'Professor'),
13 ('Christeen', 'Professor'),
14 ('Jane', 'Actor'),
15 ('Jenny', 'Doctor'),
16 ('Priya', 'Singer');
    ▷ Execute
17 select concat(name, '(', left(occupation,1),')') as name_occupation
18 from Occupations
19 order by name;
    ▷ Execute
✓ 20 select
21 concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count
22 from Occupations
23 group by occupation
24 order by count(occupation), occupation;
    ▷ Execute
```

Occupations X

select
concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation_count

Free 1 Input to filter result ↻ 1 + + - ⚡ ⬆ ⬇ ⏪ ⏩ Cost: 5ms < 1 Total 4

	occupation_count
1	There are a total of 2 doctors.
2	There are a total of 2 singers.
3	There are a total of 3 actors.
4	There are a total of 3 professors.

Q140 .

Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print NULL when there are no more names corresponding to an occupation.

Input Format

The OCCUPATIONS table is described as follows:

Column	Type
Name	String
Occupation	String

Occupation will only contain one of the following values: Doctor, Professor, Singer or Actor.

Sample Input

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

Sample Output

Jenny Ashley Meera Jane

Samantha Christeen Priya Julia

NULL Ketty NULL Maria

Hint -

The first column is an alphabetically ordered list of Doctor names.

The second column is an alphabetically ordered list of Professor names. The third column is an alphabetically ordered list of Singer names.

The fourth column is an alphabetically ordered list of Actor names.

The empty cell data for columns with less than the maximum number of names per occupation (in this case, the Professor and Actor columns) are filled with NULL values.

Solution:

```
select max(case Occupation when 'Doctor' then Name end) as Doctors,
       max(case Occupation when 'Professor' then Name end) as Professors,
       max(case Occupation when 'Singer' then Name end) as Singers,
       max(case Occupation when 'Actor' then Name end) as Actors
  from (
    select occupation, name,
           row_number() over(partition by Occupation order by name) as r
      from Occupations
     ) t
 group by r;
```

The screenshot shows a MySQL Workbench interface. The top part is a code editor with a tab labeled "create-db-template.sql". The code is a SQL script to create a table "Occupations" and insert data, followed by a query to group the data by occupation. The bottom part is a results grid titled "Occupations" with three rows of data.

```

2  use test;
3  create table Occupations
4  (name varchar(15),
5  occupation varchar(15));
6  insert into Occupations values
7  ('Samantha', 'Doctor'),
8  ('Julia', 'Actor'),
9  ('Maria', 'Actor'),
10 ('Meera', 'Singer'),
11 ('Ashley', 'Professor'),
12 ('Ketty', 'Professor'),
13 ('Christeen', 'Professor'),
14 ('Jane', 'Actor'),
15 ('Jenny', 'Doctor'),
16 ('Priya', 'Singer');
17 select max(case Occupation when 'Doctor' then Name end) as Doctors,
18       max(case Occupation when 'Professor' then Name end) as Professors,
19       max(case Occupation when 'Singer' then Name end) as Singers,
20       max(case Occupation when 'Actor' then Name end) as Actors
21   from
22   (
23     select occupation, name,
24     row_number() over(partition by Occupation order by name) as r
25   from Occupations
26   ) t
27 group by r;
28

```

	Doctors	Professors	Singers	Actors
1	Jenny	Ashley	Meera	Jane
2	Samantha	Christeen	Priya	Julia
3	(NULL)	Ketty	(NULL)	Maria

Q141.

You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.

Column	Type
N	Integer
P	Integer

Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

- Root: If node is root node.
- Leaf: If node is leaf node.
- Inner: If node is neither root nor leaf node.

Sample Input

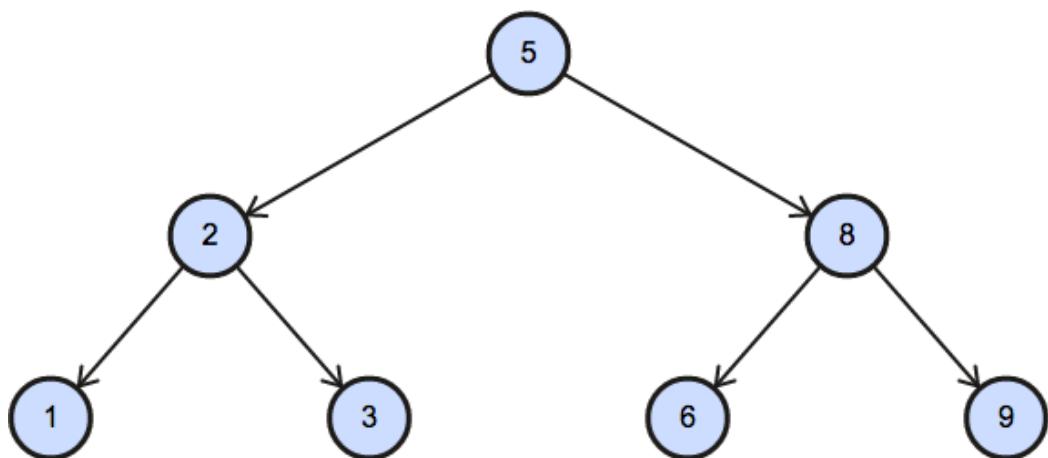
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	<i>null</i>

Sample Output

1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
8 Inner
9 Leaf

Explanation

The Binary Tree below illustrates the sample:



Solution:

```
select
(
    case
        when P is NULL then 'Root'
        when N not in (select distinct P from BST where P is not null) then 'Leaf'
    else 'Inner'
    end
) as Node_Type
from BST
order by N;
```

The screenshot shows a MySQL Workbench interface. At the top, there's a tab bar with 'create-db-template.sql' and '[Preview] README.md'. Below the tabs, the connection details are shown: config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template. A status indicator shows 'Active Connection | Execute'.

```

1 create database test;
2 use test;
3 create table BST
4 (N int,
5 P int);
6 insert into BST values
7 (1,2),
8 (3,2),(6,8),(9,8),(2,5),(8,5),(5, null);
9 select
10 (
11     case
12         when P is NULL then 'Root'
13         when N not in (select distinct P from BST where P is not null) then 'Leaf'
14     else 'Inner'
15 end
16 ) as Node_Type
17 from BST
18 order by N;
19

```

Below the script, a result set titled 'BST' is displayed. It contains a single column 'Node_Type' with 7 rows:

	Node_Type
1	Leaf
2	Inner
3	Leaf
4	Root
5	Leaf
6	Inner
7	Leaf

Q142 .

Amber's conglomerate corporation just acquired some new companies. Each of the companies

```

Founder
↓
Lead Manager
↓
Senior Manager
↓
Manager
↓
Employee

```

follows this hierarchy:

Given the table schemas below, write a query to print the company_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code.

Level - Medium

Note:

- The tables may contain duplicate records.

- The company_code is string, so the sorting should not be numeric. For example, if the company_codes are C_1, C_2, and C_10, then the ascending company_codes will be C_1, C_10, and C_2.

Input Format

The following tables contain company data:

- Company: The company_code is the code of the company and founder is the founder of the

Column	Type
company_code	String
founder	String

company.

- Lead_Manager: The lead_manager_code is the code of the lead manager, and the

Column	Type
lead_manager_code	String
company_code	String

company_code is the code of the working company.

- Senior_Manager: The senior_manager_code is the code of the senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the

Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

working company.

- Manager: The manager_code is the code of the manager, the senior_manager_code is the code of its senior manager, the lead_manager_code is the code of its lead manager, and the company_code is the code of the working company.

Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

- Employee: The employee_code is the code of the employee, the manager_code is the code of its manager, the senior_manager_code is the code of its senior manager, the

`lead_manager_code` is the code of its lead manager, and the `company_code` is the code of the

Column	Type
<code>employee_code</code>	String
<code>manager_code</code>	String
<code>senior_manager_code</code>	String
<code>lead_manager_code</code>	String
<code>company_code</code>	String

working company.

Sample Input

company_code	founder
C1	Monika
C2	Samantha

Company Table:

lead_manager_code	company_code
LM1	C1
LM2	C2

Lead_Manager Table:

Senior_Manager Table:

senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

Manager Table:

manager_code	senior_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

Employee Table:

employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

Sample Output

C1 Monika 1 2 1 2

C2 Samantha 1 1 2 2

Hint -

In company C1, the only lead manager is LM1. There are two senior managers, SM1 and SM2, under LM1. There is one manager, M1, under senior manager SM1. There are two employees, E1 and E2, under manager M1.

In company C2, the only lead manager is LM2. There is one senior manager, SM3, under LM2. There are two managers, M2 and M3, under senior manager SM3. There is one employee, E3, under manager M2, and another employee, E4, under manager M3.

Solution:

```
select concat(c.company_code, ' ', c.founder, ' ',
count(distinct l.lead_manager_code), ' ',
count(distinct s.senior_manager_code), ' ',
count(distinct m.manager_code), ' ',
count(distinct e.employee_code)) as Output
from Company c
left outer join
Lead_Manager l
on c.company_code = l.company_code
left join
Senior_Manager s
on l.lead_manager_code = s.lead_manager_code
left join
Manager m
on s.senior_manager_code = m.senior_manager_code
left join
Employee e
on m.manager_code = e.manager_code
group by c.company_code, c.founder
order by c.company_code;
```

The screenshot shows a code editor in VS Code with a MySQL query tab open. The query is a script named 'create-db-template.sql' that creates tables for Company, Lead_Manager, Senior_Manager, Manager, and Employee, and inserts data into them. A specific query at the end is highlighted with a green checkmark.

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@1
20  manager_code varchar(10),
21  senior_manager_code varchar(10),
22  lead_manager_code varchar(10),
23  company_code varchar(10));
  ▷ Execute
24  insert into Company values
25  ('C1', 'Monika'),
26  ('C2', 'Samantha');
  ▷ Execute
27  insert into Lead_Manager values
28  ('LM1', 'C1'),
29  ('LM2', 'C2');
  ▷ Execute
30  insert into Senior_Manager values
31  ('SM1', 'LM1', 'C1'),
32  ('SM2', 'LM1', 'C1'),
33  ('SM3', 'LM2', 'C2');
  ▷ Execute
34  insert into Manager values
35  ('M1', 'SM1', 'LM1', 'C1'),
36  ('M2', 'SM3', 'LM2', 'C2'),
37  ('M3', 'SM3', 'LM2', 'C2');
  ▷ Execute
38  insert into Employee values
39  ('E1', 'M1', 'SM1', 'LM1', 'C1'),
40  ('E2', 'M1', 'SM1', 'LM1', 'C1'),
41  ('E3', 'M2', 'SM3', 'LM2', 'C2'),
42  ('E4', 'M3', 'SM3', 'LM2', 'C2');
  ▷ Execute
✓ 43  select concat(c.company_code, ' ', c.founder, ' ',
44  count(distinct l.lead_manager_code), ' ',
45  count(distinct s.senior_manager_code), ' ',
46  count(distinct m.manager_code), ' ',
47  count(distinct e.employee_code)) as Output
48  from Company c
49  left outer join
50  Lead_Manager l
51  on c.company_code = l.company_code
52  left join
53  Senior_Manager s
54  on l.lead_manager_code = s.lead_manager_code
55  left join
56  Manager m
57  on s.senior_manager_code = m.senior_manager_code
58  left join
59  Employee e
60  on m.manager_code = e.manager_code
61  group by c.company_code, c.founder
62  order by c.company_code;
63
```

The results viewer on the right shows the output of the final query:

	Output
1	C1 Monika 1 2 1 2
2	C2 Samantha 1 1 2 2

Q143 .

You are given a table, Functions, containing two columns: X and Y.

Column	Type
X	Integer
Y	Integer

Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.

Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that $X_1 \leq Y_1$.

Level - Medium

Source - Hackerrank

Hint - Use group by and having clause .

Sample Input

X	Y
20	20
20	20
20	21
23	22
22	23
21	20

Sample Output

20 20

20 21

22 23

Solution:

```
select distinct a.X, a.Y from
(select *, row_number() over(order by X) as r1 from Functions) a
inner join
(select *,row_number() over(order by X) as r2 from Functions) b
on a.X = b.Y and b.X = a.Y
where a.X <= a.Y and a.r1 <> b.r2
order by a.X
```

create-db-template.sql X [Preview] README.md

```
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
    ⚡ Active Connection | ▶ Execute
1  create database test;
    ▶ Execute
2  use test;
    ▶ Execute
3  create table Functions
4  (
5      X int,
6      Y int
7  );
    ▶ Execute
8  insert into Functions values
9  (20, 20),
10 (20, 20),
11 (20, 21),
12 (23, 22),
13 (22, 23),
14 (21, 20);
    ▶ Execute
15 select distinct a.X, a.Y from
16 (select *, row_number() over(order by X) as r1 from Functions) a
17 inner join
18 (select *,row_number() over(order by X) as r2 from Functions) b
19 on a.X = b.Y and b.X = a.Y
20 where a.X <= a.Y and a.r1 <> b.r2
21 order by a.X
22
23
```

Employee X

```
select distinct a.X, a.Y from
(select *, row_number() over(order by X) as r1 from Functions) a
 Free 1
```

Cost: 3ms

	X int	Y int
1	20	20
2	20	21
3	22	23

Q144 .

You are given three tables: Students, Friends and Packages. Students contains two columns: ID and Name. Friends contains two columns: ID and Friend_ID (ID of the ONLY best friend). Packages contain two columns: ID and Salary (offered salary in \$ thousands per month).

<i>Column</i>	<i>Type</i>
<i>ID</i>	<i>Integer</i>
<i>Name</i>	<i>String</i>

Students

<i>Column</i>	<i>Type</i>
<i>ID</i>	<i>Integer</i>
<i>Friend_ID</i>	<i>Integer</i>

Friends

<i>Column</i>	<i>Type</i>
<i>ID</i>	<i>Integer</i>
<i>Salary</i>	<i>Float</i>

Packages

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

Sample Input

<i>ID</i>	<i>Friend_ID</i>
1	2
2	3
3	4
4	1

Friends

<i>ID</i>	<i>Name</i>
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students

<i>ID</i>	<i>Salary</i>
1	15.20
2	10.06
3	11.55
4	12.12

Packages

Sample Output

Samantha
Julia
Scarlet

Explanation

See the following table:

<i>ID</i>	1	2	3	4
<i>Name</i>	Ashley	Samantha	Julia	Scarlet
<i>Salary</i>	15.20	10.06	11.55	12.12
<i>Friend ID</i>	2	3	4	1
<i>Friend Salary</i>	10.06	11.55	12.12	15.20

Now,

- Samantha's best friend got offered a higher salary than her at 11.55
- Julia's best friend got offered a higher salary than her at 12.12
- Scarlet's best friend got offered a higher salary than her at 15.2
- Ashley's best friend did NOT get offered a higher salary than her

The name output, when ordered by the salary offered to their friends, will be:

- Samantha
- Julia
- Scarlet

Solution:

```
select s.name
from
Students s
join
Friends f
on s.id = f.id
join
Packages sp
on sp.id = s.id
join
Packages fp
on fp.id = f.friend_id
where fp.salary > sp.salary
order by fp.salary;
```

```

create-db-template.sql ×
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-temp
  ↘ Active Connection | ⌂ Execute
1   create database test;
  ⌂ Execute
2   use test;
  ⌂ Execute
3   create table Students
4     (id int,
5      name varchar(15)
6    );
  ⌂ Execute
7   create table Friends
8     (id int,
9      friend_id int);
  ⌂ Execute
10  create table Packages
11    (id int,
12      salary float);
  ⌂ Execute
13  insert into Students values
14    (1, 'Ashley'),
15    (2, 'Samantha'),
16    (3, 'Julia'),
17    (4, 'Scarlet');
  ⌂ Execute
18  insert into Friends values
19    (1, 2),
20    (2, 3),
21    (3, 4),
22    (4, 1);
  ⌂ Execute
23  insert into Packages values
24    (1, 15.28),
25    (2, 10.06),
26    (3, 11.55),
27    (4, 12.12);
  ⌂ Execute
28  select s.name
29  from
30  Students s
31  join
32  Friends f
33  on s.id = f.id
34  join
35  Packages sp
36  on sp.id = s.id
37  join
38  Packages fp
39  on fp.id = f.friend_id
40  where fp.salary > sp.salary
41  order by fp.salary;
42
Data ×
select s.name
Free
Input to filter result
1
name
varchar
1 Samantha
2 Julia
3 Scarlet
Cost: 2ms < 1 > Total 3

```

Q145.

Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective hacker_id and name of hackers who achieved full scores for more than one challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in the same number of challenges, then sort them by ascending hacker_id.

Level - Medium

Hint - Use group by and having clause and order by .Input

Format

The following tables contain contest data:

- Hackers: The hacker_id is the id of the hacker, and name is the name of the hacker.

Column	Type
hacker_id	Integer
name	String

- Difficulty: The difficult_level is the level of difficulty of the challenge, and score is the

Column	Type
difficulty_level	Integer
score	Integer

score of the challenge for the difficulty level.

- Challenges: The challenge_id is the id of the challenge, the hacker_id is the id of the hacker who created the challenge, and difficulty_level is the level of difficulty of the challenge.

Column	Type
challenge_id	Integer
hacker_id	Integer
difficulty_level	Integer

- Submissions: The submission_id is the id of the submission, hacker_id is the id of the hacker who made the submission, challenge_id is the id of the challenge that the submission belongs

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

to, and score is the score of the submission.

Sample Input

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

Hackers Table:

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

Difficulty Table:

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

Challenges Table:

:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35
3924	8439	36566	80
97397	90411	66730	100
84162	83082	4810	40
97431	90411	71055	30

Submissions Table

Sample Output

90411 Joe

Explanation

Hacker 86870 got a score of 30 for challenge 71055 with a difficulty level of 2, so 86870 earned a full score for this challenge.

Hacker 90411 got a score of 30 for challenge 71055 with a difficulty level of 2, so 90411 earned a full score for this challenge.

Hacker 90411 got a score of 100 for challenge 66730 with a difficulty level of 6, so 90411 earned a full score for this challenge.

Only hacker 90411 managed to earn a full score for more than one challenge, so we print their hacker_id and name as 2 space-separated values.

Solution:

```
select concat(t1.hacker_id, ' ', t1.name) as Result from
(
    select t.hacker_id, t.name,
dense_rank() over(order by full_score_challenge_count desc) as r
from
(
    select h.hacker_id, h.name, count(h.hacker_id) as
full_score_challenge_count
        from
        Submissions s
        join
        Hackers h
        on s.hacker_id = h.hacker_id
        join
        Challenges c
        on s.challenge_id = c.challenge_id
        join
        Difficulty d
        on d.difficulty_level = c.difficulty_level
        where s.score = d.score
        group by h.hacker_id, h.name
        having full_score_challenge_count > 1
    ) t
) t1
where t1.r = 1
order by t1.hacker_id;
```

```

create-db-template.sql ✘
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
34      (5,80),
35      (6,100),
36      (7,120);
    ▷ Execute
37  insert into Challenges values
38  (4810, 77726, 4),
39  (21089, 27205, 1),
40  (36566, 5580, 7),
41  (66730, 52243, 6),
42  (71055, 52243, 2);
    ▷ Execute
43  insert into Submissions values
44  (68628,77726,36566,30),
45  (65300,77726,21089,10),
46  (40326,52243,36566,77),
47  (8941,27205,4810,4),
48  (83554,77726,66730,30),
49  (43353,52243,66730,8),
50  (55385,52348,71055,20),
51  (39784,27205,71055,23),
52  (94613,86870,71055,30),
53  (45788,52348,36566,8),
54  (93058,86870,36566,30),
55  (7344,8439,66730,92),
56  (2721,8439,4810,36),
57  (523,5580,71055,4),
58  (49105,52348,66730,8),
59  (55877,57645,66730,80),
60  (38355,27205,66730,35),
61  (3924,8439,36566,80),
62  (97397,98411,66730,100),
63  (84162,83082,4810,48),
64  (97431,98411,71055,30);
65
    ▷ Execute
66  select concat(t1.hacker_id, ' ', t1.name) as Result from
67  (
68    select t.hacker_id, t.name,
69    dense_rank() over(order by full_score_challenge_count desc) as r
70  from
71    (
72      select h.hacker_id, h.name, count(h.hacker_id) as full_score_challenge_count
73      from
74      Submissions s
75      join
76      Hackers h
77      on s.hacker_id = h.hacker_id
78      join
79      Challenges c
80      on s.challenge_id = c.challenge_id
81      join
82      Difficulty d
83      on d.difficulty_level = c.difficulty_level
84      where s.score = d.score
85      group by h.hacker_id, h.name
86      having full_score_challenge_count > 1
87    ) t
88  ) t1
89  where t1.r = 1
90  order by t1.hacker_id;
91
Data X
select concat(t1.hacker_id, ' ', t1.name) as Result from
/
+-----+-----+
| 1 | 90411 Joe |
+-----+-----+

```

Q146.

You are given a table, Projects, containing three columns: Task_ID, Start_Date and End_Date. It is guaranteed that the difference between the End_Date and the Start_Date is equal to 1 day for each row in the table.

Level - Medium

Hint - Use Advance join

<i>Column</i>	<i>Type</i>
<i>Task_ID</i>	<i>Integer</i>
<i>Start_Date</i>	<i>Date</i>
<i>End_Date</i>	<i>Date</i>

If the *End_Date* of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

Sample Input

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

Sample Output

2015-10-28 2015-10-29
 2015-10-30 2015-10-31
 2015-10-13 2015-10-15
 2015-10-01 2015-10-04

Explanation

The example describes following four projects:

- Project 1: Tasks 1, 2 and 3 are completed on consecutive days, so these are part of the project. Thus the start date of project is 2015-10-01 and end date is 2015-10-04, so it took 3 days to complete the project.
- Project 2: Tasks 4 and 5 are completed on consecutive days, so these are part of the project. Thus, the start date of project is 2015-10-13 and end date is 2015-10-15, so it took 2 days to complete the project.
- Project 3: Only task 6 is part of the project. Thus, the start date of project is 2015-10-28 and end date is 2015-10-29, so it took 1 day to complete the project.
- Project 4: Only task 7 is part of the project. Thus, the start date of project is 2015-10-30 and end date is 2015-10-31, so it took 1 day to complete the project.

Solution:

```
select s.start_date, min(e.end_date) as end_date, (min(e.end_date) -
s.start_date) as number_of_days
from
(select start_date from Projects where start_date - 1 not in (select start_date
from Projects)) s,
(select end_date from Projects where end_date + 1 not in (select end_date from
Projects)) e
where s.start_date <= e.end_date
group by s.start_date;
```

create-db-template.sql X [Preview] README.md

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

```
2      ⌂ Execute
3  create database test;
4      ⌂ Execute
5  use test;
6      ⌂ Execute
7  create table Projects
8  (task_id int,
9   start_date date,
10  end_date date);
11      ⌂ Execute
12  insert into Projects values
13  (1, '2015-10-01', '2015-10-02'),
14  (2, '2015-10-02', '2015-10-03'),
15  (3, '2015-10-03', '2015-10-04'),
16  (4, '2015-10-13', '2015-10-14'),
17  (5, '2015-10-14', '2015-10-15'),
18  (6, '2015-10-28', '2015-10-29'),
19  (7, '2015-10-30', '2015-10-31');
20      ⌂ Execute
21  select s.start_date, min(e.end_date) as end_date, (min(e.end_date) - s.start_date) as number_of_days
22  from
23  (select start_date from Projects where start_date - 1 not in (select start_date from Projects)) s,
24  (select end_date from Projects where end_date + 1 not in (select end_date from Projects)) e
25  where s.start_date <= e.end_date
26  group by s.start_date;
27
28
```

Data X

select s.start_date, min(e.end_date) as end_date, (min(e.end_date) - s.start_date) as number_of_days

Free 1 Input to filter result

Cost: 3ms < 1 Total 4

	start_date	end_date	number_of_days
1	2015-10-01	2015-10-04	3
2	2015-10-13	2015-10-15	2
3	2015-10-28	2015-10-29	1
4	2015-10-30	2015-10-31	1

Q147.

In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days.

List the user IDs who have gone on at least 1 shopping spree in ascending order.

transactions Table:

Column Name	Type
user_id	integer
amount	float
transaction_date	timestamp

transactions Example Input:

user_id	amount	transaction_date
1	9.99	08/01/2022 10:00:00
1	55	08/17/2022 10:00:00
2	149.5	08/05/2022 10:00:00
2	4.89	08/06/2022 10:00:00
2	34	08/07/2022 10:00:00

Example Output:

user_id
2

Solution:

```
select distinct t.user_id
from
(
    select user_id, transaction_date as first,
    lead(transaction_date,1) over(partition by user_id order by
transaction_date) as second,
    lead(transaction_date,2) over(partition by user_id order by
transaction_date) as third
    from transactions
) t
where timestampdiff(day, first, second) = 1 and timestampdiff(day, second,
third) = 1;
```

The screenshot shows the MySQL Workbench interface. At the top, there's a breadcrumb navigation: config > data > User > globalStorage > cwejan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql > ...

The main area contains the SQL code for creating a database and table, and executing the solution query. The code is numbered from 1 to 25.

```
1 create database test;
2 use test;
3 create table transactions
4 (
5     user_id int,
6     amount float,
7     transaction_date timestamp
8 );
9 insert into transactions values
10 (1, 9.99, '2022/08/01 10:00:00'),
11 (1, 55, '2022/08/17 10:00:00'),
12 (2, 149.5, '2022/08/05 10:00:00'),
13 (2, 4.89, '2022/08/06 10:00:00'),
14 (2, 34, '2022/08/07 10:00:00');
15 select distinct t.user_id
16 from
17 (
18     select user_id, transaction_date as first,
19     lead(transaction_date,1) over(partition by user_id order by transaction_date) as second,
20     lead(transaction_date,2) over(partition by user_id order by transaction_date) as third
21     from transactions
22 ) t
23 where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 1;
```

Below the code, there's a table named "transactions" with one row of data:

user_id	amount	transaction_date
1	2	2022/08/01 10:00:00

Q148 .

You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data.

Assumption:

- A payer can send money to the same recipient multiple times.

payments Table:

Column Name	Type
payer_id	integer
recipient_id	integer
amount	integer

payments Example Input:

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

Example Output:

unique_relationships
2

```
Solution:  
select count(*) as unique_relationships  
from  
(select count(*) as relation_count  
from  
(  
select greatest(payer_id, recipient_id) as person1,  
least(payer_id, recipient_id) as person2  
from  
(select distinct * from payments) t  
) t1  
group by person1, person2  
) t2  
where relation_count = 2;
```

The screenshot shows a MySQL Workbench interface. The top part is a query editor with the following SQL code:

```

1 use test;
2 create table payments
3     (payer_id int,
4      recipient_id int,
5      amount int);
6 insert into payments values
7     (101,    201,    30),
8     (201,    101,    10),
9     (101,    301,    20),
10    (301,    101,    80),
11    (201,    301,    70);
12
13 select count(*) as unique_relationships
14 from
15     (select count(*) as relation_count
16      from
17      (
18          select greatest(payer_id, recipient_id) as person1,
19              least(payer_id, recipient_id) as person2
20          from
21          (select distinct * from payments) t
22      ) t1
23      group by person1, person2
24  ) t2
25  where relation_count = 2;
26

```

The bottom part is a results viewer titled "payments" showing the output of the query:

	unique_relationships
1	2

Q149. Assume you are given the table below on user transactions. Write a query to obtain the list of customers whose first transaction was valued at \$50 or more. Output the number of users.

Clarification:

- Use the transaction_date field to determine which transaction should be labeled as the first for each user.
- Use a specific function (we can't give too much away!) to account for scenarios where a user had multiple transactions on the same day, and one of those was the first.

user_transactions Table:

Column Name	Type
transaction_id	integer

user_id	Integer
Spend	Decimal
transaction_date	Timestamp

user_transactions Example Input:

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

Example Output:

Users
1

Solution:

```
select count(*) as users from
(
    select transaction_id, user_id, spend,
    row_number() over(partition by user_id order by transaction_date) as r
    from user_transactions
) t
where t.r =1 and t.spend >= 50;
```

```
create-db-template.sql X
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create
    > Execute
2  use test;
    > Execute
3  create table user_transactions
4      (transaction_id int,
5       user_id Int,
6       spend float,
7       transaction_date TIMESTAMP
8   );
    > Execute
9  insert into user_transactions values
10     (759274,      111,      49.50, '2022/02/03 00:00:00'),
11     (850371,      111,      51.00, '2022/03/15 00:00:00'),
12     (615348,      145,      36.30, '2022/03/22 00:00:00'),
13     (137424,      156,     151.00, '2022/04/04 00:00:00'),
14     (248475,      156,      87.00, '2022/04/16 00:00:00');
    > Execute
✓ 15  select count(*) as users from
16  (
17      select transaction_id, user_id, spend,
18          row_number() over(partition by user_id order by transaction_date) as r
19      from user_transactions
20  ) t
21  where t.r =1 and t.spend >= 50;
22
```

user_transactions X

select count(*) as users from

Free 1

	users	bigint
	1	1

Cost: 5ms < 1 > To

Q150.

Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day.

Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

Note that the 1st, 3rd, 5th measurements within a day are considered odd-numbered measurements and the 2nd, 4th, 6th measurements are even-numbered measurements.

measurements Table:

Column Name	Type
measurement_id	Integer
measurement_value	Decimal
measurement_time	Datetime

measurements Example Input:

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

Example Output:

measurement_day	odd_sum	even_sum
07/10/2022 00:00:00	2355.75	1662.74

07/11/2022 00:00:00	1124.50	1234.14
---------------------	---------	---------

Solution:

```

select measurement_day,
round(sum(case when r % 2 != 0 then measurement_value else 0 end),2) as odd_sum,
round(sum(case when r % 2 = 0 then measurement_value else 0 end),2) as even_sum
from
(
    select date_format(measurement_time, '%m/%d/%Y 00:00:00') as
measurement_day,
measurement_value, row_number() over(partition by date(measurement_time) order
by measurement_time) as r
from measurements
)t
group by measurement_day;

```

The screenshot shows the MySQL Workbench interface with two main panes. The top pane displays a SQL script named 'create-db-template.sql' containing the solution code. The bottom pane shows the results of a query named 'measurements'.

create-db-template.sql

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@127.0.0.1@3306 > create-db-template.sql > ...
3   create table measurements
4  (
5    measurement_id  Int,
6    measurement_value  float,
7    measurement_time  Datetime
8  );
9
10  insert into measurements values
11  (131233,      1109.51,      '2022/07/10 09:00:00'),
12  (135211,      1662.74,      '2022/07/10 11:00:00'),
13  (523542,      1246.24,      '2022/07/10 13:15:00'),
14  (143562,      1124.50,      '2022/07/11 15:00:00'),
15  (346462,      1234.14,      '2022/07/11 16:45:00');
16
17  select measurement_day,
18  round(sum(case when r % 2 != 0 then measurement_value else 0 end),2) as odd_sum,
19  round(sum(case when r % 2 = 0 then measurement_value else 0 end),2) as even_sum
20  from
21  (
22    select date_format(measurement_time, '%m/%d/%Y 00:00:00') as measurement_day,
23    measurement_value, row_number() over(partition by date(measurement_time) order by measurement_time)
24  from measurements
25 )t
26  group by measurement_day;
27

```

measurements

```

select measurement_day,
round(sum(case when r % 2 != 0 then measurement_value else 0 end),2) as odd_sum,
round(sum(case when r % 2 = 0 then measurement_value else 0 end),2) as even_sum

```

Input to filter result: **Free** 1

	measurement_day	odd_sum	even_sum
1	07/10/2022 00:00:00	2355.75	1662.74
2	07/11/2022 00:00:00	1124.5	1234.14

Q151.

In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days.

List the user IDs who have gone on at least 1 shopping spree in ascending order.

Level - Medium

Hint - Use self join

transactions Table:

Column Name	Type
user_id	integer
amount	float
transaction_date	timestamp

transactions Example Input:

user_id	Amount	transaction_date
1	9.99	08/01/2022 10:00:00
1	55	08/17/2022 10:00:00
2	149.5	08/05/2022 10:00:00
2	4.89	08/06/2022 10:00:00
2	34	08/07/2022 10:00:00

Example Output:

user_id
2

Solution:

```
select distinct t.user_id
from
(
    select user_id, transaction_date as first,
    lead(transaction_date,1) over(partition by user_id order by
transaction_date) as second,
    lead(transaction_date,2) over(partition by user_id order by
transaction_date) as third
    from transactions
) t
where timestampdiff(day, first, second) = 1 and timestampdiff(day, second,
third) = 1;
```

The screenshot shows a MySQL Workbench interface. The top window is a query editor titled "create-db-template.sql" with the following SQL code:

```

1 create database test;
2 use test;
3 create table transactions
4 (
5     user_id int,
6     amount float,
7     transaction_date timestamp
8 );
9 insert into transactions values
10 (1, 9.99, '2022/08/01 10:00:00'),
11 (1, 55, '2022/08/17 10:00:00'),
12 (2, 149.5, '2022/08/05 10:00:00'),
13 (2, 4.89, '2022/08/06 10:00:00'),
14 (2, 34, '2022/08/07 10:00:00');
15 select distinct t.user_id
16 from
17 (
18     select user_id, transaction_date as first,
19         lead(transaction_date,1) over(partition by user_id order by transaction_date) as second,
20         lead(transaction_date,2) over(partition by user_id order by transaction_date) as third
21     from transactions
22 ) t
23 where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 1;
24
25

```

The bottom window is a results viewer titled "transactions" showing the following data:

user_id
1
2

Q152.

The Airbnb Booking Recommendations team is trying to understand the "substitutability" of two rentals and whether one rental is a good substitute for another. They want you to write a query to find the unique combination of two Airbnb rentals with the same exact amenities offered. Output the count of the unique combination of Airbnb rentals.

Level - Medium

Hint - Use unique statement

Assumptions:

- If property 1 has a kitchen and pool, and property 2 has a kitchen and pool too, it is a good substitute and represents a unique matching rental.
- If property 3 has a kitchen, pool and fireplace, and property 4 only has a pool and fireplace, then it is not a good substitute.

rental_amenities Table:

Column Name	Type
rental_id	integer
Amenity	String

rental_amenities Example Input:

rental_id	Amenity
123	Pool
123	Kitchen
234	hot tub
234	fireplace
345	Kitchen

345	Pool
456	Pool

Example Output:

matching_airbnb
1

Solution:

```
select count(t1.amenity_count) as matching_airbnb
from
(
    select t.amenities, count(*) as amenity_count
    from
    (
        select rental_id, group_concat(amenity order by amenity) amenities
        from rental_amenities
        group by rental_id
    )t
    group by t.amenities
)t1
where t1.amenity_count>1;
```

```
create-db-template.sql × [Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
1  create database test;
  ▷ Execute
2  use test;
  ▷ Execute
3  create table rental_amenities
4  (rental_id int,
5  amenity varchar(20));
  ▷ Execute
6  insert into rental_amenities values
7  (123, 'Pool'),
8  (123, 'Kitchen'),
9  (234, 'hot tub'),
10 (234, 'fireplace'),
11 (345, 'kitchen'),
12 (345, 'pool'),
13 (456, 'pool');
  ▷ Execute
14 select count(t1.amenity_count) as matching_airbnb
15 from
16 (
17     select t.amenities, count(*) as amenity_count
18     from
19     (
20         select rental_id, group_concat(amenity order by amenity) amenities
21         from rental_amenities
22         group by rental_id
23     )t
24     group by t.amenities
25 )t1
26 where t1.amenity_count>1;
27
rental_amenities ×
select count(t1.amenity_count) as matching_airbnb
from
  ▷ Input to filter result
  ▷ Free 1
  ▷ matching_airbnb bigint
  ▷ Cost: 5ms < 1 >
  ▷ 1 | 1
```

Q153.

Google marketing managers are analysing the performance of various advertising accounts over the last month. They need your help to gather the relevant data.

Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser_id.

Level - Medium

Hint: ROAS = Ad Revenue / Ad Spend

ad_campaigns Table:

Column Name	Type
campaign_id	Integer
spend	Integer
revenue	float
advertiser_id	Integer

ad_campaigns Example Input:

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

Example Output:

advertiser_id	ROAS
1	0.9
2	4
3	1.5
4	4

Solution:

```
select advertiser_id,  
sum(revenue)/sum(spend) as ROAS  
from ad_campaigns  
group by advertiser_id  
order by advertiser_id;
```

create-db-template.sql [Preview] README.md

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 166835512992
    ▶ Execute
2   use test;
    ▶ Execute
3   create table ad_campaigns
4     (campaign_id Int,
5      spend Int,
6      revenue float,
7      advertiser_id Int
8    );
    ▶ Execute
9   insert into ad_campaigns values
10  (1, 5000, 7500, 3),
11  (2, 1000, 900, 1),
12  (3, 3000, 12000, 2),
13  (4, 500, 2000, 4),
14  (5, 100, 400, 4);
    ▶ Execute
15  select advertiser_id,
16    sum(revenue)/sum(spend) as ROAS
17  from ad_campaigns
18  group by advertiser_id
19  order by advertiser_id;
20

```

ad_campaigns

	advertiser_id	ROAS
1	1	0.9
2	2	4
3	3	1.5
4	4	4

Q154.

Your team at Accenture is helping a Fortune 500 client revamp their compensation and benefits program. The first step in this analysis is to manually review employees who are potentially overpaid or underpaid.

An employee is considered to be potentially overpaid if they earn more than 2 times the average salary for people with the same title. Similarly, an employee might be underpaid if they earn less than half of the average for their title. We'll refer to employees who are both underpaid and overpaid as

compensation outliers for the purposes of this problem.

Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below).

Hint: ROAS = Ad Revenue / Ad Spend

employee_pay Table:

Column Name	Type
employee_id	integer
Salary	integer
Title	varchar

employee_pay Example Input:

employee_id	salary	Title
101	80000	Data Analyst
102	90000	Data Analyst
103	100000	Data Analyst
104	30000	Data Analyst

105	120000	Data Scientist
106	100000	Data Scientist
107	80000	Data Scientist
108	310000	Data Scientist

employee_id	Salary	status
104	30000	Underpaid
108	310000	Overpaid

Example Output:

Solution:

```

select t.employee_id, t.salary, case
when t.salary > t.base_for_overpaid then 'Overpaid'
when t.salary < t.base_for_underpaid then 'Underpaid'
end as status
from
(select employee_id, salary, 2*avg(salary) over(partition by title) as
base_for_overpaid,
0.5*avg(salary) over(partition by title) as base_for_underpaid
from employee_pay
)t
having status is not null
order by t.employee_id;

```

create-db-template.sql X [Preview] README.md

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
    ▾ EXECUTE
  2  use test;
    ▷ Execute
  3  create table employee_pay
  4  (employee_id      int,
  5   salary      int,
  6   title      varchar(30)
  7  );
    ▷ Execute
  8  insert into employee_pay values
  9  (101,     80000,  'Data Analyst'),
 10  (102,     90000,  'Data Analyst'),
 11  (103,    100000,  'Data Analyst'),
 12  (104,     30000,  'Data Analyst'),
 13  (105,    120000,  'Data Scientist'),
 14  (106,    100000,  'Data Scientist'),
 15  (107,     80000,  'Data Scientist'),
 16  (108,    310000,  'Data Scientist');
    ▷ Execute
  17 select t.employee_id, t.salary, case
  18 when t.salary > t.base_for_overpaid then 'Overpaid'
  19 when t.salary < t.base_for_underpaid then 'Underpaid'
  20 end as status
  21 from
  22 (select employee_id, salary,
  23 2*avg(salary) over(partition by title) as base_for_overpaid,
  24 0.5*avg(salary) over(partition by title) as base_for_underpaid
  25 from employee_pay
  26 )t
  27 having status is not null
  28 order by t.employee_id;
  29

```

employee_pay X

```

select t.employee_id, t.salary, case
when t.salary > t.base_for_overpaid then 'Overpaid'
when t.salary < t.base_for_underpaid then 'Underpaid'

```

Free 1 Input to filter result

	employee_id	salary	status
1	104	30000	Underpaid
2	108	310000	Overpaid

Cost: 6ms <

Q155.

You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data.

Assumption:

- A payer can send money to the same recipient multiple times.

Hint- Use the INTERSECT set operator.

payments Table:

Column Name	Type
payer_id	integer
recipient_id	integer
amount	integer

payments Example Input:

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

Example Output:

unique_relationships
2

Solution:

```
select count(*) as unique_relationships
from
(select count(*) as relation_count
from
(
select greatest(payer_id, recipient_id) as person1,
least(payer_id, recipient_id) as person2
from
(select distinct * from payments) t
) t1
group by person1, person2
) t2
where relation_count = 2;
```

create-db-template.sql X [Preview] README.md

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > 
    ▷ Execute
2   use test;
    ▷ Execute
3   create table payments
4     (payer_id int,
5      recipient_id int,
6      amount int);
    ▷ Execute
7   insert into payments values
8     (101,    201,    30),
9     (201,    101,    10),
10    (101,    301,    20),
11    (301,    101,    80),
12    (201,    301,    70);
    ▷ Execute
13  select count(*) as unique_relationships
14  from
15  (select count(*) as relation_count
16  from
17  (
18  select greatest(payer_id, recipient_id) as person1,
19  least(payer_id, recipient_id) as person2
20  from
21  (select distinct * from payments) t
22  ) t1
23  group by person1, person2
24  ) t2
25  where relation_count = 2;
26

```

payments X

```

select count(*) as unique_relationships

```

Free 1

Input to filter result

	unique_relationships	bigint
<input checked="" type="checkbox"/>	1	2

Cost: 7ms < 1

Q156.

Assume you are given the table below containing information on user purchases. Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users.

PS. On 26 Oct 2022, we expanded the purchases data set, thus the official output may vary from before.

Hint- Count the distinct number of dates formatted into the DATE format in the COUNT(DISTINCT).

purchases Table:

Column Name	Type
-------------	------

user_id	Integer
product_id	Integer
quantity	Integer
purchase_date	Datetime

purchasesExample Input:

user_id	product_id	quantity	purchase_date
536	3223	6	01/11/2022 12:33:44

827	3585	35	02/20/2022 14:05:26
536	3223	5	03/02/2022 09:33:28
536	1435	10	03/02/2022 08:40:00
827	2452	45	04/09/2022 00:00:00

Example Output:

repeat_purchasers
1

Solution:

```
select count(distinct t.user_id) as repeat_purchasers
from
(
    select user_id, product_id, count(*) as c
    from purchases
    group by user_id, product_id
    having c > 1
) t;
```

The screenshot shows the MySQL Workbench interface. The top part is a script editor with a dark theme containing SQL code. The bottom part is a results grid for a query named 'purchases'.

```

create-db-template.sql X [Preview] README.md
config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 >
  ⚡ Active Connection | ▶ Execute
1   create database test;
  ▶ Execute
2   use test;
  ▶ Execute
3   create table purchases
4     (user_id      Int,
5      product_id  Int,
6      quantity    Int,
7      purchase_date Datetime
8    );
  ▶ Execute
9   insert into purchases values
10  (536,    3223,    6,    '2022/01/11 12:33:44'),
11  (827,    3585,    35,   '2022/02/20 14:05:26'),
12  (536,    3223,    5,    '2022/03/02 09:33:28'),
13  (536,    1435,    10,   '2022/03/02 08:40:00'),
14  (827,    2452,    45,   '2022/04/09 00:00:00');
  ▶ Execute
✓ 15  select count(distinct t.user_id) as repeat_purchasers
16  from
17  (
18    select user_id, product_id, count(*) as c
19    from purchases
20    group by user_id, product_id
21    having c > 1
22  ) t;
23

```

purchases

```

select count(distinct t.user_id) as repeat_purchasers
from

```

Input to filter result: Free 1

	repeat_purchasers	bigint
1	1	

Cost: 7ms

Q157.

Say you have access to all the transactions for a given merchant account. Write a query to print the cumulative balance of the merchant account at the end of each day, with the total balance reset back to zero at the end of the month. Output the transaction date and cumulative balance.

Hint-You should use CASE.

transactions Table:

Column Name	Type
transaction_id	Integer
Type	string ('deposit', 'withdrawal')
Amount	Decimal
transaction_date	Timestamp

transactions Example Input:

transaction_id	Type	amount	transaction_date
19153	Deposit	65.90	07/10/2022 10:00:00
53151	Deposit	178.55	07/08/2022 10:00:00

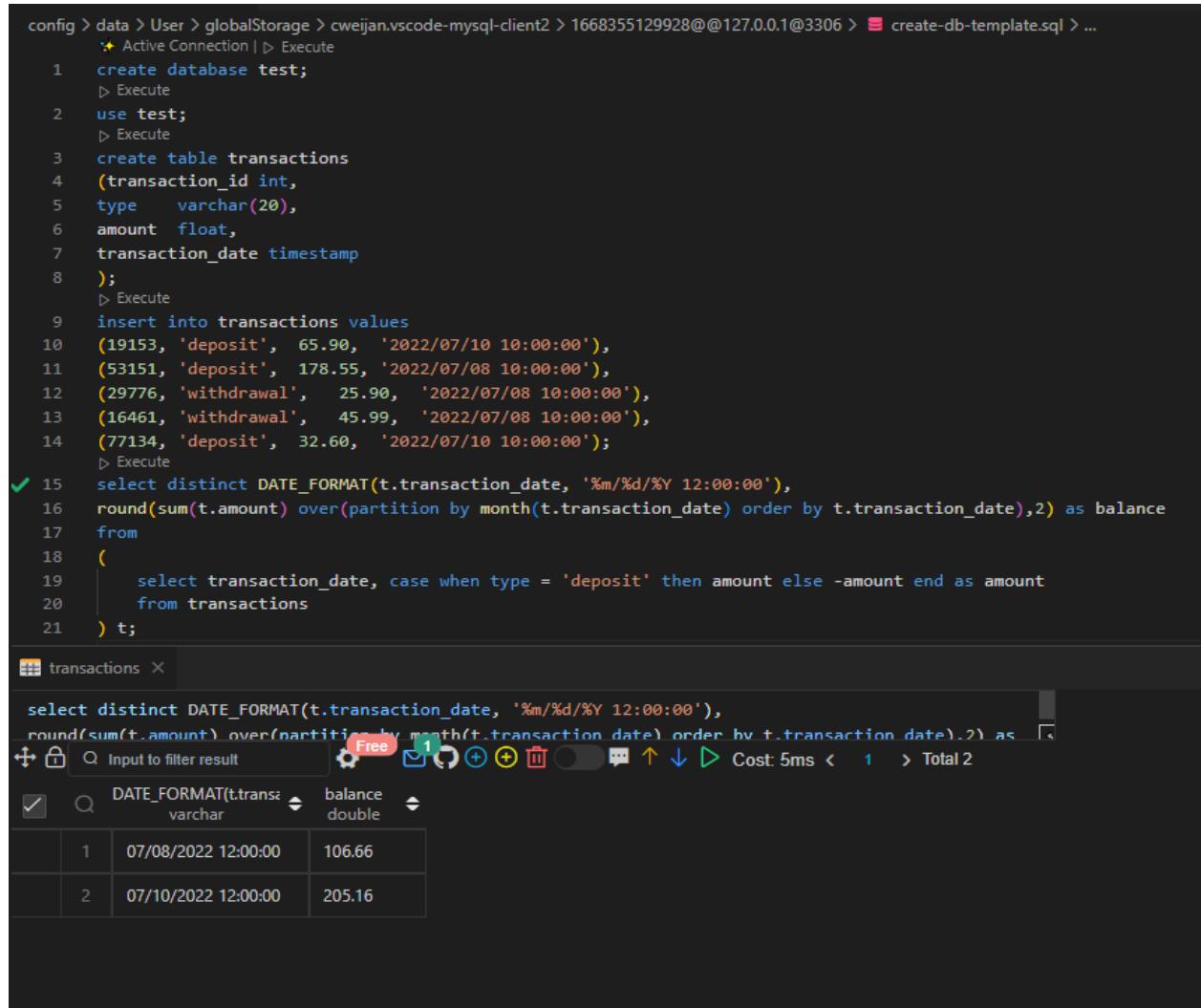
29776	Withdrawal	25.90	07/08/2022 10:00:00
16461	Withdrawal	45.99	07/08/2022 10:00:00
77134	Deposit	32.60	07/10/2022 10:00:00

Example Output:

transaction_date	balance
07/08/2022 12:00:00	106.66
07/10/2022 12:00:00	205.16

Solution:

```
select distinct DATE_FORMAT(transaction_date, '%m/%d/%Y 12:00:00'),
round(sum(amount) over(partition by month(transaction_date) order by
transaction_date),2) as balance
from
(
    select transaction_date, case when type = 'deposit' then amount else -amount
end as amount
    from transactions
) t;
```



The screenshot shows the MySQL Workbench interface with the following steps:

- Configuration:** A connection named "User" is selected.
- SQL Editor:**
 - Line 1: `create database test;`
 - Line 2: `use test;`
 - Line 3: `create table transactions`
 - Line 4: `(transaction_id int,`
 - Line 5: `type varchar(20),`
 - Line 6: `amount float,`
 - Line 7: `transaction_date timestamp`
 - Line 8: `);`
 - Line 9: `insert into transactions values`
 - Lines 10-14: Data insertion: `(19153, 'deposit', 65.00, '2022/07/10 10:00:00'),
(53151, 'deposit', 178.55, '2022/07/08 10:00:00'),
(29776, 'withdrawal', 25.90, '2022/07/08 10:00:00'),
(16461, 'withdrawal', 45.99, '2022/07/08 10:00:00'),
(77134, 'deposit', 32.60, '2022/07/10 10:00:00');`
 - Line 15: The provided SQL solution for calculating the balance.
 - Line 16: `select distinct DATE_FORMAT(t.transaction_date, '%m/%d/%Y 12:00:00'),`
 - Line 17: `round(sum(t.amount) over(partition by month(t.transaction_date) order by t.transaction_date),2) as balance`
 - Line 18: `from`
 - Line 19: `(`
 - Line 20: `select transaction_date, case when type = 'deposit' then amount else -amount end as amount`
 - Line 21: `from transactions`
 - Line 22: `) t;`
- Results Tab:** The results of the final query are displayed in a table:

	transaction_date	balance
1	07/08/2022 12:00:00	106.66
2	07/10/2022 12:00:00	205.16

Q158.

Assume you are given the table below containing information on Amazon customers and their spend on products belonging to various categories. Identify the top two highest-grossing products within each category in 2022. Output the category, product, and total spend.

Hint- Use where ,and, group by .

product_spend Table:

Column Name	Type
Category	String
Product	String
user_id	Integer
Spend	Decimal
transaction_date	Timestamp

product_spend Example Input:

Category	Product	user_id	spend	transaction_date

Appliance	Refrigerator	165	246.00	12/26/2021 12:00:00
Appliance	Refrigerator	123	299.99	03/02/2022 12:00:00
Appliance	washing machine	123	219.80	03/02/2022 12:00:00
electronics	Vacuum	178	152.00	04/05/2022 12:00:00
electronics	wireless headset	156	249.90	07/08/2022 12:00:00
electronics	Vacuum	145	189.00	07/15/2022 12:00:00

Example Output:

Category	Product	total_spend
Appliance	Refrigerator	545.99
Appliance	washing machine	219.80
electronics	Vacuum	341.00
electronics	wireless headset	249.90

Solution:

```

select t.category, t.product, t.total_spend
from
(
    select category, product, round(sum(spend),2) as total_spend,
dense_rank() over(partition by category order by sum(spend) desc) as r
from product_spend
group by category, product
) t
where r <= 2

```

create-db-template.sql [Preview] README.md

```

config > data > User > globalStorage > cweijan.vscode-mysql-client2 > 1668355129928@@127.0.0.1@3306 > create-db-template.sql
      ↳ Execute
3   create table product_spend
4     (category  varchar(20),
5      product  varchar(20),
6      user_id  Int,
7      Spend    float,
8      transaction_date  Timestamp
9    );
      ↳ Execute
10  insert into product_spend values
11    ('appliance',  'Refrigerator', 165,    246.00, '2021/12/26 12:00:00'),
12    ('appliance',  'Refrigerator', 123,    299.99, '2022/03/02 12:00:00'),
13    ('appliance',  'washing machine', 123,    219.80, '2022/03/02 12:00:00'),
14    ('electronics', 'Vacuum', 178,    152.00, '2022/04/05 12:00:00'),
15    ('electronics', 'wireless headset', 156,    249.90, '2022/07/08 12:00:00'),
16    ('electronics', 'Vacuum', 145,    189.00, '2022/07/15 12:00:00');
17
      ↳ Execute
18  select t.category, t.product, t.total_spend
19  from
20  (
21    select category, product, round(sum(spend),2) as total_spend,
22    dense_rank() over(partition by category order by sum(spend) desc) as r
23  from product_spend
24  group by category, product
25  ) t
26  where r <= 2

```

product_spend

	category	product	total_spend
1	appliance	Refrigerator	545.99
2	appliance	washing machine	219.8
3	electronics	Vacuum	341
4	electronics	wireless headset	249.9

Q159.

Facebook is analysing its user signup data for June 2022. Write a query to generate the churn rate by week in June 2022. Output the week number (1, 2, 3, 4, ...) and the corresponding churn rate rounded to 2 decimal places.

For example, week number 1 represents the dates from 30 May to 5 Jun, and week 2 is from 6 Jun to 12 Jun.

Hint- Use Extract.

Assumptions:

- If the last_login date is within 28 days of the signup_date, the user can be considered churned.

- If the last_login is more than 28 days after the signup date, the user didn't churn.

users Table:

Column Name	Type
user_id	integer

signup_date	Datetime
last_login	Datetime

users Example Input:

user_id	signup_date	last_login
1001	06/01/2022 12:00:00	07/05/2022 12:00:00
1002	06/03/2022 12:00:00	06/15/2022 12:00:00
1004	06/02/2022 12:00:00	06/15/2022 12:00:00
1006	06/15/2022 12:00:00	06/27/2022 12:00:00
1012	06/16/2022 12:00:00	07/22/2022 12:00:00

Example Output:

signup_week	churn_rate
1	66.67
3	50.00

User ids 1001, 1002, and 1004 signed up in the first week of June 2022. Out of the 3 users, 1002 and 1004's last login is within 28 days from the signup date, hence they are churned users.

To calculate the churn rate, we take churned users divided by total users signup in the week. Hence 2 users / 3 users = 66.67%.

```
Solution: according to week of year
select week(signup_date),
round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1
else 0 end)/count(*),2) as churn_rate
from users
group by week(signup_date);
```

The screenshot shows a database development environment with a code editor and a results viewer.

Code Editor:

```

config > data > User > globalStorage > cweijia
  ↘ Execute
  3  create table users
  4  (user_id    int,
  5  signup_date datetime,
  6  last_login  datetime
  7  );
  ↘ Execute
  8  insert into users values
  9  (1001, '2022/06/01 12:00:00', '2022/07/05 12:00:00'),
 10  (1002, '2022/06/03 12:00:00', '2022/06/15 12:00:00'),
 11  (1004, '2022/06/02 12:00:00', '2022/06/15 12:00:00'),
 12  (1006, '2022/06/15 12:00:00', '2022/06/27 12:00:00'),
 13  (1012, '2022/06/16 12:00:00', '2022/07/22 12:00:00');
  ↘ Execute
  ✓ 14  select week(signup_date),
 15  round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1 else 0 end)/count(*),2) as churn_rate
 16  from users
 17  group by week(signup_date);

```

A tooltip message "Please enter a value for the variable 00" is displayed above the code editor area. Another tooltip message "Note that strings are not automatically quoted. (Press 'Enter' to confirm or 'Escape' to cancel)" is also visible.

Results Viewer:

	week(signup_date)	churn_rate
1	22	66.67
2	24	50.00

```

Solution: according to week of month
(select signup_date, last_login, case
when week(signup_date) = 22 then 1
when week(signup_date) = 23 then 2
when week(signup_date) = 24 then 3
when week(signup_date) = 25 then 4
when week(signup_date) = 26 then 5
end as signup_week
from users)
select signup_week,
round(100*sum(case when timestampdiff(day,signup_date,last_login) <= 28 then 1
else 0 end)/count(*),2) as churn_rate
from cte
group by signup_week;

```

