SQL D1

10 February 2025 10:11

ranawatjinesh@gmail.com

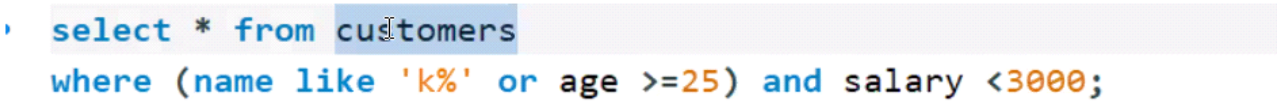
Structure ;- anything which have fixed structure
Query: question
Language;-
Table ;- rows & columns where we can store

Types of lang
DDL : data definition lang(allows you to work on schema level)
Create, modify, alter
Alter table table1 rename to table 2 (for various purposes)
Rename table 1 to table2
For clone table - create table 1 select * from table 2;(deep copy)

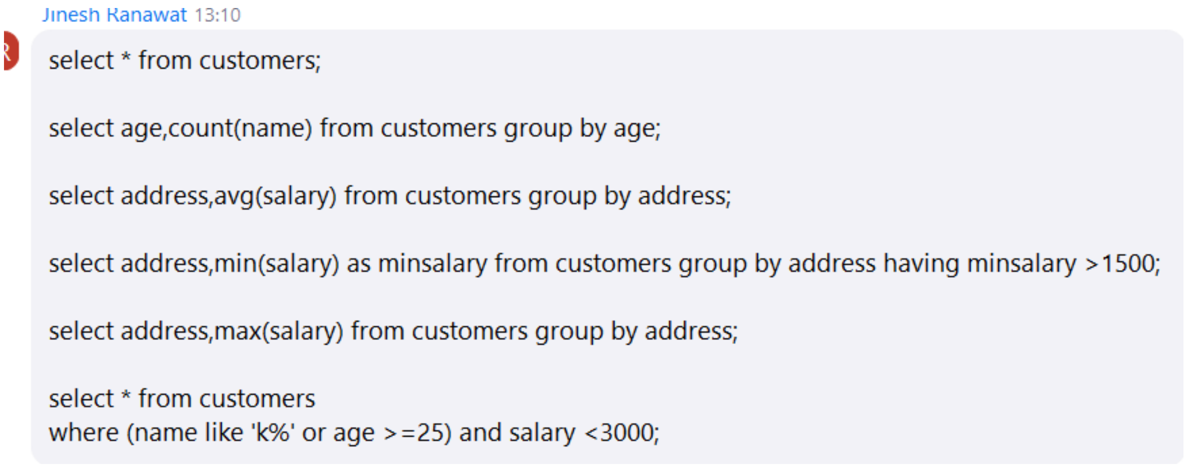
Create table1 like table2;(shallow)
Desc tablename;(description)
Create temporary table tablename(int id, varchar name(20));

Alter table cy modify column age varchar(20);
Update student set address = "tedg' where id=2;

Select * from student order by age desc;
Create view S_v as select id, name from students;(hide age)
Select * from S_v;
We use DISTINCT to avoid repetition.
If you will update anything in table it will update in view table as well (shallow copy)



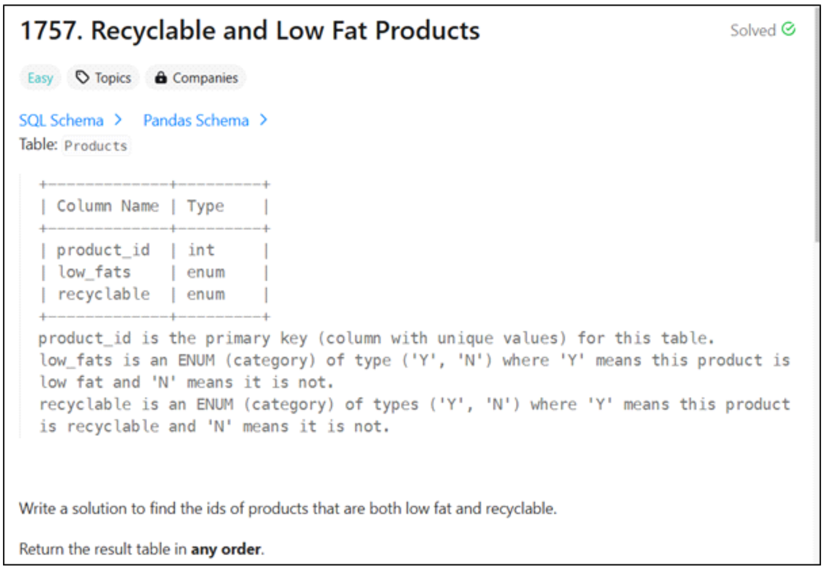
```
select * from customers
where (name like 'k%' or age >=25) and salary <3000;
```



Jinesh Ranawat 13:10

```
select * from customers;

select age,count(name) from customers group by age;

select address,avg(salary) from customers group by address;

select address,min(salary) as minsalary from customers group by address having minsalary >1500;

select address,max(salary) from customers group by address;

select * from customers
where (name like 'k%' or age >=25) and salary <3000;
```

select address, avg(salary) from CUSTOMERS group by address; (first group by works then avg)
NOTE in sql servr you have to mention aggregate function instead of as MN/anything with HAVING)
For cupboard rows-racks, compartments-shirt, pant, formal shirt



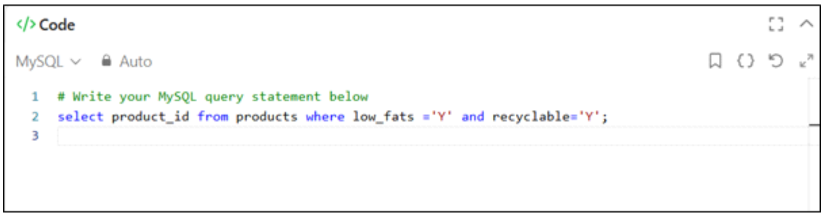
1757. Recyclable and Low Fat Products

Table: `Products`

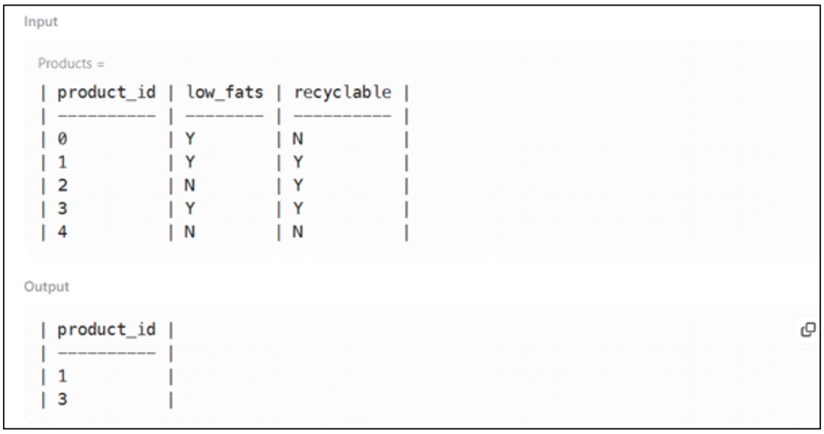
```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| product_id  | int     |
| low_fats    | enum    |
| recyclable  | enum    |
+-------------+---------+
product_id is the primary key (column with unique values) for this table.
low_fats is an ENUM (category) of type ('Y', 'N') where 'Y' means this product is
low fat and 'N' means it is not.
recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product
is recyclable and 'N' means it is not.
```

Write a solution to find the ids of products that are both low fat and recyclable.

Return the result table in **any order**.



```sql
# Write your MySQL query statement below
select product_id from products where low_fats ='Y' and recyclable='Y';

```



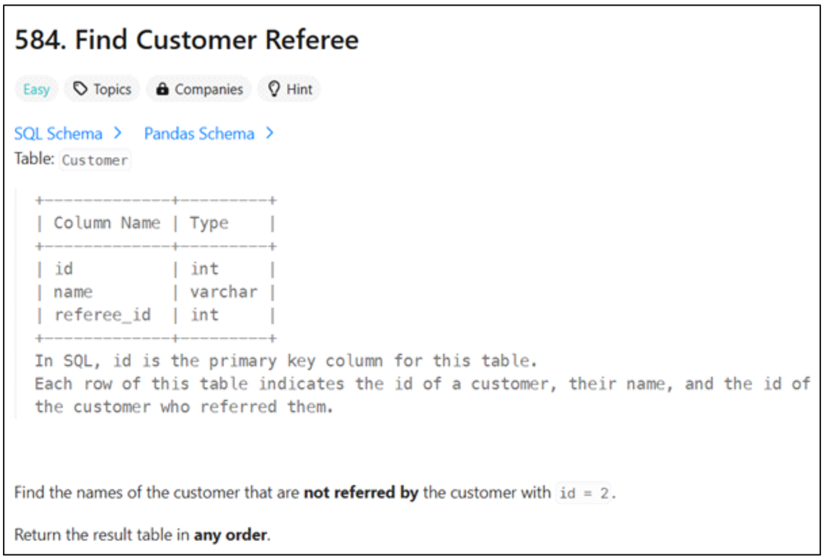
Input

Products =

```
| product_id | low_fats | recyclable |
| ---------- | -------- | ---------- |
| 0          | Y        | N          |
| 1          | Y        | Y          |
| 2          | N        | Y          |
| 3          | Y        | Y          |
| 4          | N        | N          |
```

Output

```
| product_id |
| ---------- |
| 1          |
| 3          |
```



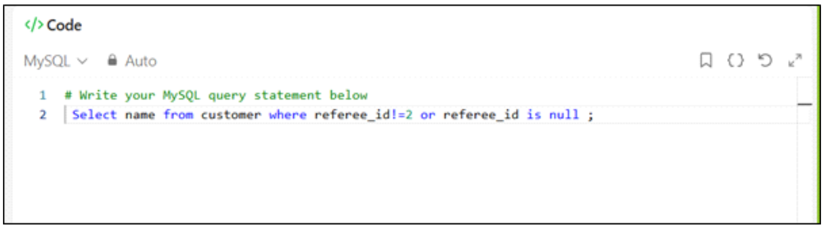
584. Find Customer Referee

Table: `Customer`

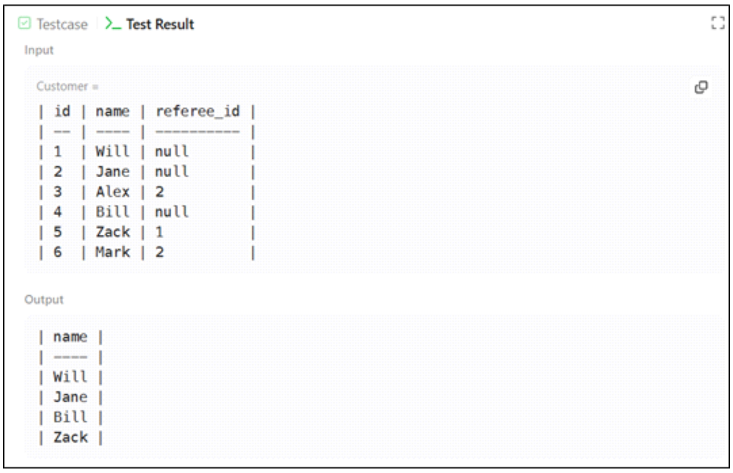
```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| referee_id  | int     |
+-------------+---------+
In SQL, id is the primary key column for this table.
Each row of this table indicates the id of a customer, their name, and the id of
the customer who referred them.
```

Find the names of the customer that are **not referred by** the customer with `id = 2`.

Return the result table in **any order**.



```
# Write your MySQL query statement below
Select name from customer where referee_id!=2 or referee_id is null ;
```



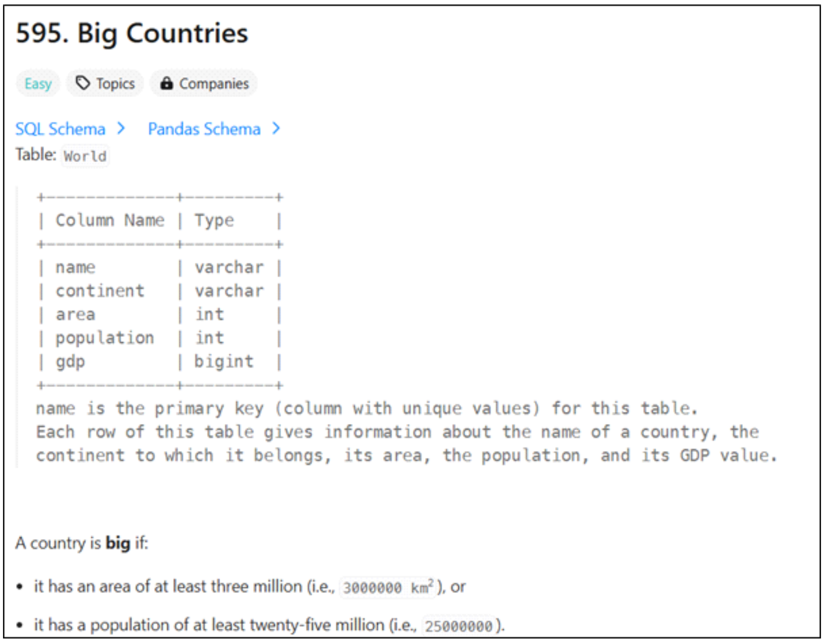
Input

Customer =

```
| 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 |
```



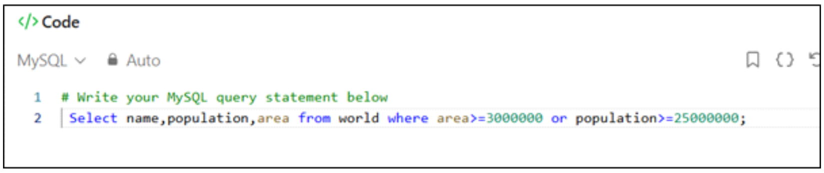
595. Big Countries

Table: `World`

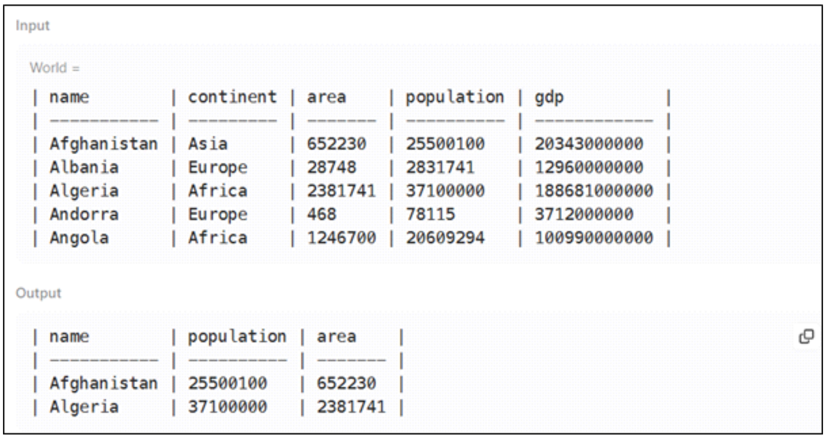
```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| name        | varchar |
| continent   | varchar |
| area        | int     |
| population  | int     |
| gdp         | bigint  |
+-------------+---------+
name is the primary key (column with unique values) for this table.
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`).



```sql
 # Write your MySQL query statement below
 Select name,population,area from world where area>=3000000 or population>=25000000;
```

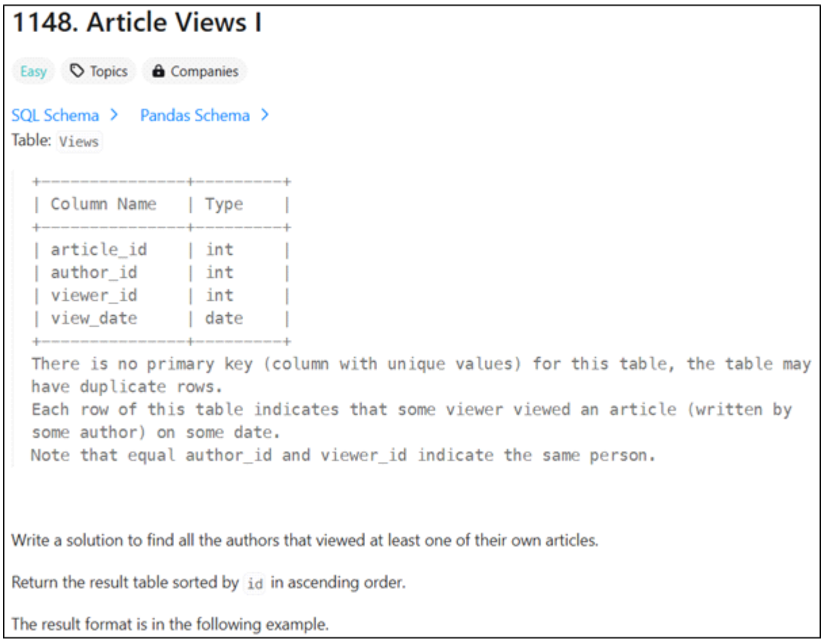


Input

```
World =
| 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 |
```



1148. Article Views I

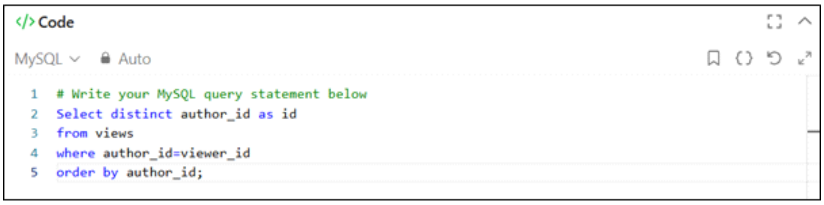
Table: `Views`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
There is no primary key (column with unique values) for this table, the table 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.
```

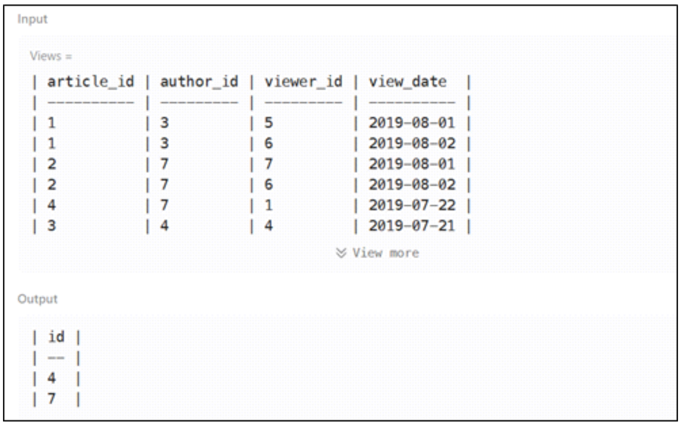
Write a solution to find all the authors that viewed at least one of their own articles.

Return the result table sorted by `id` in ascending order.

The result format is in the following example.



```sql
# Write your MySQL query statement below
Select distinct author_id as id
from views
where author_id=viewer_id
order by author_id;
```



Input

Views =

```
| 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 |
```

Output

```
| id |
| -- |
| 4  |
| 7  |
```

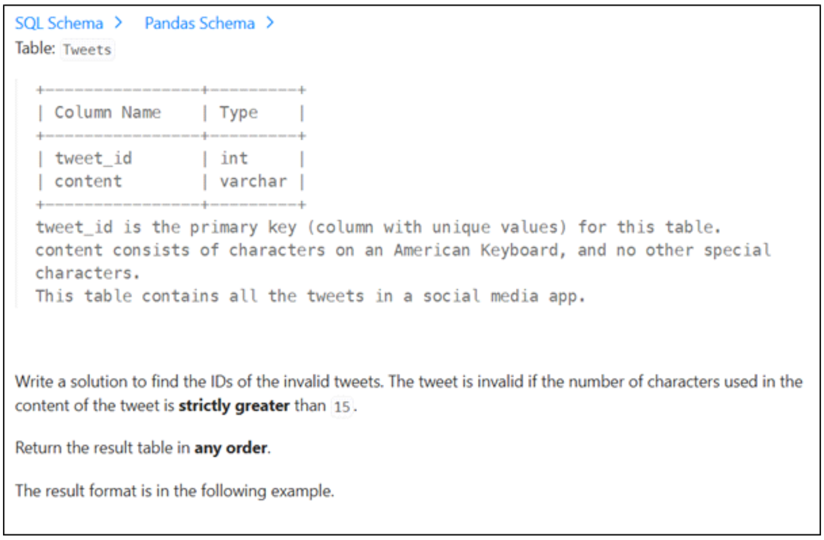


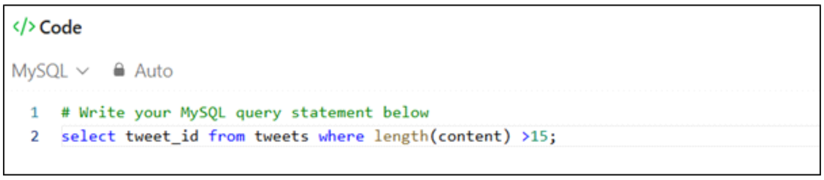
Table: `Tweets`

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| tweet_id       | int     |
| content        | varchar |
+----------------+---------+
tweet_id is the primary key (column with unique values) for this table.
content consists of characters on an American Keyboard, and no other special
characters.
This table contains all the tweets in a social media app.
```

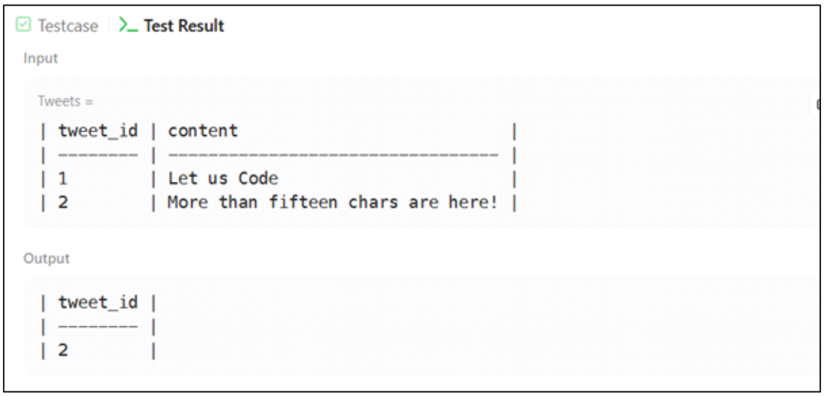
Write a solution to find the IDs of the invalid tweets. The tweet is invalid if the number of characters used in the content of the tweet is **strictly greater** than `15`.

Return the result table in **any order**.

The result format is in the following example.



```sql
# Write your MySQL query statement below
select tweet_id from tweets where length(content) >15;
```



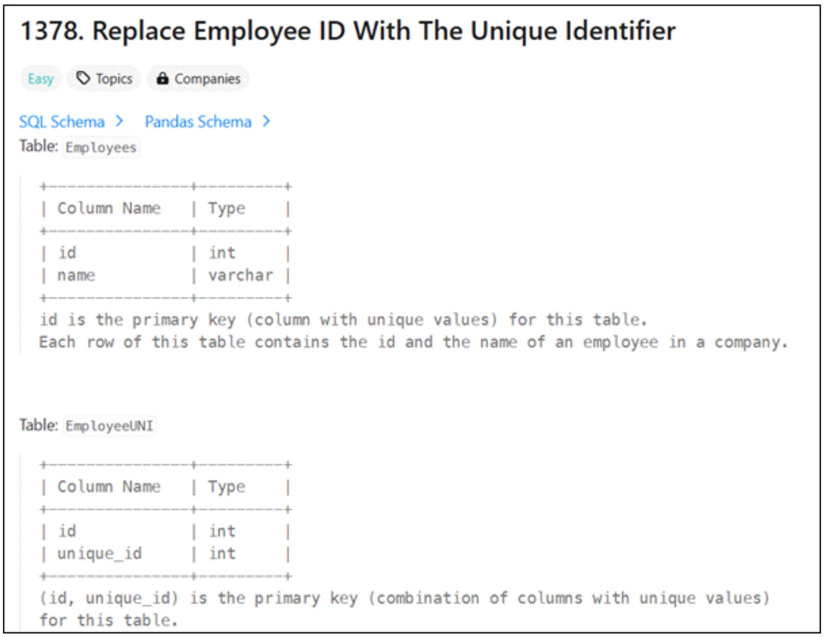
Input

Tweets =

```
| tweet_id | content                            |
| -------- | ---------------------------------- |
| 1        | Let us Code                        |
| 2        | More than fifteen chars are here!  |
```

Output

```
| tweet_id |
| -------- |
| 2        |
```



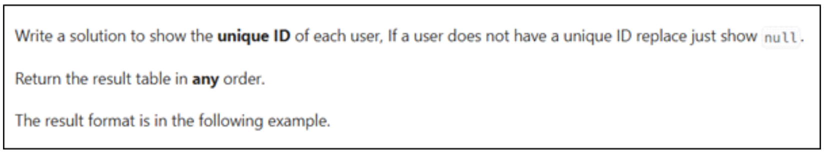
1378. Replace Employee ID With The Unique Identifier

Table: `Employees`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| name          | varchar |
+---------------+---------+
id is the primary key (column with unique values) for this table.
Each row of this table contains the id and the name of an employee in a company.
```

Table: `EmployeeUNI`

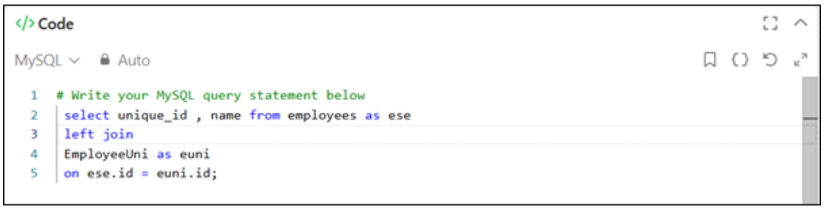
```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| unique_id     | int     |
+---------------+---------+
(id, unique_id) is the primary key (combination of columns with unique values)
for this table.
```



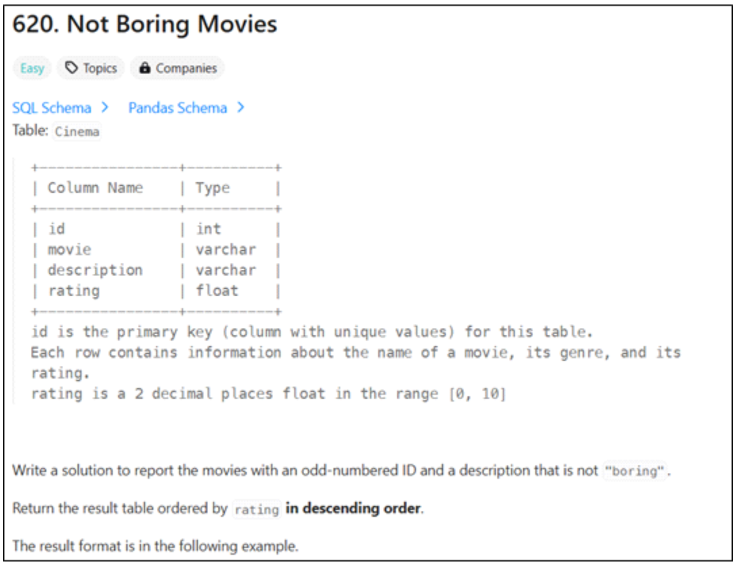
Write a solution to show the **unique ID** of each user, If a user does not have a unique ID replace just show `null`.

Return the result table in **any** order.

The result format is in the following example.



```sql
# Write your MySQL query statement below
select unique_id , name from employees as ese
left join
EmployeeUni as euni
on ese.id = euni.id;
```



620. Not Boring Movies

SQL Schema > Pandas Schema >

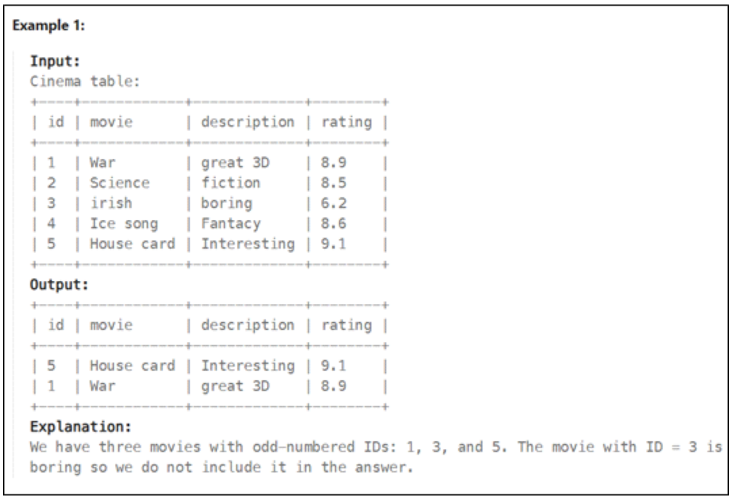
Table: `Cinema`

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| id             | int      |
| movie          | varchar  |
| description    | varchar  |
| rating         | float    |
+----------------+----------+
id is the primary key (column with unique values) for this table.
Each row contains information about the name of a movie, its genre, and its
rating.
rating is a 2 decimal places float in the range [0, 10]
```

Write a solution to report the movies with an odd-numbered ID and a description that is not `"boring"`.

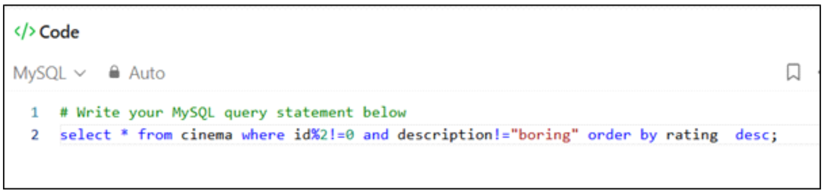
Return the result table ordered by `rating` **in descending order**.

The result format is in the following example.

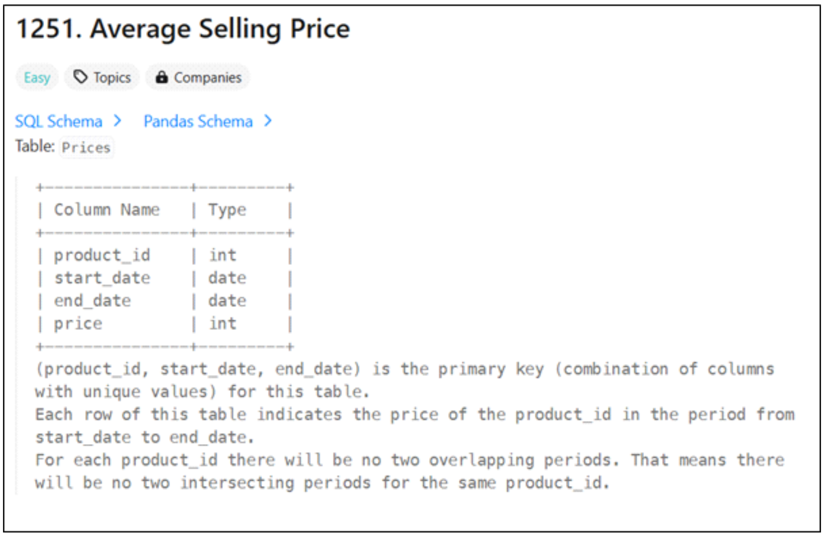


Example 1:

```
Input:
Cinema table:
+----+------------+-------------+--------+
| id | movie      | description | rating |
+----+------------+-------------+--------+
| 1  | War        | great 3D    | 8.9    |
| 2  | Science    | fiction     | 8.5    |
| 3  | irish      | boring      | 6.2    |
| 4  | Ice song   | Fantacy     | 8.6    |
| 5  | House card | Interesting | 9.1    |
+----+------------+-------------+--------+
Output:
+----+------------+-------------+--------+
| id | movie      | description | rating |
+----+------------+-------------+--------+
| 5  | House card | Interesting | 9.1    |
| 1  | War        | great 3D    | 8.9    |
+----+------------+-------------+--------+
Explanation:
We have three movies with odd-numbered IDs: 1, 3, and 5. The movie with ID = 3 is
boring so we do not include it in the answer.
```



```sql
# Write your MySQL query statement below
select * from cinema where id%2!=0 and description!="boring" order by rating  desc;
```



1251. Average Selling Price

SQL Schema > Pandas Schema >

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 (combination of columns
with unique values) 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.
```

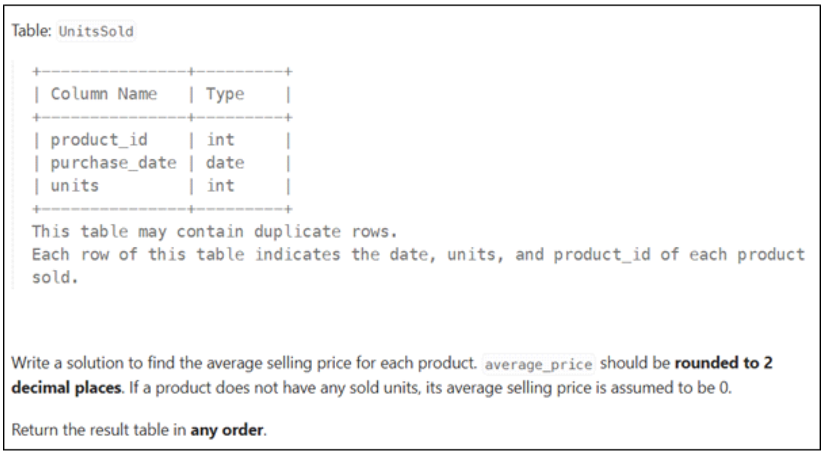
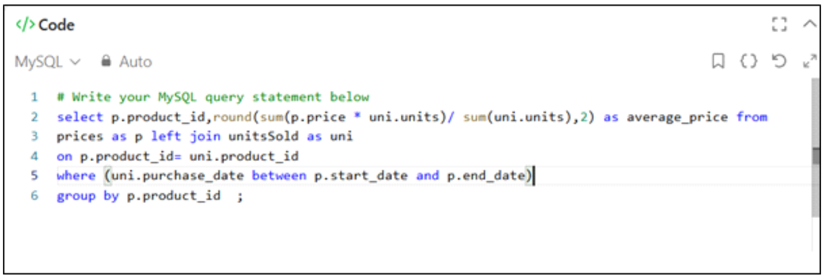


Table: `UnitsSold`

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| product_id    | int     |
| purchase_date | date    |
| units         | int     |
+---------------+---------+
This table may contain duplicate rows.
Each row of this table indicates the date, units, and product_id of each product
sold.
```

Write a solution to find the average selling price for each product. `average_price` should be **rounded to 2 decimal places**. If a product does not have any sold units, its average selling price is assumed to be 0.

Return the result table in **any order**.



```
# Write your MySQL query statement below
select p.product_id,round(sum(p.price * uni.units)/ sum(uni.units),2) as average_price from
prices as p left join unitsSold as uni
on p.product_id= uni.product_id
where (uni.purchase_date between p.start_date and p.end_date)
group by p.product_id  ;
```

SQL D2

12 February 2025 01:26

Bitwise operations, joins, ranking functions, and analytical queries.

```
CREATE TABLE permissions (
    user_id INT PRIMARY KEY,
    username VARCHAR(50),
    permission_flags INT  -- Stores permission bits
);
```

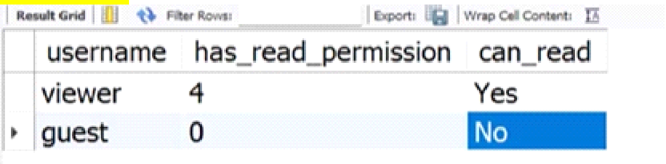
```
INSERT INTO permissions (user_id, username, permission_flags) VALUES
(1, 'admin', 7),     -- Binary: 111 (Read: 1, Write: 1, Execute: 1)
(2, 'developer', 6), -- Binary: 110 (Read: 1, Write: 1, Execute: 0)
(3, 'viewer', 4),    -- Binary: 100 (Read: 1, Write: 0, Execute: 0)
(4, 'guest', 1);     -- Binary: 001 (Read: 0, Write: 0, Execute: 1)
```

WRITE -- 4(100)
READ -- 2(010)
EXECUTE -- 1(001)

[1]Check which users have read permission (4)

```
select username, permission_flags & 4 as has_read_permission,
case when permission_flags & 4 > 0 then 'Yes' else 'No'
end as can_read from permissions;
```

OUTPUT

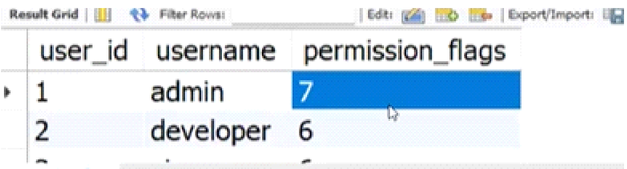


username	has_read_permission	can_read
viewer	4	Yes
guest	0	No

[2]Add write permission to all user who don't have it

```
update permissions
set permission_flags = permission_flags | 2
where (permission_flags & 2) =0

select * from permissions
```

OUTPUT

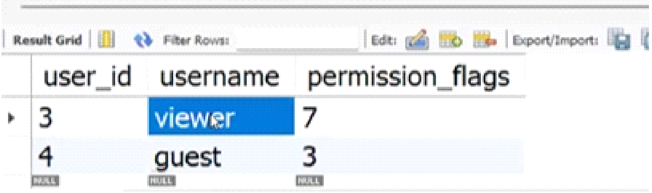


user_id	username	permission_flags
1	admin	7
2	developer	6

[3]Toggle the execute permission for user [XOR]
(i.e. if they have permission take that permission form them and if not give them that permission)

```
update permissions
set permission_flags =permission_flags ^ 1
where (permission_flags & 1)=0;
```

OUTPUT



user_id	username	permission_flags
3	viewer	7
4	guest	3
NULL	NULL	NULL

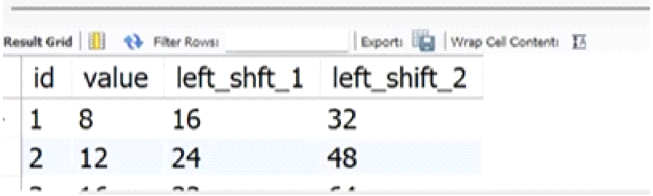
2. Bit Shifting Operations

```
CREATE TABLE bit_shift_demo (
   id INT PRIMARY KEY,
   value INT
);

INSERT INTO bit_shift_demo (id, value) VALUES
(1, 8),   -- Binary: 1000
(2, 12),  -- Binary: 1100
(3, 16);  -- Binary: 10000
```

Left Shift (Multiply by 2)

```
SELECT id, value, (value << 1) AS left_shift_1, (value << 2) AS left_shift_2 FROM
bit_shift_demo;
```

OUTPUT



id	value	left_shft_1	left_shift_2
1	8	16	32
2	12	24	48

Right Shift (Divide by 2)

```
SELECT id, value, (value >> 1) AS right_shift_1, (value >> 2) AS right_shift_2 FROM
bit_shift_demo;
```

3. SQL Clauses (NOT, BETWEEN, EXISTS)

```
CREATE TABLE Customers (
  CustomerID INT PRIMARY KEY,
  Name VARCHAR(100),
  Country VARCHAR(50),
  IsActive BIT,
  CreditLimit DECIMAL(10,2)
);

CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  CustomerID INT,
  OrderDate DATE,
  TotalAmount DECIMAL(10,2),
  Status VARCHAR(20)
);
```

Find products in stock (NOT)

```
SELECT * FROM Products WHERE InStock != 0;
SELECT * FROM Products WHERE InStock <> 0;
```

Find orders with amount between 1000-2000 (BETWEEN)

```
SELECT * FROM Orders WHERE TotalAmount BETWEEN 1000 AND 2000;
```

Find customers with at least one order (EXISTS)

```
SELECT Name
FROM Customers C
WHERE EXISTS (SELECT 1 FROM Orders O WHERE O.CustomerID = C.CustomerID);
```

[3] SQL Joins (INNER, LEFT, RIGHT)

Create Table

```
CREATE TABLE Employees (EmpID INT PRIMARY KEY, Name VARCHAR(50), DeptID INT);
CREATE TABLE Departments (DeptID INT PRIMARY KEY, DeptName VARCHAR(50));
```

Inner Join (Common records only)

```
SELECT E.Name, D.DeptName
FROM Employees E
INNER JOIN Departments D ON E.DeptID = D.DeptID;
```

Left Join (All Employees + Matching Departments)

```
SELECT E.Name, D.DeptName
FROM Employees E
LEFT JOIN Departments D ON E.DeptID = D.DeptID;
```

Right Join (All Departments + Matching Employees)

```
SELECT E.Name, D.DeptName
FROM Employees E
RIGHT JOIN Departments D ON E.DeptID = D.DeptID;
```

```
CREATE TABLE Customers (
        CustomerID INT PRIMARY KEY,
         Name VARCHAR(100),
        Country VARCHAR(50),
        IsActive BIT,
         CreditLimit DECIMAL(10,2)
);

 CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderDate DATE,
   TotalAmount DECIMAL(10,2),
   Status VARCHAR(20)
);

 CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductName VARCHAR(100),
   Category VARCHAR(50),
   Price DECIMAL(10,2),
   InStock BIT
);



CREATE TABLE OrderDetails (
   OrderID INT,
   ProductID INT,
   Quantity INT,
   UnitPrice DECIMAL(10,2),
   PRIMARY KEY (OrderID, ProductID)
);

INSERT INTO Customers VALUES
```

```
(1, 'John Doe', 'USA', 1, 5000.00),
(2, 'Jane Smith', 'Canada', 1, 3000.00),
(3, 'Bob Johnson', 'USA', 0, 2000.00),
(4, 'Alice Brown', 'UK', 1, 4000.00),
(5, 'Charlie Wilson', 'Canada', 1, 6000.00);
```

```
INSERT INTO Orders VALUES
(1, 1, '2024-01-01', 1500.00, 'Delivered'),
(2, 1, '2024-01-15', 2000.00, 'Pending'),
(3, 2, '2024-01-20', 1000.00, 'Delivered'),
(4, 3, '2024-02-01', 500.00, 'Cancelled'),
(5, 4, '2024-02-15', 3000.00, 'Processing');
```

```
INSERT INTO Products VALUES
(1, 'Laptop', 'Electronics', 1200.00, 1),
(2, 'Smartphone', 'Electronics', 800.00, 1),
(3, 'Desk Chair', 'Furniture', 200.00, 0),
(4, 'Coffee Maker', 'Appliances', 100.00, 1),
(5, 'Headphones', 'Electronics', 150.00, 1);
```

```
INSERT INTO OrderDetails VALUES
(1, 1, 1, 1200.00),
(1, 2, 1, 800.00),
(2, 3, 2, 200.00),
(3, 4, 1, 100.00),
(4, 5, 2, 150.00);
```

```
select distinct c.Name,c.Country from Customers c
join orders o on  c.customerID =o.customerID
where c.country<>'USA' and o.totalamount >
ANY(
select totalamount from orders o2
join customers c2 on c2.customerID=o2.customerId
where c2.country='USA'
);
```

CATEGORISING THE CUSTOMER BASED ON CREDITLIMIT -

```
SELECT NAME,
CASE
WHEN CreditLimit >=5000 THEN "PREMIUM CUSTOMER"
WHEN CreditLimit >=3000 THEN "GOLD CUSTOMER"
ELSE "STANDARD CUSTOMER"
END AS CUSTOMER_CATEGORY
FROM CUSTOMERS;
```

LAG() FUNCTION

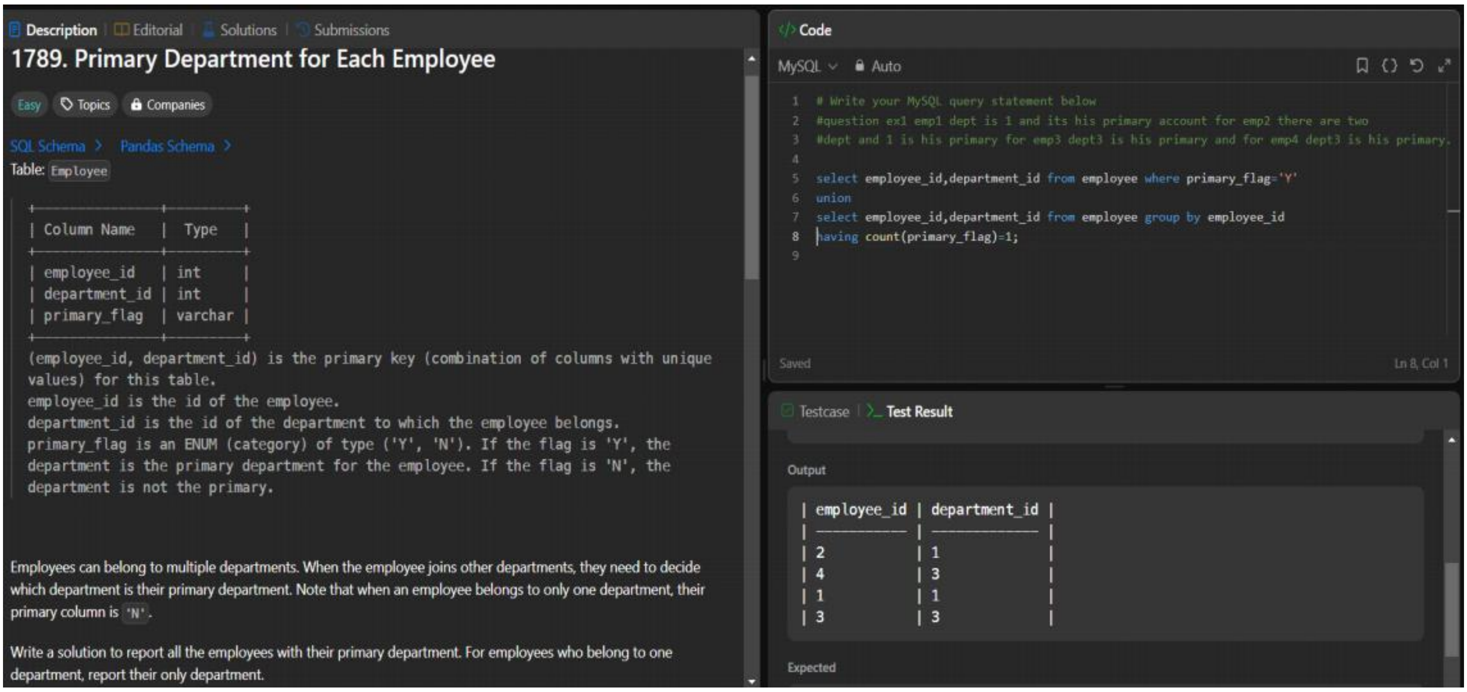
Used to compare for example like sales of a company from previous month to this month

Rank() and Dense Rank()

Difference is that if two values have same rank then rank will skip by one but dense rank will not skip any ranking.

SQL D3

12 February 2025 02:37



1789. Primary Department for Each Employee

Table: `Employee`

```
+---------------+---------+
| Column Name   |  Type   |
+---------------+---------+
| employee_id   | int     |
| department_id | int     |
| primary_flag  | varchar |
+---------------+---------+
(employee_id, department_id) is the primary key (combination of columns with unique
values) for this table.
employee_id is the id of the employee.
department_id is the id of the department to which the employee belongs.
primary_flag is an ENUM (category) of type ('Y', 'N'). If the flag is 'Y', the
department is the primary department for the employee. If the flag is 'N', the
department is not the primary.
```

Employees can belong to multiple departments. When the employee joins other departments, they need to decide which department is their primary department. Note that when an employee belongs to only one department, their primary column is `'N'`.

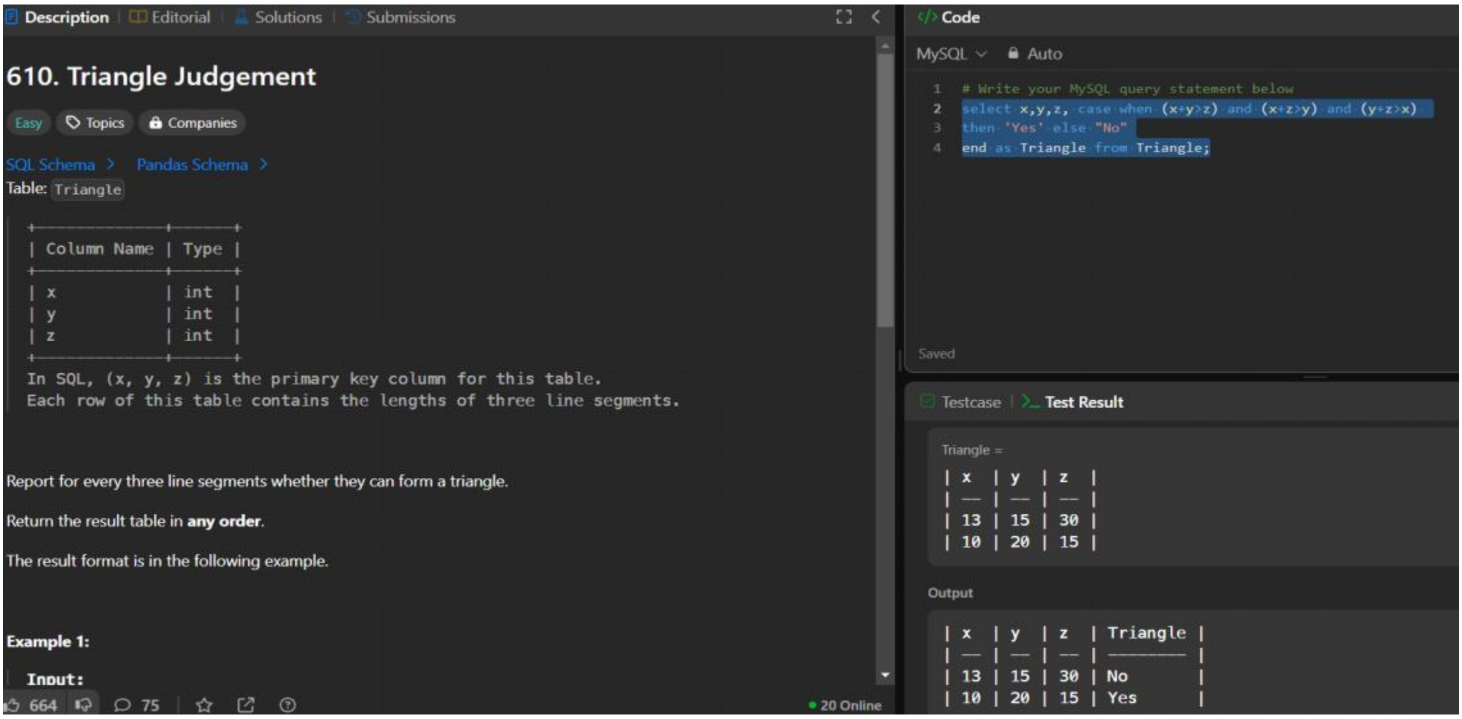
Write a solution to report all the employees with their primary department. For employees who belong to one department, report their only department.

```sql
# Write your MySQL query statement below
#question ex1 emp1 dept is 1 and its his primary account for emp2 there are two
#dept and 1 is his primary for emp3 dept3 is his primary and for emp4 dept3 is his primary.

select employee_id,department_id from employee where primary_flag='Y'
union
select employee_id,department_id from employee group by employee_id
having count(primary_flag)=1;
```

Output

```
| employee_id | department_id |
| ----------- | ------------- |
| 2           | 1             |
| 4           | 3             |
| 1           | 1             |
| 3           | 3             |
```

Expected



610. Triangle Judgement

Table: `Triangle`

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| x           | int  |
| y           | int  |
| z           | int  |
+-------------+------+
In SQL, (x, y, z) is the primary key column for this table.
Each row of this table contains the lengths of three line segments.
```

Report for every three line segments whether they can form a triangle.

Return the result table in **any order**.

The result format is in the following example.

Example 1:

Input:

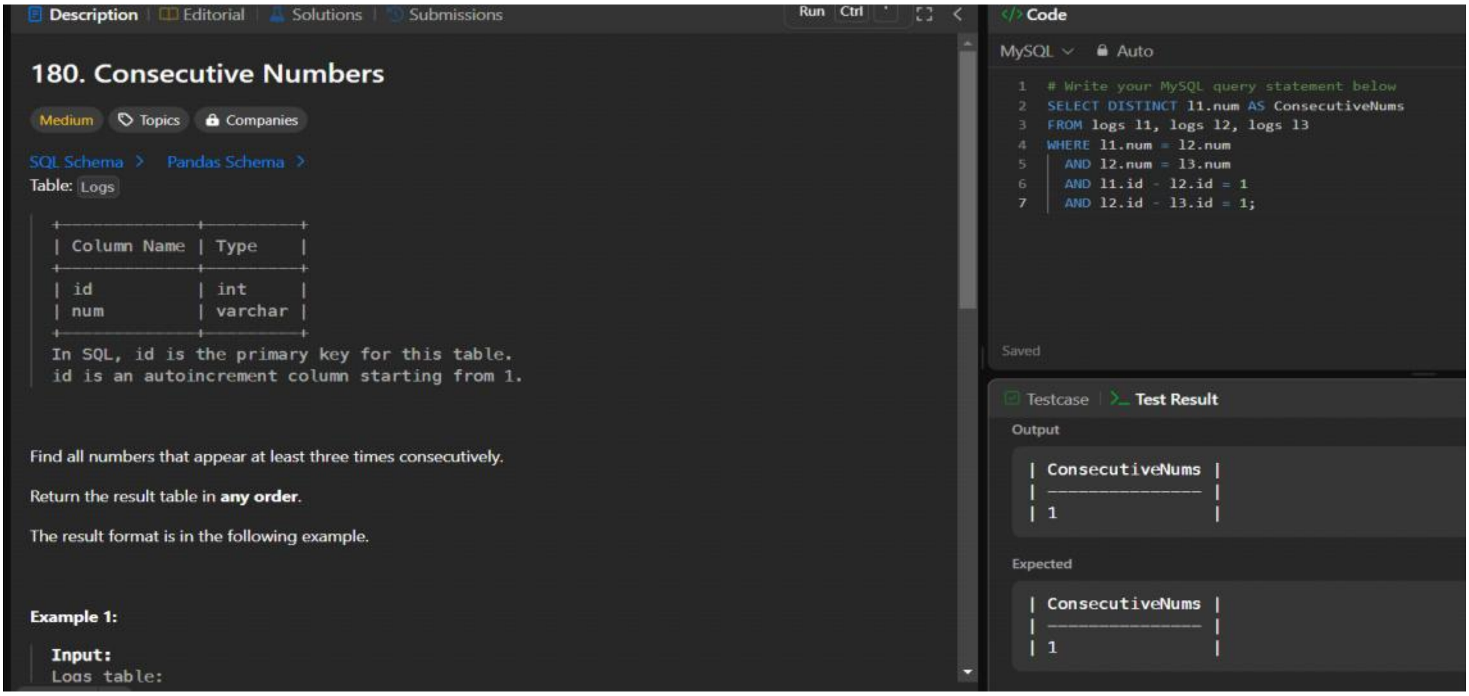
```sql
# Write your MySQL query statement below
select x,y,z, case when (x+y>z) and (x+z>y) and (y+z>x)
then 'Yes' else "No"
end as Triangle from Triangle;
```

Triangle =

```
| x  | y  | z  |
| -- | -- | -- |
| 13 | 15 | 30 |
| 10 | 20 | 15 |
```

Output

```
| x  | y  | z  | Triangle |
| -- | -- | -- | -------- |
| 13 | 15 | 30 | No       |
| 10 | 20 | 15 | Yes      |
```



180. Consecutive Numbers

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: `Logs`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| num         | varchar |
+-------------+---------+
In SQL, id is the primary key for this table.
id is an autoincrement column starting from 1.
```

Find all numbers that appear at least three times consecutively.

Return the result table in **any order**.

The result format is in the following example.

Example 1:

```
Input:
Logs table:
```

```sql
# Write your MySQL query statement below
SELECT DISTINCT l1.num AS ConsecutiveNums
FROM logs l1, logs l2, logs l3
WHERE l1.num = l2.num
  AND l2.num = l3.num
  AND l1.id - l2.id = 1
  AND l2.id - l3.id = 1;
```

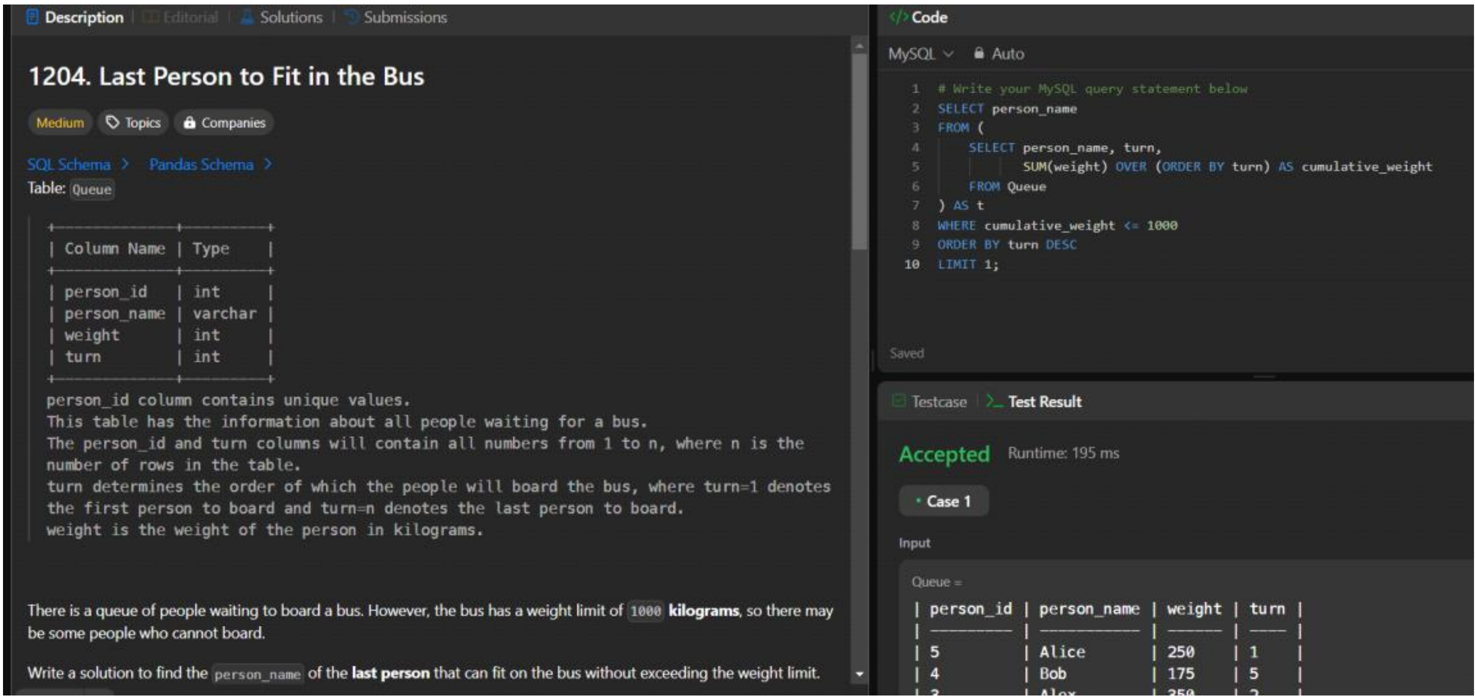
Saved

Output

```
| ConsecutiveNums |
| --------------- |
| 1               |
```

Expected

```
| ConsecutiveNums |
| --------------- |
| 1               |
```



1204. Last Person to Fit in the Bus

Medium Topics Companies

SQL Schema > Pandas Schema >

Table: `Queue`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| person_id   | int     |
| person_name | varchar |
| weight      | int     |
| turn        | int     |
+-------------+---------+
person_id column contains unique values.
This table has the information about all people waiting for a bus.
The person_id and turn columns will contain all numbers from 1 to n, where n is the
number of rows in the table.
turn determines the order of which the people will board the bus, where turn=1 denotes
the first person to board and turn=n denotes the last person to board.
weight is the weight of the person in kilograms.
```

There is a queue of people waiting to board a bus. However, the bus has a weight limit of `1000` **kilograms**, so there may be some people who cannot board.

Write a solution to find the `person_name` of the **last person** that can fit on the bus without exceeding the weight limit.

```sql
# Write your MySQL query statement below
SELECT person_name
FROM (
    SELECT person_name, turn,
           SUM(weight) OVER (ORDER BY turn) AS cumulative_weight
    FROM Queue
) AS t
WHERE cumulative_weight <= 1000
ORDER BY turn DESC
LIMIT 1;
```

Saved

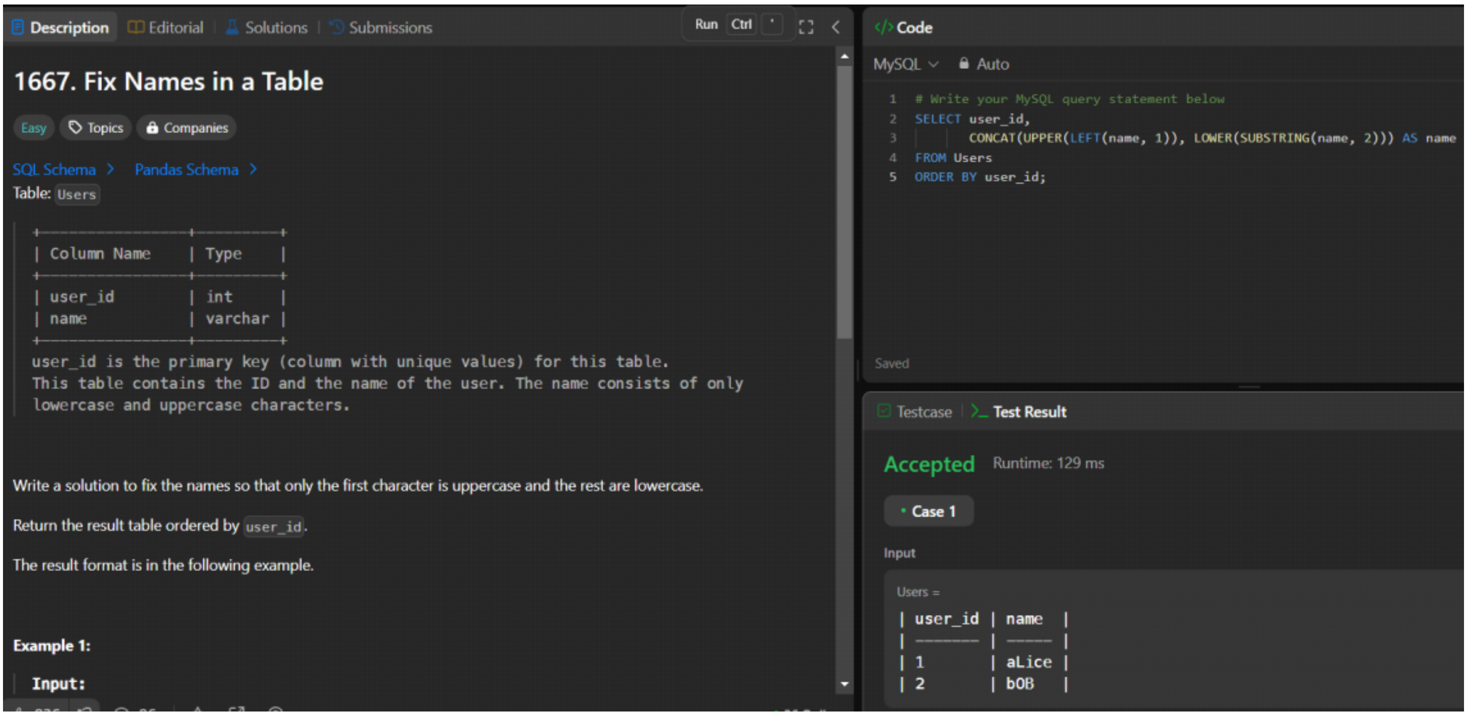
Accepted Runtime: 195 ms

• Case 1

Input

Queue =

```
| person_id | person_name | weight | turn |
| --------- | ----------- | ------ | ---- |
| 5         | Alice       | 250    | 1    |
| 4         | Bob         | 175    | 5    |
| 3         | Alex        | 350    | 2    |
```



1667. Fix Names in a Table

Easy · Topics · Companies

SQL Schema > Pandas Schema >

Table: `Users`

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| user_id        | int     |
| name           | varchar |
+----------------+---------+
user_id is the primary key (column with unique values) for this table.
This table contains the ID and the name of the user. The name consists of only
lowercase and uppercase characters.
```

Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase.

Return the result table ordered by `user_id`.

The result format is in the following example.

Example 1:

Input:

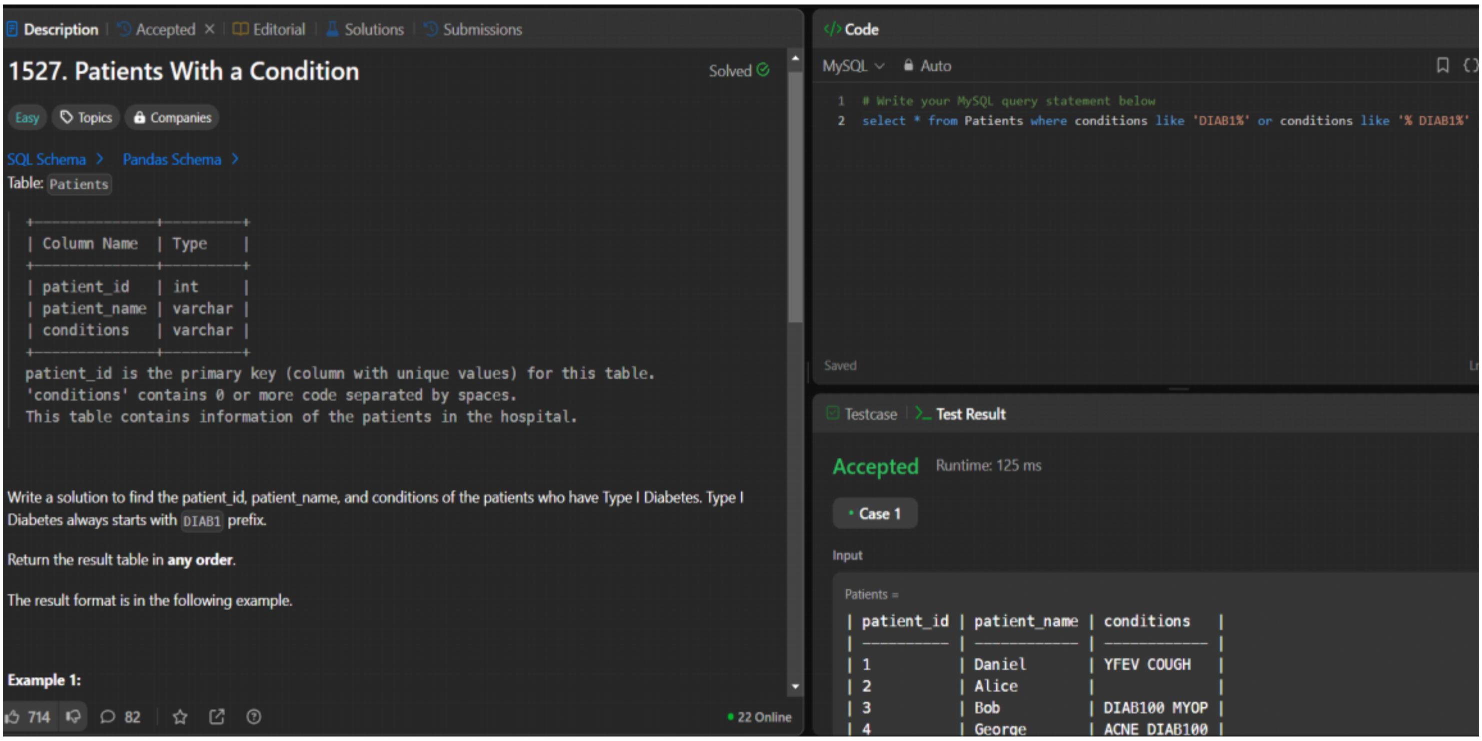
```
# Write your MySQL query statement below
SELECT user_id,
        CONCAT(UPPER(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name
FROM Users
ORDER BY user_id;
```

Accepted Runtime: 129 ms

Case 1

Input

```
Users =
| user_id | name  |
| ------- | ----- |
| 1       | aLice |
| 2       | bOB   |
```


Description | Accepted × | Editorial | Solutions | Submissions
Code
MySQL Auto
Saved
Testcase | Test Result
714
82
22 Online

1527. Patients With a Condition

Solved ✓

Easy · Topics · Companies

SQL Schema > Pandas Schema >

Table: `Patients`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| patient_id   | int     |
| patient_name | varchar |
| conditions   | varchar |
+--------------+---------+
patient_id is the primary key (column with unique values) for this table.
'conditions' contains 0 or more code separated by spaces.
This table contains information of the patients in the hospital.
```

Write a solution to find the patient_id, patient_name, and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with `DIAB1` prefix.

Return the result table in **any order**.

The result format is in the following example.

Example 1:

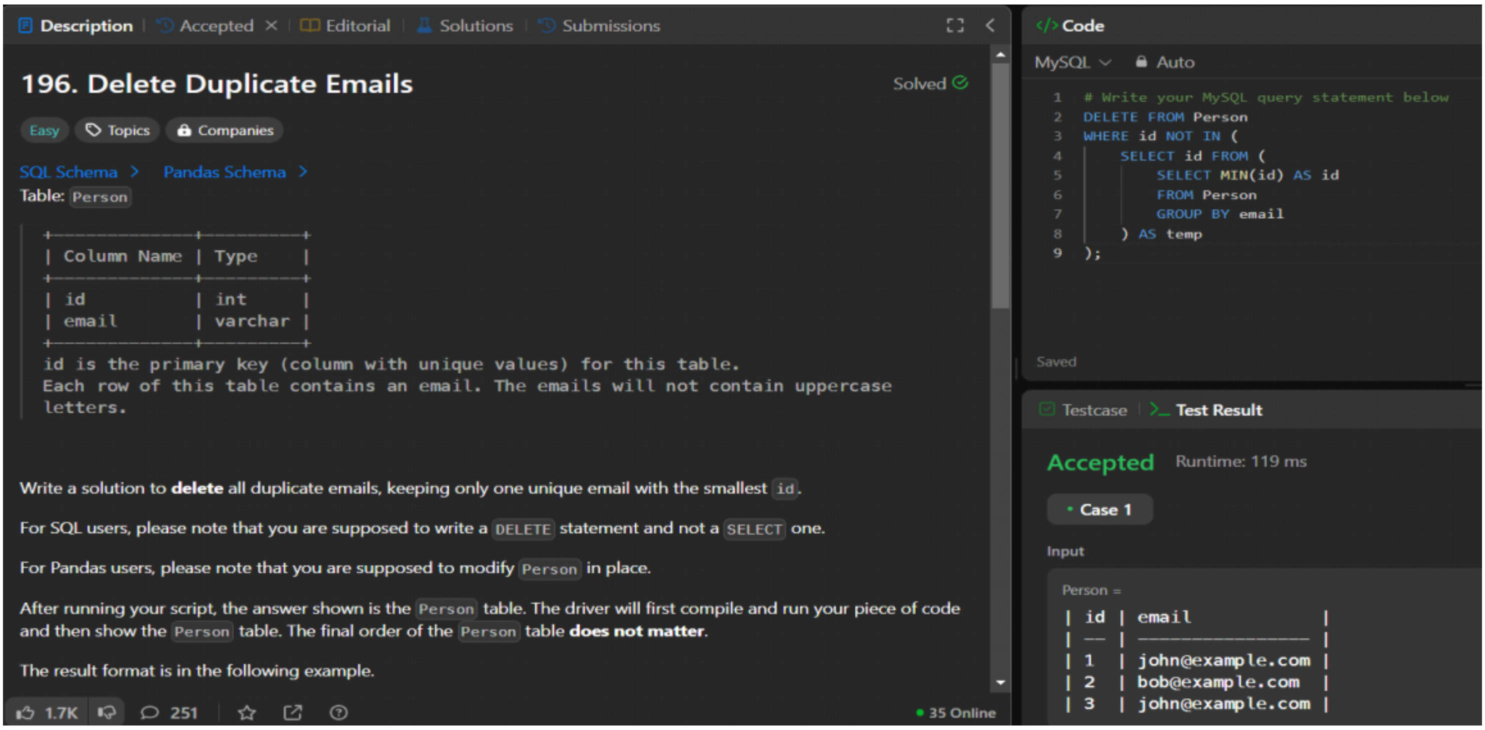
```
# Write your MySQL query statement below
select * from Patients where conditions like 'DIAB1%' or conditions like '% DIAB1%'
```

Accepted Runtime: 125 ms

Case 1

Input

```
Patients =
| patient_id | patient_name | conditions   |
| ---------- | ------------ | ------------ |
| 1          | Daniel       | YFEV COUGH   |
| 2          | Alice        |              |
| 3          | Bob          | DIAB100 MYOP |
| 4          | George       | ACNE DIAB100 |
```


Description | Accepted × | Editorial | Solutions | Submissions

196. Delete Duplicate Emails

Solved ✓

Easy | Topics | Companies

SQL Schema > Pandas Schema >

Table: `Person`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| email       | varchar |
+-------------+---------+
id is the primary key (column with unique values) for this table.
Each row of this table contains an email. The emails will not contain uppercase
letters.
```

Write a solution to **delete** all duplicate emails, keeping only one unique email with the smallest `id`.

For SQL users, please note that you are supposed to write a `DELETE` statement and not a `SELECT` one.

For Pandas users, please note that you are supposed to modify `Person` in place.

After running your script, the answer shown is the `Person` table. The driver will first compile and run your piece of code and then show the `Person` table. The final order of the `Person` table **does not matter**.

The result format is in the following example.

Code

MySQL | Auto

```
# Write your MySQL query statement below
DELETE FROM Person
WHERE id NOT IN (
    SELECT id FROM (
        SELECT MIN(id) AS id
        FROM Person
        GROUP BY email
    ) AS temp
);
```

Saved

Accepted Runtime: 119 ms

Case 1

Input

```
Person =
| id | email            |
| -- | ---------------- |
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
```

Foreign Key

A foreign key is a column or group of columns in a table that links it to another table's primary key

```
CREATE TABLE departments (
   department_id INT PRIMARY KEY,
   department_name VARCHAR(100) NOT NULL
);
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(100) NOT NULL,
   department_id INT,
   FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

NOTE if you want to delete some values from foreign key table it will not delete it untill you unrefer it or delete from the reference table

A shallow copy typically refers to copying the structure without data, while a deep copy refers to copying both structure and data.

Normalization in SQL is a process to organize data in a database efficiently by reducing redundancy and improving data integr ity. It involves creating multiple related tables and establishing relationships using **keys**.

1NF

Rule

Eliminate repeating groups (store atomic values).

- Ensure each column contains a single value (no lists or arrays).
- All rows must be uniquely identifiable (implicitly or explicitly).

Example:

A table storing multiple courses in one column (Courses: "Math, Physics") violates 1NF.

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    Course VARCHAR(50) -- No repeating columns like Course1, Course2
  );
```

2NF

Rule

- Must be in 1NF.
- Remove partial dependencies: Non-key attributes must depend on the entire primary key (not a subset).

Example:
A table with a composite key (StudentID, Course) and ProfessorName (dependent only on Course) violates 2NF.

```
-- Students Table
  CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50)
  );

  -- Courses Table
  CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(50)
  );

  -- Enrollment Table (Composite Key)
  CREATE TABLE Enrollment (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
  );
```

3NF

Rule

- Must be in 2NF.
- Remove transitive dependencies: Non-key attributes must not depend on other non-key attributes.

Example:
A table with OrderID → CustomerID → CustomerCity violates 3NF.

```
-- Split a table with redundant data:
  CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
  );

  CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    City VARCHAR(50) -- No redundant "City" data in Orders table
  );
```

BCNF

Rule

- Must be in 3NF.
- Every determinant(left side of a functional dependency) must be a superkey(a candidate key).

Example:
A table with `Professor` → `Course` (but `Professor` is not a key) violates BCNF

```
CREATE TABLE ProfessorCourses (
    Professor VARCHAR(50) PRIMARY KEY, -- Superkey
    Course VARCHAR(50)
  );
CREATE TABLE StudentEnrollment (
    StudentID INT,
    Professor VARCHAR(50),
    PRIMARY KEY (StudentID, Professor),
   FOREIGN KEY (Professor) REFERENCES ProfessorCourses(Professor)
  );
```

FORIEGN KEY

Foreign Key with ON DELETE and ON UPDATE Actions

```
CREATE TABLE Employees (
   EmpID INT PRIMARY KEY,
   Name VARCHAR(50),
   DeptID INT,
   FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
   ON DELETE CASCADE
   ON UPDATE CASCADE;
);
```

ON DELETE CASCADE → If a department is deleted, all employees in that department are also deleted.

ON UPDATE CASCADE → If a department ID is updated, the update is reflected in Employees.

ON DELETE Actions

Action	Effect
CASCADE	Deletes child records when the parent record is deleted.
SET NULL	Sets the foreign key column in the child table to `NULL`.
RESTRICT	Prevents deletion if references exist.
NO ACTION	Similar to `RESTRICT` but enforced at the end of a transaction.

ON UPDATE Actions

Action	Effect
CASCADE	Updates foreign key values in child table if parent key changes.
SET NULL	Sets foreign key in the child table to `NULL` when parent key updates.
RESTRICT	Prevents update if foreign key references exist.
NO ACTION	Similar to `RESTRICT` but enforced at the end of a transaction.

Stored procedure

A stored procedure in SQL is like a pre-saved SQL script that you can reuse anytime. It’s a way to store a set of SQL commands under a name and run them with a single call. Think of it as a custom function for your database.

- What: A named block of SQL code stored in the database.
- Why: Avoid rewriting the same code repeatedly. Perform complex tasks in one step.
- How: Define it once, then execute it by name (e.g., `EXEC GetUsers`).

Example:

Create a stored procedure to fetch all users from a table:

```
-- Create the procedure
CREATE PROCEDURE GetUsers
AS
BEGIN
  SELECT * FROM Users;
END;
```

Run it:

```
EXEC GetUsers;  -- Executes the procedure
```

Example with Parameters:

Create a procedure to fetch users by country:

```
CREATE PROCEDURE GetUsersByCountry
  @Country VARCHAR(50)  -- Input parameter
AS
```

~~~
BEGIN
  SELECT * FROM Users WHERE Country = @Country;
END;
```
~~~

Run it:

~~~
EXEC GetUsersByCountry @Country = 'USA';
~~~

~~~
DELIMITER //

CREATE procedure sp_insert_student(
in p_first_name varchar(50),
in p_last_name varchar(50),
in p_email varchar(100)
)
BEGIN
insert into students(first_name,last_name,email, enrollment_date)
values(p_first_name,p_last_name,p_email,CURDATE());
select last_insert_id() AS STUDENT_ID;
END //
DELIMITER;
~~~

ACID Properties

Property	What It Means	Real-World Analogy
Atomicity	**"All or Nothing"** – The entire transaction happens completely or not at all.	Like a light switch: it's either ON or OFF.
Consistency	**"Follow the Rules"** – Data stays valid (e.g., no negative balances in a bank).	Like a rulebook the database must obey.
Isolation	**"No Interference"** – Transactions don't step on each other's toes.	Like separate checkout lanes in a grocery store.
Durability	**"Permanent Once Done"** – Changes survive crashes/power loss.	Like writing with permanent ink.

Example: Bank Transfer

1. Atomicity:
  - Transfer $100 from Alice to Bob.
  - Both steps must happen:
    - Deduct $100 from Alice's account.
    - Add $100 to Bob's account.
  - If either step fails, both are canceled (no partial updates).

2. Consistency:
  - Total money in the system remains the same (e.g., $100 moved, not created/destroyed).
  - No invalid data (e.g., Alice's balance can't go negative).

3. Isolation:

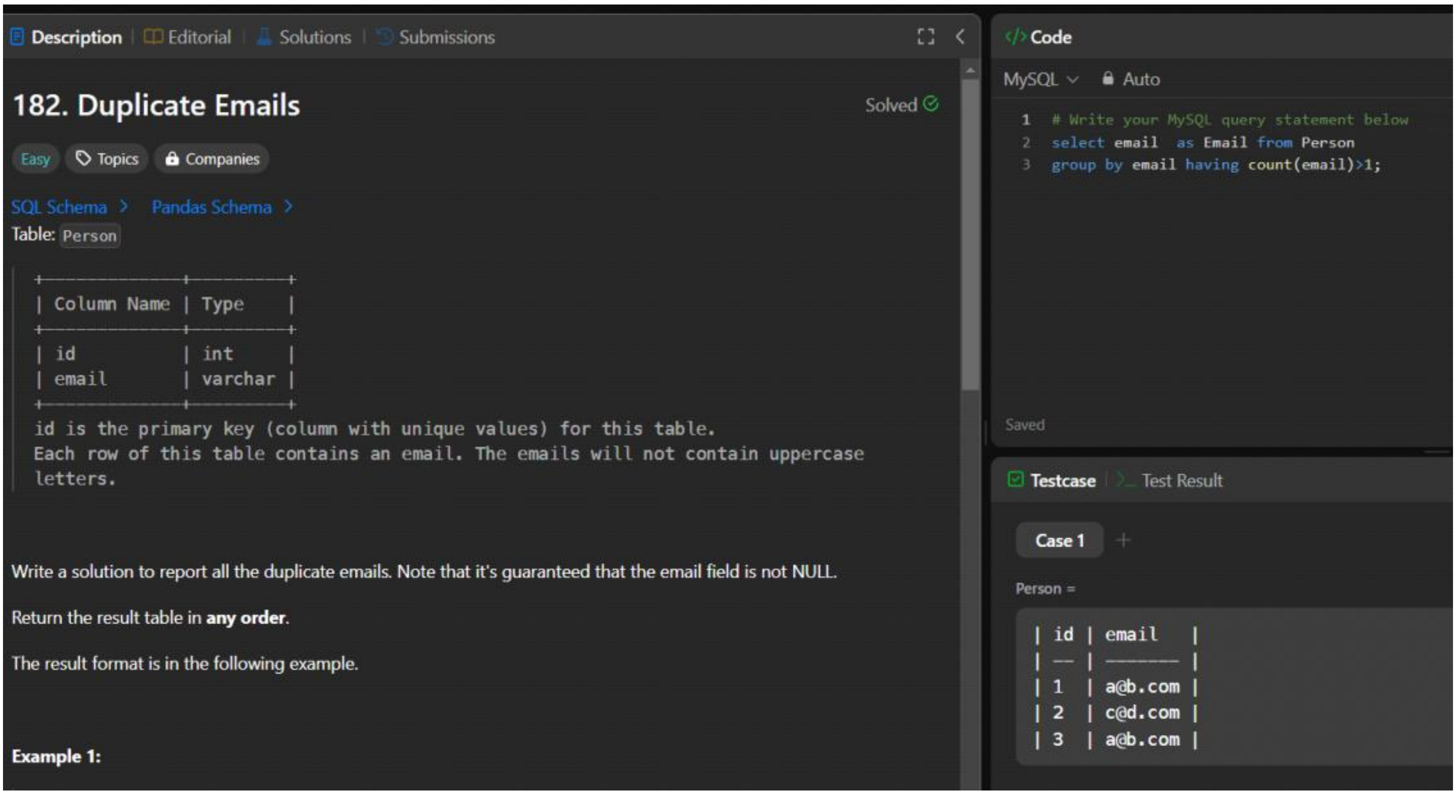
  - If Bob checks his balance mid-transfer, he sees either the old or new balance (not a half-updated value).
  - Another transfer between Alice and Charlie happens separately, without mixing data.

4. Durability:
  - Once the transfer succeeds, the changes are saved permanently (even if the bank's system crashes).

ACID in SQL
```
BEGIN TRANSACTION;  -- Start a transaction
  UPDATE Accounts SET Balance = Balance - 100 WHERE Name = 'Alice';
  UPDATE Accounts SET Balance = Balance + 100 WHERE Name = 'Bob';
COMMIT;  -- Finalize changes (ACID enforced here)
```
- If anything fails between `BEGIN` and `COMMIT`, the database automatically rolls back (Atomicity).

SQL D4

13 February 2025 16:35



182. Duplicate Emails

Solved

Easy Topics Companies

SQL Schema > Pandas Schema >

Table: `Person`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| email       | varchar |
+-------------+---------+
id is the primary key (column with unique values) for this table.
Each row of this table contains an email. The emails will not contain uppercase
letters.
```

Write a solution to report all the duplicate emails. Note that it's guaranteed that the email field is not NULL.

Return the result table in **any order**.

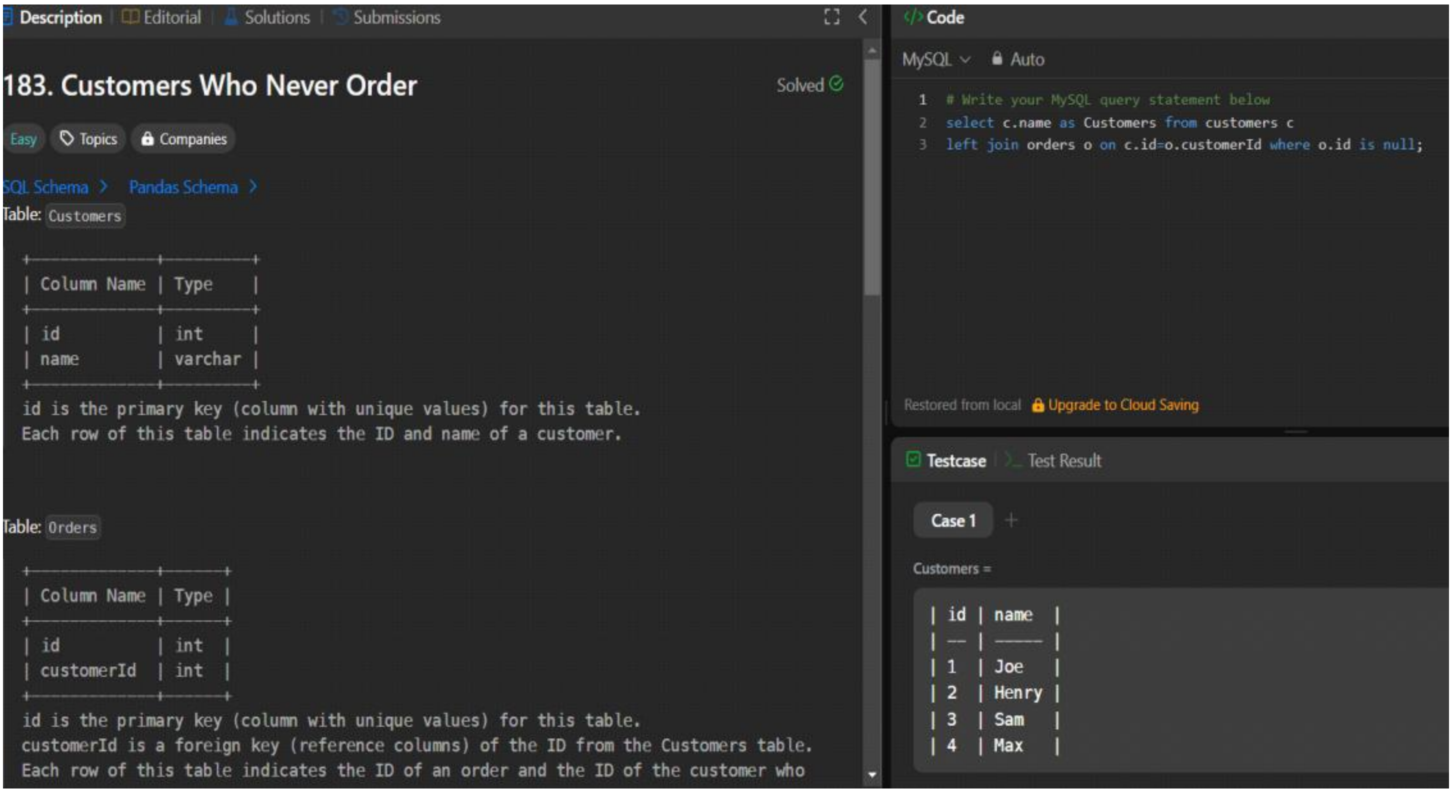
The result format is in the following example.

Example 1:

```sql
# Write your MySQL query statement below
select email  as Email from Person
group by email having count(email)>1;
```

Person =

id	email
1	a@b.com
2	c@d.com
3	a@b.com



183. Customers Who Never Order

Solved

Easy Topics Companies

SQL Schema > Pandas Schema >

Table: `Customers`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
+-------------+---------+
id is the primary key (column with unique values) for this table.
Each row of this table indicates the ID and name of a customer.
```

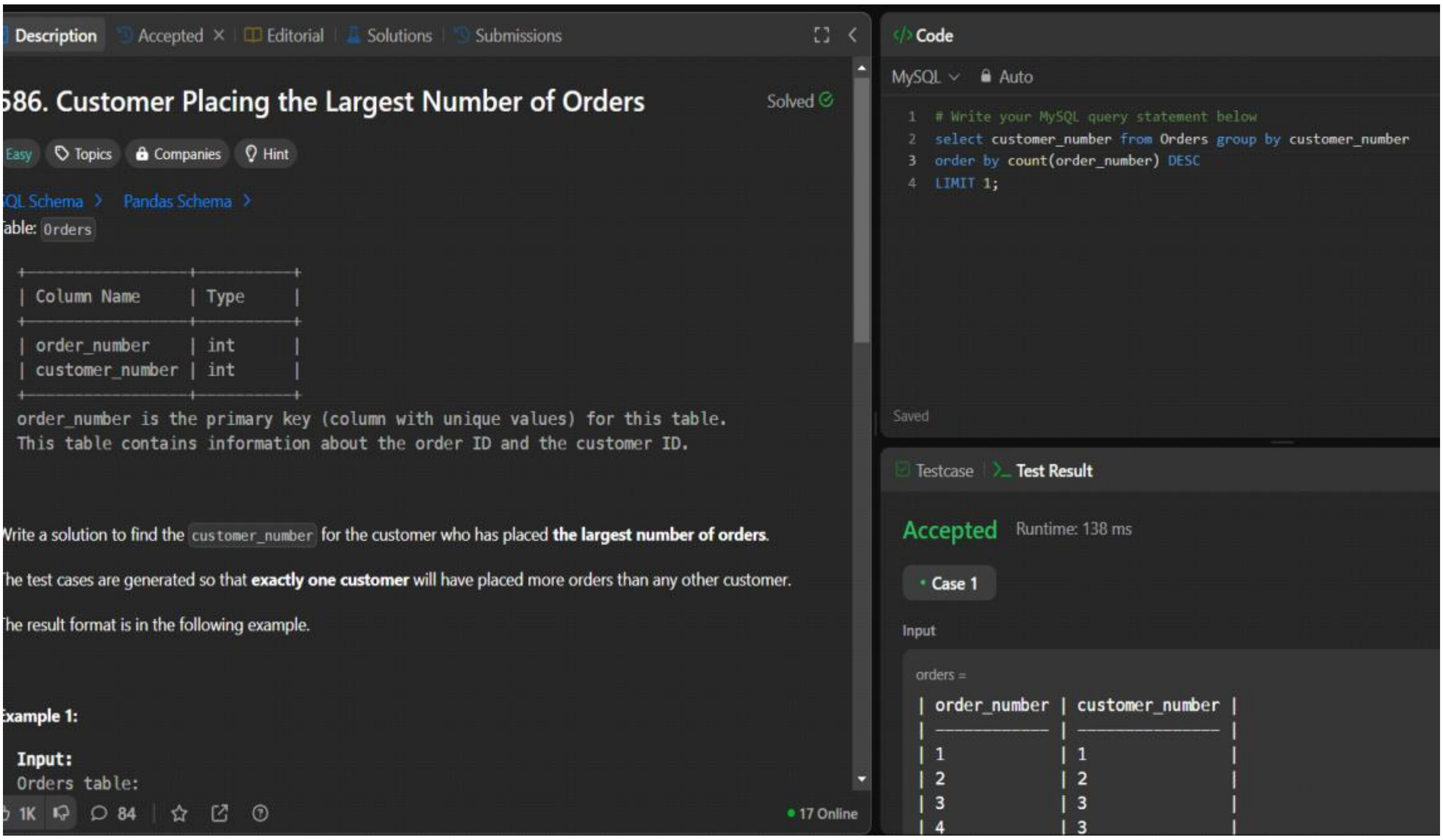
Table: `Orders`

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| customerId  | int  |
+-------------+------+
id is the primary key (column with unique values) for this table.
customerId is a foreign key (reference columns) 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
```

```sql
# Write your MySQL query statement below
select c.name as Customers from customers c
left join orders o on c.id=o.customerId where o.id is null;
```

Customers =

id	name
1	Joe
2	Henry
3	Sam
4	Max



586. Customer Placing the Largest Number of Orders

QL Schema > Pandas Schema >

able: `Orders`

```
+-----------------+----------+
| Column Name     | Type     |
+-----------------+----------+
| order_number    | int      |
| customer_number | int      |
+-----------------+----------+
order_number is the primary key (column with unique values) for this table.
This table contains information about the order ID and the customer ID.
```

Vrite a solution to find the `customer_number` for the customer who has placed **the largest number of orders**.

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

he result format is in the following example.

xample 1:

```
Input:
Orders table:
```

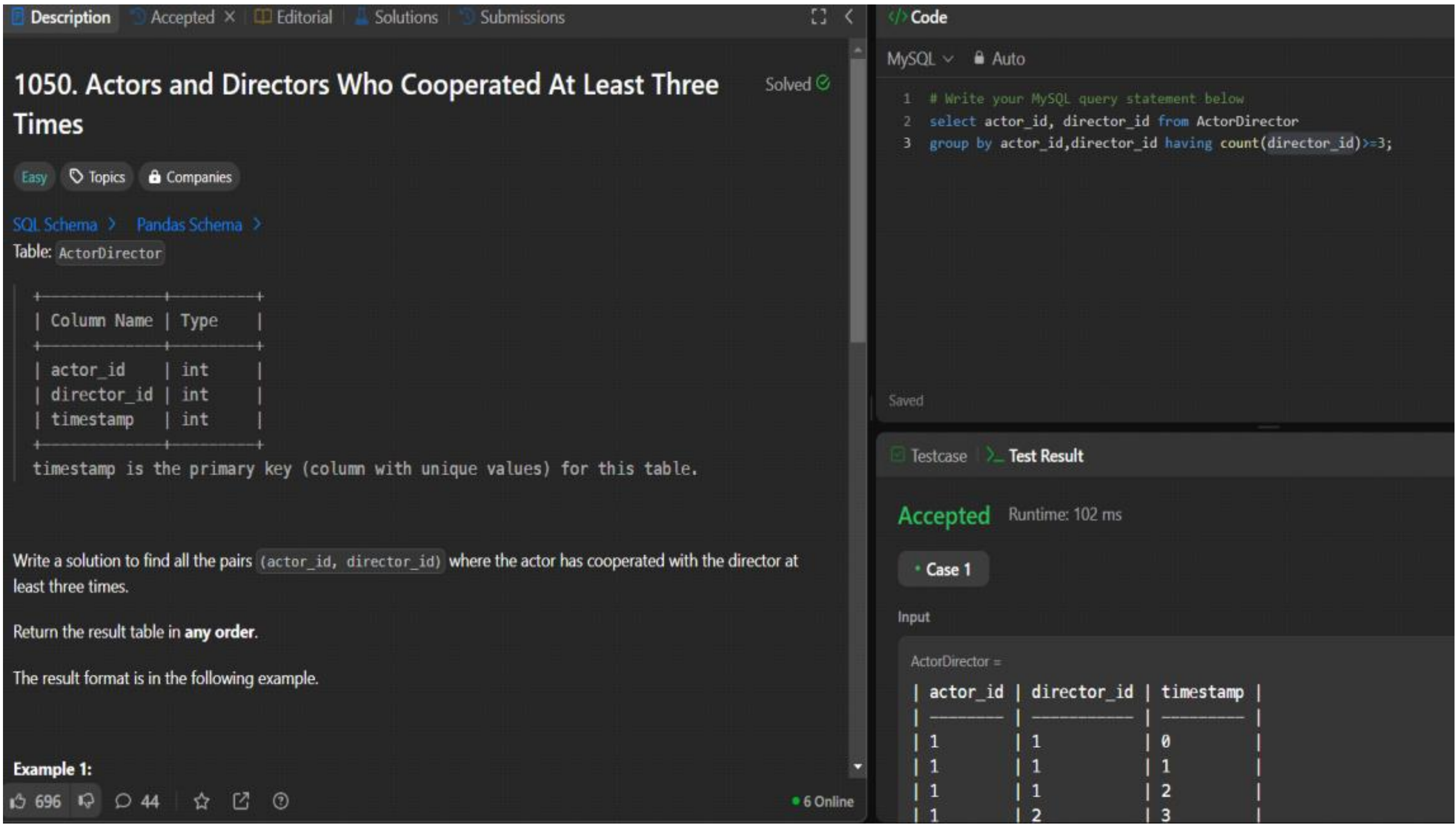
```
# Write your MySQL query statement below
select customer_number from Orders group by customer_number
order by count(order_number) DESC
LIMIT 1;
```

Accepted Runtime: 138 ms

Case 1

Input

```
orders =
| order_number | customer_number |
| ------------ | --------------- |
| 1            | 1               |
| 2            | 2               |
| 3            | 3               |
| 4            | 3               |
```



1050. Actors and Directors Who Cooperated At Least Three Times

SQL Schema > Pandas Schema >

Table: `ActorDirector`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| actor_id    | int     |
| director_id | int     |
| timestamp   | int     |
+-------------+---------+
timestamp is the primary key (column with unique values) for this table.
```

Write a solution to find all 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 result format is in the following example.

Example 1:

```
# Write your MySQL query statement below
select actor_id, director_id from ActorDirector
group by actor_id,director_id having count(director_id)>=3;
```

Accepted Runtime: 102 ms

Case 1

Input

```
ActorDirector =
| actor_id | director_id | timestamp |
| -------- | ----------- | --------- |
| 1        | 1           | 0         |
| 1        | 1           | 1         |
| 1        | 1           | 2         |
| 1        | 2           | 3         |
```

GitD1

17 February 2025 01:19

Git is a distributed source control system that is widely used in the software development industry. Git allows us to store copies of our source code on multiple computer systems throughout our network and across the world. We can have multiple local Git repositories on our own machine and also have them sync with repositories on other systems to keep each of them up to date.

Git Commits

A commit in a git repository records a snapshot of all the (tracked) files in your directory. It's like a giant copy and paste, but even better!
Git wants to keep commits as lightweight as possible though, so it doesn't just blindly copy the entire directory every time you commit. It can (when possible) compress a commit as a set of changes, or a "delta", from one version of the repository to the next.

git init

Initializes a new Git repository in the current directory.
Example: git init creates a .git folder to track changes.

git clone <repository-url>

Creates a copy of a remote repository on your local machine.
Example: git clone https://github.com/user/repo.git

git add <file>

Stages changes for the next commit.
Example: git add file.txt stages file.txt. Use git add . to stage all changes.

git commit -m "commit message"

Saves staged changes to the repository with a message.
Example: git commit -m "Added new feature"

git status

Shows the status of the working directory (staged, unstaged, and untracked files).
Example: git status

git log

Displays the commit history of the repository.
Example: git log shows all commits. Use git log --oneline for a compact view.

git diff

Shows differences between the working directory and the staging area.
Example: git diff shows unstaged changes. Use git diff --staged for staged changes.

git pull

Fetches changes from a remote repository and merges them into the current branch.
Example: git pull origin main

git push

Uploads local commits to a remote repository.
Example: git push origin main

git remote -v

Lists all remote repositories connected to the local repository.
Example: git remote -v

git fetch

Downloads changes from a remote repository but does not merge them.
Example: git fetch origin

git rm <file>

Removes a file from the working directory and stages the deletion.
Example: git rm file.txt

git mv <old-file> <new-file>

Renames a file and stages the change.
Example: git mv old.txt new.txt

git reset <file>

Unstages a file while keeping changes in the working directory.
Example: git reset file.txt

git checkout <file>

Discards changes in the working directory for a specific file.
Example: git checkout file.txt

Branching Commands

git branch

Lists all local branches. The current branch is highlighted with an asterisk (*).
Example: git branch

git branch <branch-name>

Creates a new branch.
Example: git branch feature-branch

git checkout <branch-name>

Switches to the specified branch.
Example: git checkout feature-branch

git checkout -b <branch-name>

Creates a new branch and switches to it.
Example: git checkout -b feature-branch

git merge <branch-name>

Merges the specified branch into the current branch.
Example: git merge feature-branch

git branch -d <branch-name>

Deletes the specified branch.
Example: git branch -d feature-branch

git branch -m <new-branch-name>

Renames the current branch.
Example: git branch -m new-branch-name

git push origin <branch-name>

Pushes a local branch to the remote repository.
Example: git push origin feature-branch

git push --delete origin <branch-name>

Deletes a remote branch.
Example: git push --delete origin feature-branch

Stashing Commands

git stash

Temporarily saves changes in the working directory that are not ready to be committed.
Example: git stash

git stash list

Lists all stashed changes.
Example: git stash list

git stash apply

Applies the most recent stashed changes to the working directory.
Example: git stash apply

git stash apply stash@{n}

Applies a specific stash from the stash list.
Example: git stash apply stash@{1}

git stash pop

Applies the most recent stash and removes it from the stash list.
Example: git stash pop

git stash drop stash@{n}

Deletes a specific stash from the stash list.
Example: git stash drop stash@{1}

git stash clear

Deletes all stashed changes.
Example: git stash clear

git stash branch <branch-name>

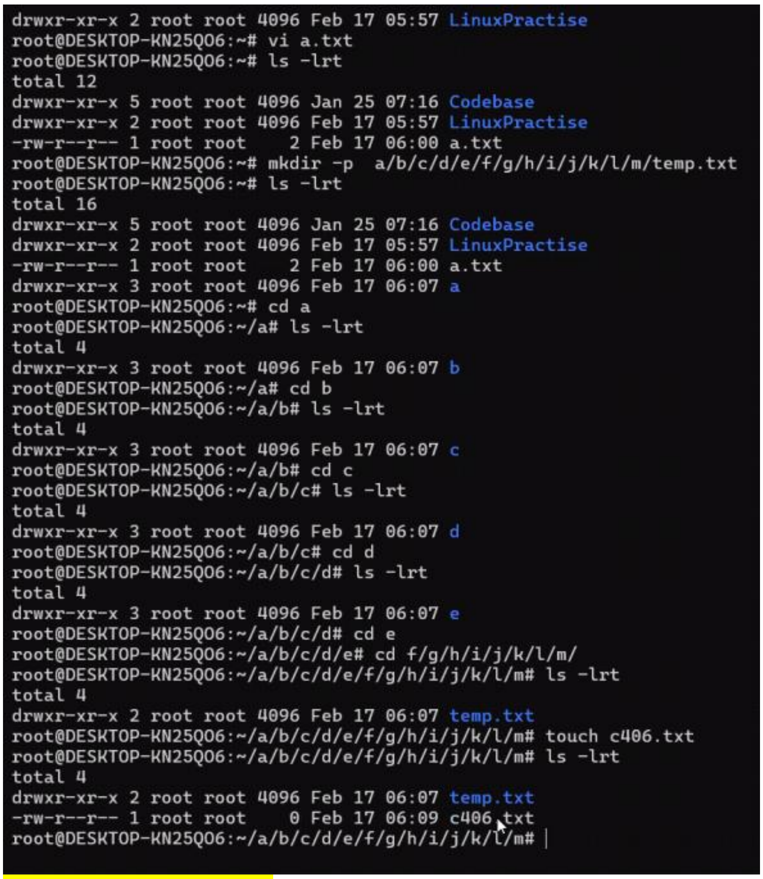
Creates a new branch from the stashed changes.
Example: git stash branch new-branch

Git Reset

If two people wants to make changes in a git file
Then the first person can easily push it but the second person have to first do "git reset" then push the code/content.

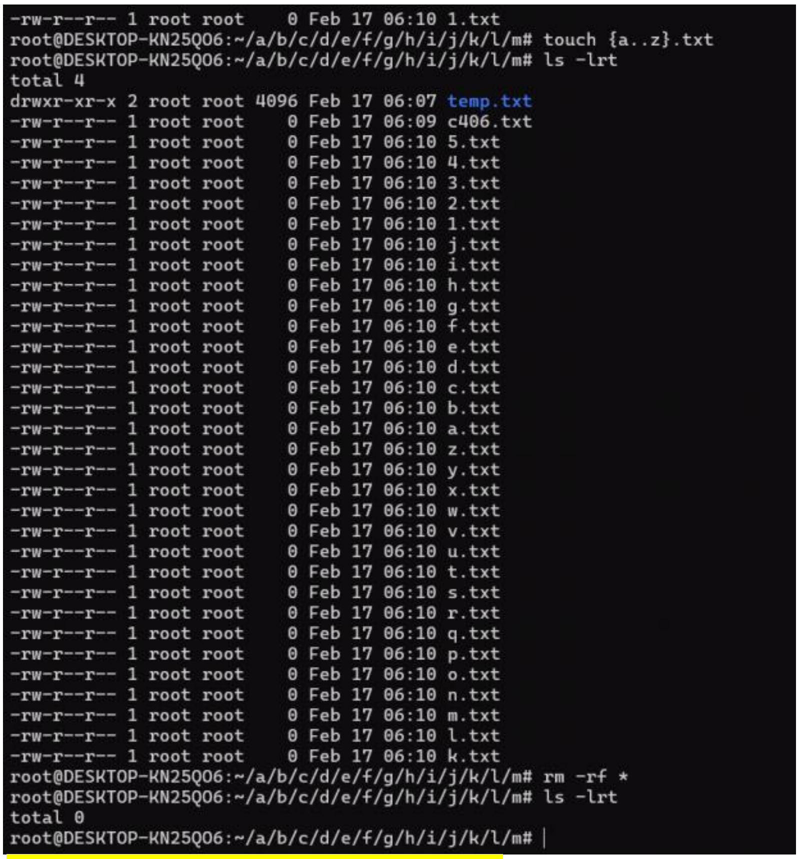
D1

17 February 2025 11:41



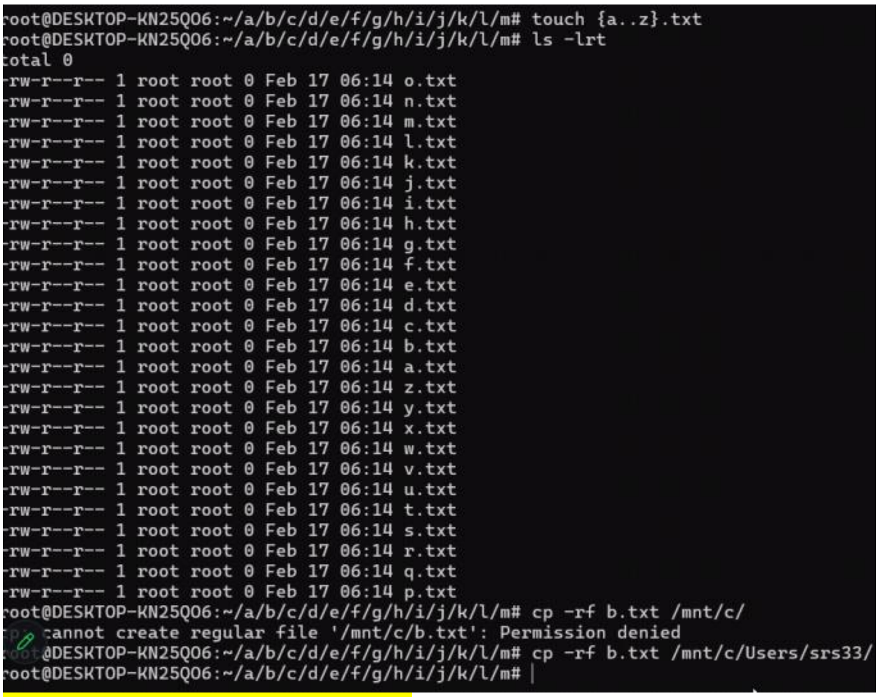
```
drwxr-xr-x 2 root root 4096 Feb 17 05:57 LinuxPractise
root@DESKTOP-KN25Q06:~# vi a.txt
root@DESKTOP-KN25Q06:~# ls -lrt
total 12
drwxr-xr-x 5 root root 4096 Jan 25 07:16 Codebase
drwxr-xr-x 2 root root 4096 Feb 17 05:57 LinuxPractise
-rw-r--r-- 1 root root    2 Feb 17 06:00 a.txt
root@DESKTOP-KN25Q06:~# mkdir -p  a/b/c/d/e/f/g/h/i/j/k/l/m/temp.txt
root@DESKTOP-KN25Q06:~# ls -lrt
total 16
drwxr-xr-x 5 root root 4096 Jan 25 07:16 Codebase
drwxr-xr-x 2 root root 4096 Feb 17 05:57 LinuxPractise
-rw-r--r-- 1 root root    2 Feb 17 06:00 a.txt
drwxr-xr-x 3 root root 4096 Feb 17 06:07 a
root@DESKTOP-KN25Q06:~# cd a
root@DESKTOP-KN25Q06:~/a# ls -lrt
total 4
drwxr-xr-x 3 root root 4096 Feb 17 06:07 b
root@DESKTOP-KN25Q06:~/a# cd b
root@DESKTOP-KN25Q06:~/a/b# ls -lrt
total 4
drwxr-xr-x 3 root root 4096 Feb 17 06:07 c
root@DESKTOP-KN25Q06:~/a/b# cd c
root@DESKTOP-KN25Q06:~/a/b/c# ls -lrt
total 4
drwxr-xr-x 3 root root 4096 Feb 17 06:07 d
root@DESKTOP-KN25Q06:~/a/b/c# cd d
root@DESKTOP-KN25Q06:~/a/b/c/d# ls -lrt
total 4
drwxr-xr-x 3 root root 4096 Feb 17 06:07 e
root@DESKTOP-KN25Q06:~/a/b/c/d# cd e
root@DESKTOP-KN25Q06:~/a/b/c/d/e# cd f/g/h/i/j/k/l/m/
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# ls -lrt
total 4
drwxr-xr-x 2 root root 4096 Feb 17 06:07 temp.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# touch c406.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# ls -lrt
total 4
drwxr-xr-x 2 root root 4096 Feb 17 06:07 temp.txt
-rw-r--r-- 1 root root    0 Feb 17 06:09 c406.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m#
```

To create files in linux space



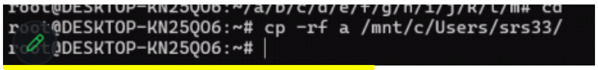
```
-rw-r--r-- 1 root root    0 Feb 17 06:10 1.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# touch {a..z}.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# ls -lrt
total 4
drwxr-xr-x 2 root root 4096 Feb 17 06:07 temp.txt
-rw-r--r-- 1 root root    0 Feb 17 06:09 c406.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 5.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 4.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 3.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 2.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 1.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 j.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 i.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 h.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 g.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 f.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 e.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 d.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 c.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 b.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 a.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 z.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 y.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 x.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 w.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 v.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 u.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 t.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 s.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 r.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 q.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 p.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 o.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 n.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 m.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 l.txt
-rw-r--r-- 1 root root    0 Feb 17 06:10 k.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# rm -rf *
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# ls -lrt
total 0
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m#
```

rf* recursive force it delete all the files you created



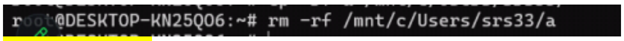
```
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# touch {a..z}.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# ls -lrt
total 0
-rw-r--r-- 1 root root 0 Feb 17 06:14 o.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 n.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 m.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 l.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 k.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 j.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 i.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 h.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 g.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 f.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 e.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 d.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 c.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 b.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 a.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 z.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 y.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 x.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 w.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 v.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 u.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 t.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 s.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 r.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 q.txt
-rw-r--r-- 1 root root 0 Feb 17 06:14 p.txt
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# cp -rf b.txt /mnt/c/
cp: cannot create regular file '/mnt/c/b.txt': Permission denied
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# cp -rf b.txt /mnt/c/Users/srs33/
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# |
```

COPY file from one place to another place.



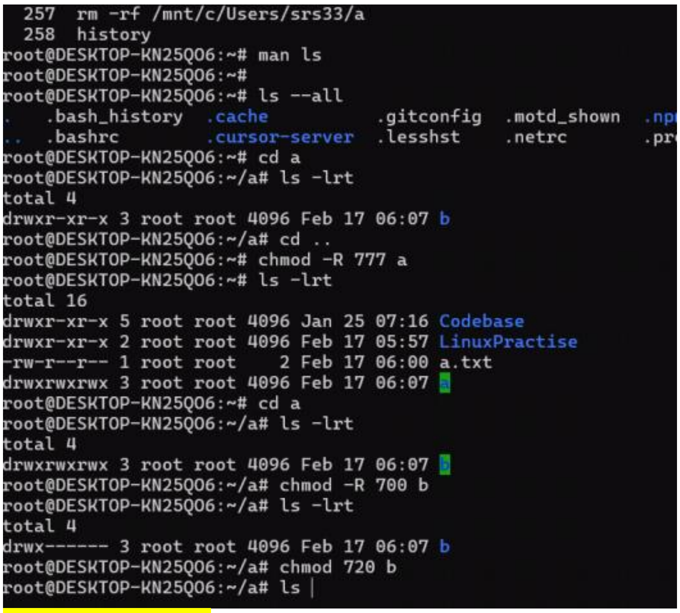
```
root@DESKTOP-KN25Q06:~/a/b/c/d/e/f/g/h/i/j/k/l/m# cd
root@DESKTOP-KN25Q06:~# cp -rf a /mnt/c/Users/srs33/
root@DESKTOP-KN25Q06:~# |
```

To create recursive files in users folder



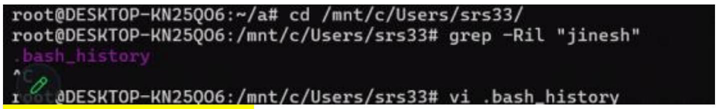
```
root@DESKTOP-KN25Q06:~# rm -rf /mnt/c/Users/srs33/a
```

To Delete a File



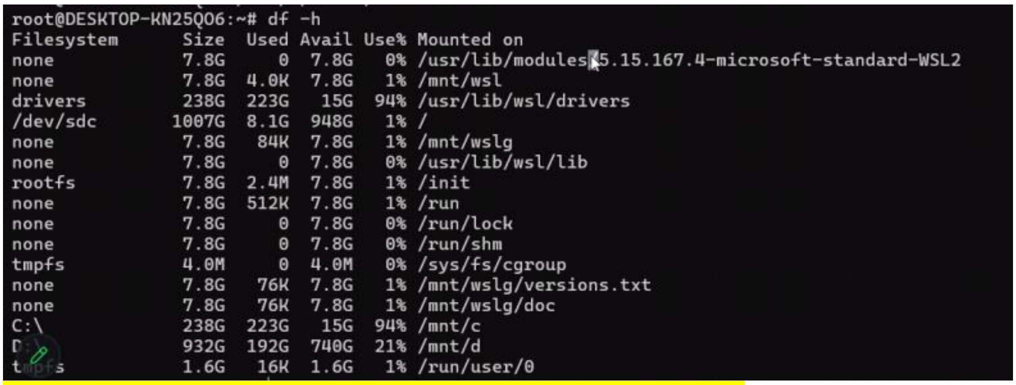
```
  257  rm -rf /mnt/c/Users/srs33/a
  258  history
root@DESKTOP-KN25Q06:~# man ls
root@DESKTOP-KN25Q06:~#
root@DESKTOP-KN25Q06:~# ls --all
.    .bash_history  .cache         .gitconfig  .motd_shown  .np
..   .bashrc        .cursor-server .lesshst    .netrc       .pr
root@DESKTOP-KN25Q06:~# cd a
root@DESKTOP-KN25Q06:~/a# ls -lrt
total 4
drwxr-xr-x 3 root root 4096 Feb 17 06:07 b
root@DESKTOP-KN25Q06:~/a# cd ..
root@DESKTOP-KN25Q06:~# chmod -R 777 a
root@DESKTOP-KN25Q06:~# ls -lrt
total 16
drwxr-xr-x 5 root root 4096 Jan 25 07:16 Codebase
drwxr-xr-x 2 root root 4096 Feb 17 05:57 LinuxPractise
-rw-r--r-- 1 root root    2 Feb 17 06:00 a.txt
drwxrwxrwx 3 root root 4096 Feb 17 06:07 a
root@DESKTOP-KN25Q06:~# cd a
root@DESKTOP-KN25Q06:~/a# ls -lrt
total 4
drwxrwxrwx 3 root root 4096 Feb 17 06:07 b
root@DESKTOP-KN25Q06:~/a# chmod -R 700 b
root@DESKTOP-KN25Q06:~/a# ls -lrt
total 4
drwx------ 3 root root 4096 Feb 17 06:07 b
root@DESKTOP-KN25Q06:~/a# chmod 720 b
root@DESKTOP-KN25Q06:~/a# ls |
```

#CHMOD COMMAND



```
root@DESKTOP-KN25Q06:~/a# cd /mnt/c/Users/srs33/
root@DESKTOP-KN25Q06:/mnt/c/Users/srs33# grep -Ril "jinesh"
.bash_history
^C
root@DESKTOP-KN25Q06:/mnt/c/Users/srs33# vi .bash_history
```

TO SEARCH IN YOUR FILE '

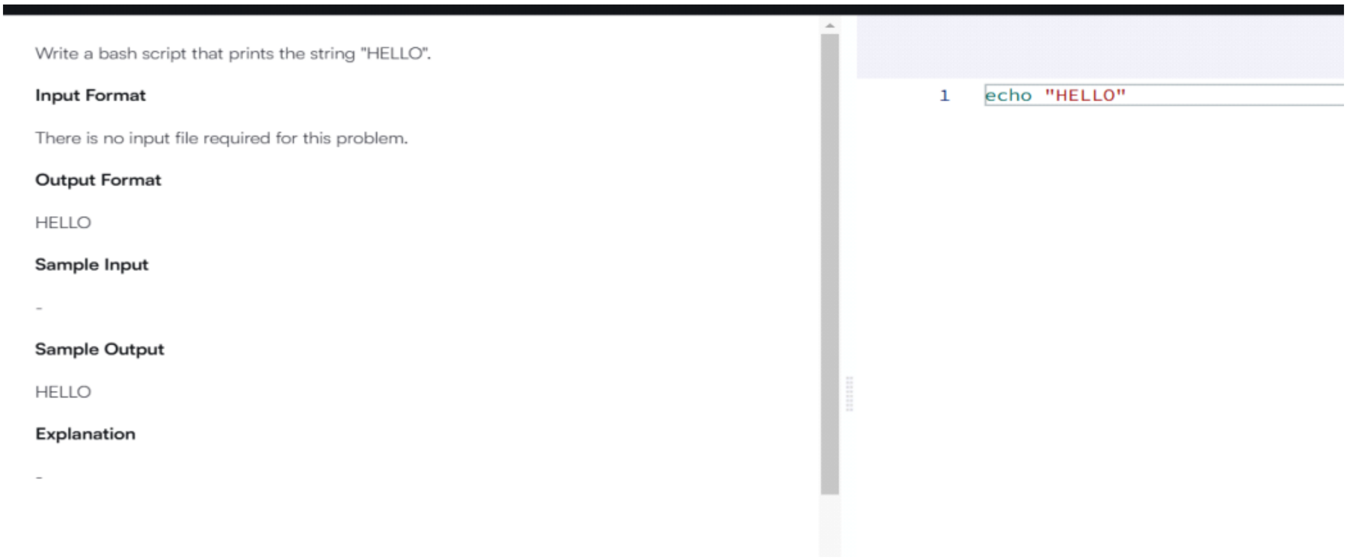


```
root@DESKTOP-KN25Q06:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
none            7.8G     0  7.8G   0% /usr/lib/modules/5.15.167.4-microsoft-standard-WSL2
none            7.8G  4.0K  7.8G   1% /mnt/wsl
drivers         238G  223G   15G  94% /usr/lib/wsl/drivers
/dev/sdc       1007G  8.1G  948G   1% /
none            7.8G   84K  7.8G   1% /mnt/wslg
none            7.8G     0  7.8G   0% /usr/lib/wsl/lib
rootfs          7.8G  2.4M  7.8G   1% /init
none            7.8G  512K  7.8G   1% /run
none            7.8G     0  7.8G   0% /run/lock
none            7.8G     0  7.8G   0% /run/shm
tmpfs           4.0M     0  4.0M   0% /sys/fs/cgroup
none            7.8G   76K  7.8G   1% /mnt/wslg/versions.txt
none            7.8G   76K  7.8G   1% /mnt/wslg/doc
C:\             238G  223G   15G  94% /mnt/c
D:\             932G  192G  740G  21% /mnt/d
tmpfs           1.6G   16K  1.6G   1% /run/user/0
```

TO CHECK HOW MUCH STORAGE YOU HAVE USED AND WHAT IS REMAINING

NOTE:- Man GREP IS COMMAND USED TO SEARCH

Pwd is used to know where we are in the file currently
Print with directory



Write a bash script that prints the string "HELLO".

Input Format

There is no input file required for this problem.

Output Format

HELLO

Sample Input

-

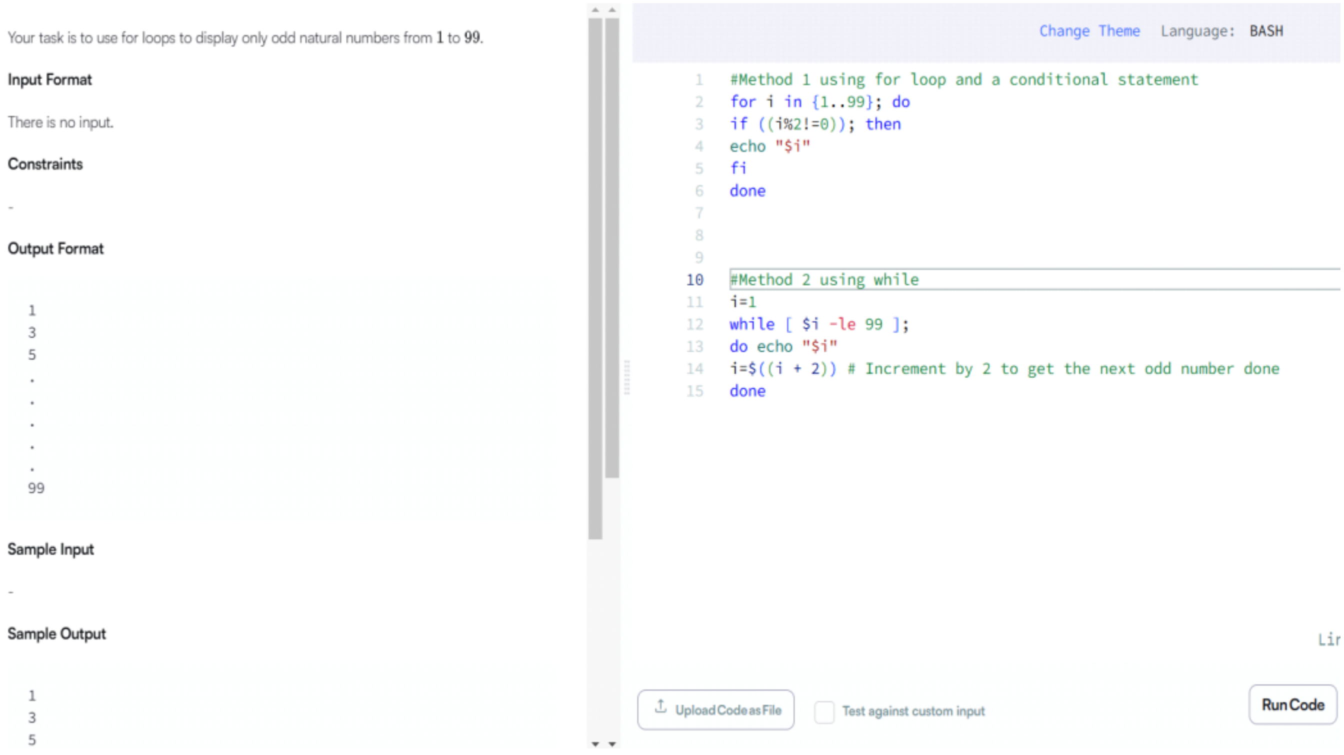
Sample Output

HELLO

Explanation

-

```
echo "HELLO"
```



Your task is to use for loops to display only odd natural numbers from 1 to 99.

Input Format

There is no input.

Constraints

-

Output Format

```
1
3
5
.
.
.
.
.
99
```

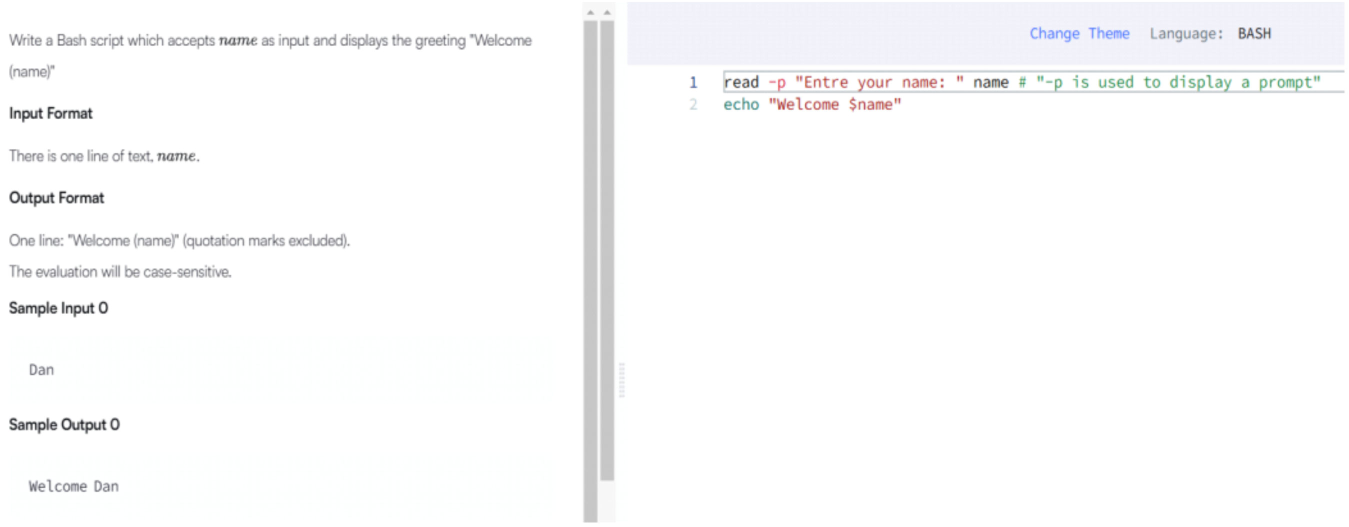
Sample Input

-

Sample Output

```
1
3
5
```

```bash
#Method 1 using for loop and a conditional statement
for i in {1..99}; do
if ((i%2!=0)); then
echo "$i"
fi
done



#Method 2 using while
i=1
while [ $i -le 99 ];
do echo "$i"
i=$((i + 2)) # Increment by 2 to get the next odd number done
done
```



Write a Bash script which accepts *name* as input and displays the greeting "Welcome (name)"

Input Format

There is one line of text, *name*.

Output Format

One line: "Welcome (name)" (quotation marks excluded).

The evaluation will be case-sensitive.

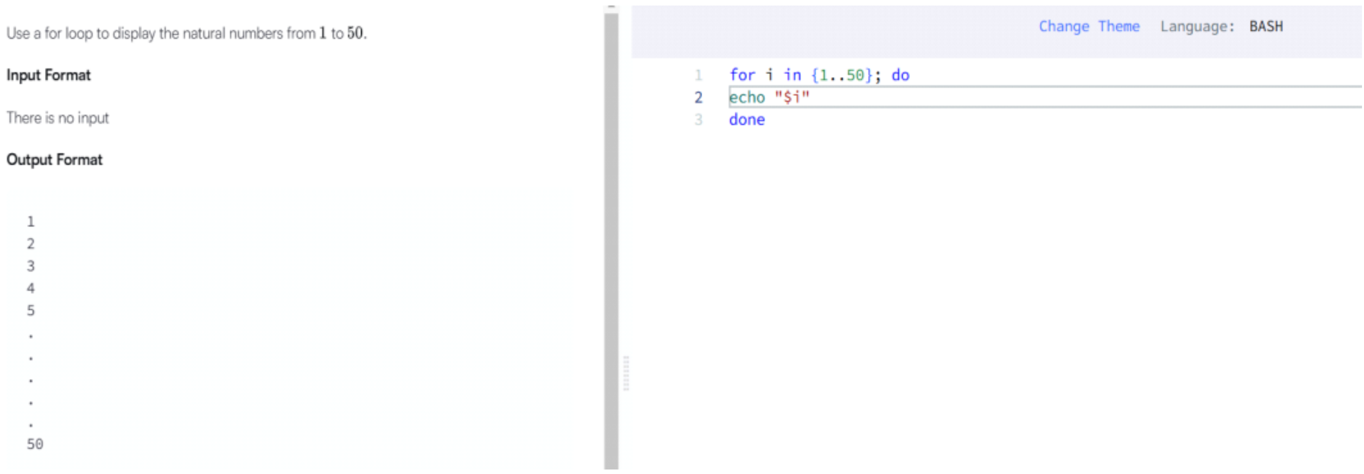
Sample Input 0

```
Dan
```

Sample Output 0

```
Welcome Dan
```

```bash
read -p "Entre your name: " name # "-p is used to display a prompt"
echo "Welcome $name"
```



Use a for loop to display the natural numbers from 1 to 50.

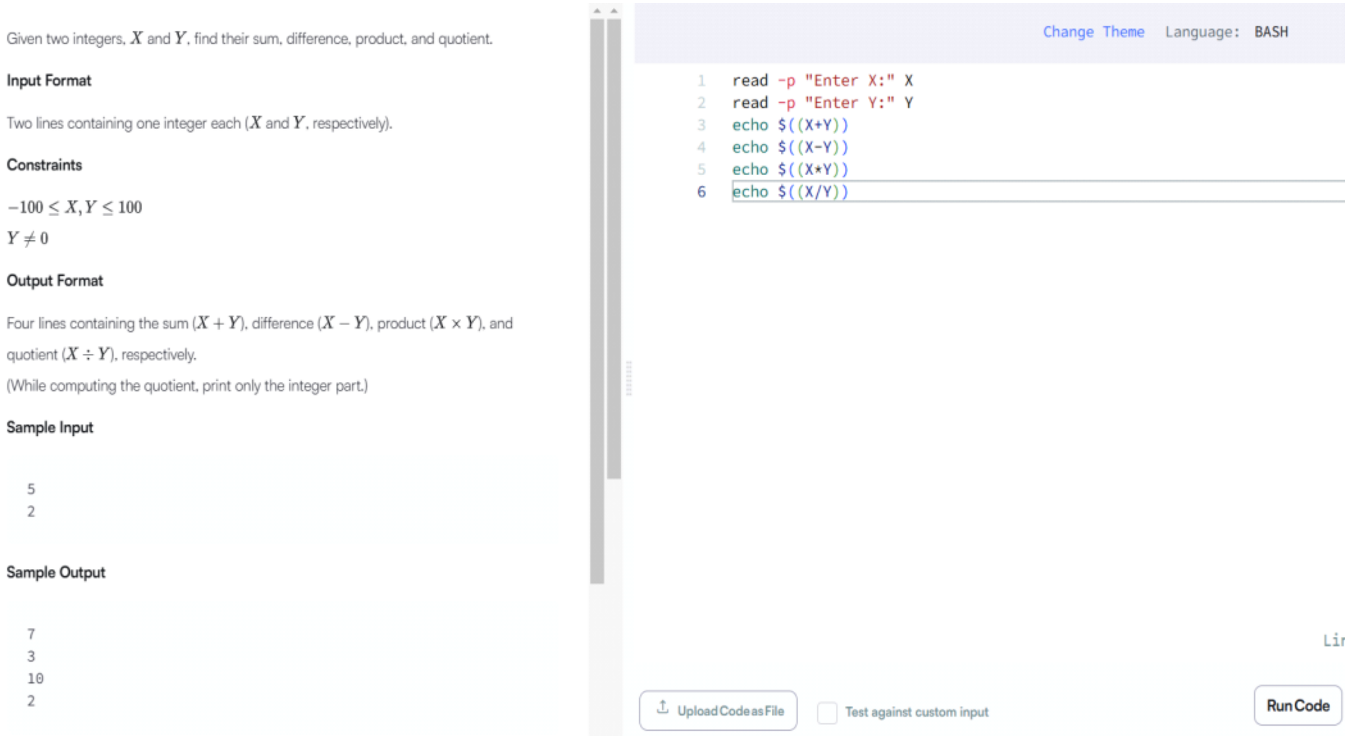
Input Format

There is no input

Output Format

```
1
2
3
4
5
.
.
.
.
.
50
```

```bash
for i in {1..50}; do
echo "$i"
done
```



Given two integers, X and Y, find their sum, difference, product, and quotient.

Input Format

Two lines containing one integer each (X and Y, respectively).

Constraints

$-100 \leq X, Y \leq 100$

$Y \neq 0$

Output Format

Four lines containing the sum ($X + Y$), difference ($X - Y$), product ($X \times Y$), and quotient ($X \div Y$), respectively.

(While computing the quotient, print only the integer part.)

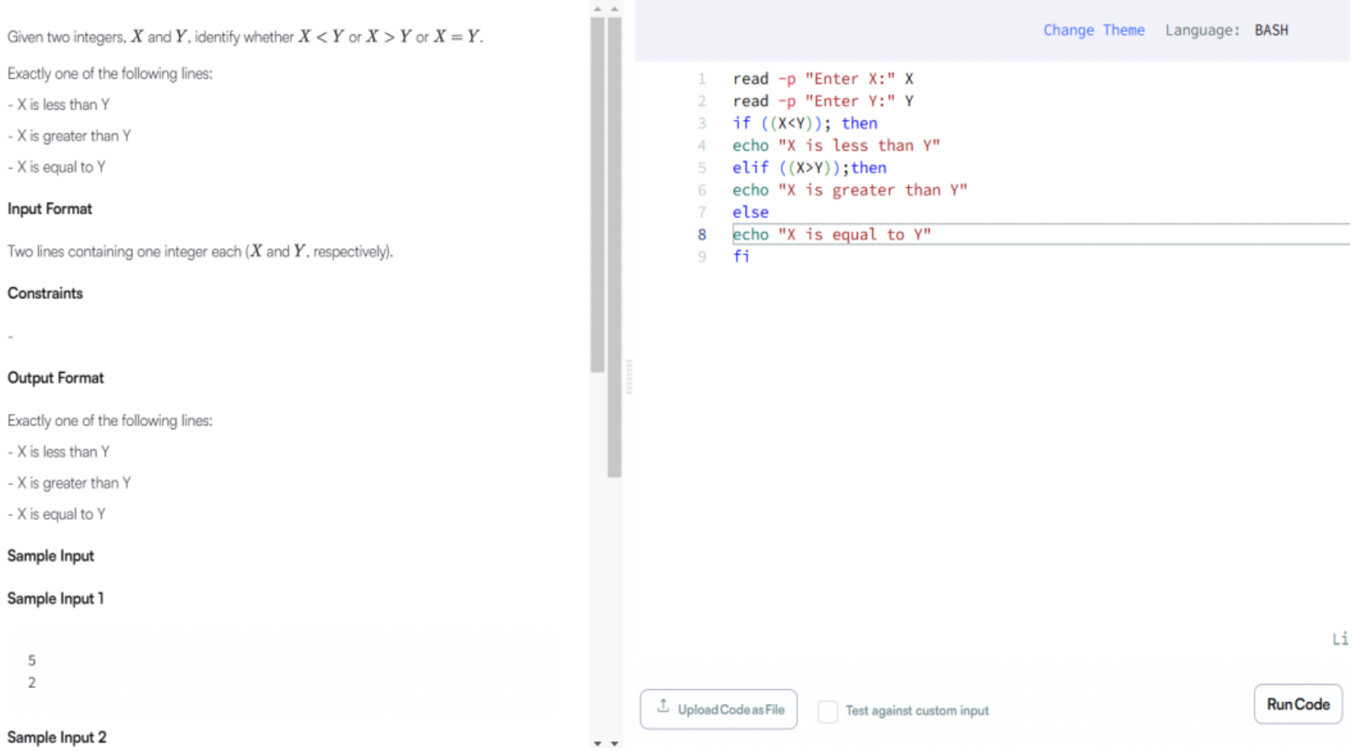
Sample Input

```
5
2
```

Sample Output

```
7
3
10
2
```

```bash
read -p "Enter X:" X
read -p "Enter Y:" Y
echo $((X+Y))
echo $((X-Y))
echo $((X*Y))
echo $((X/Y))
```



Given two integers, X and Y, identify whether $X < Y$ or $X > Y$ or $X = Y$.

Exactly one of the following lines:

- X is less than Y

- X is greater than Y

- X is equal to Y

Input Format

Two lines containing one integer each (X and Y, respectively).

Constraints

-

Output Format

Exactly one of the following lines:

- X is less than Y

- X is greater than Y

- X is equal to Y

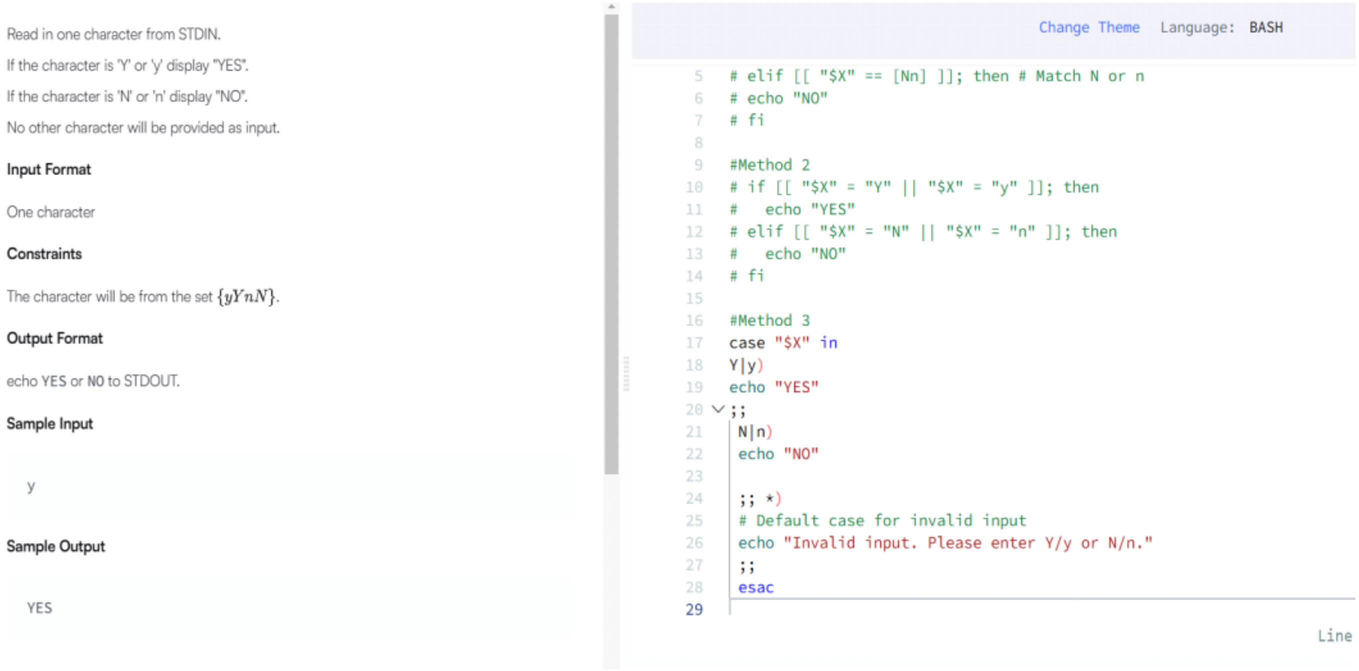
Sample Input

Sample Input 1

```
5
2
```

Sample Input 2

```bash
read -p "Enter X:" X
read -p "Enter Y:" Y
if ((X<Y)); then
echo "X is less than Y"
elif ((X>Y));then
echo "X is greater than Y"
else
echo "X is equal to Y"
fi
```



Read in one character from STDIN.

If the character is 'Y' or 'y' display "YES".

If the character is 'N' or 'n' display "NO".

No other character will be provided as input.

Input Format

One character

Constraints

The character will be from the set $\{yYnN\}$.

Output Format

echo `YES` or `NO` to STDOUT.

Sample Input

```
y
```

Sample Output

```
YES
```

Language: BASH

```bash
# elif [[ "$X" == [Nn] ]]; then # Match N or n
# echo "NO"
# fi

#Method 2
# if [[ "$X" = "Y" || "$X" = "y" ]]; then
#   echo "YES"
# elif [[ "$X" = "N" || "$X" = "n" ]]; then
#   echo "NO"
# fi

#Method 3
case "$X" in
Y|y)
echo "YES"
;;
 N|n)
 echo "NO"

 ;; *)
 # Default case for invalid input
 echo "Invalid input. Please enter Y/y or N/n."
 ;;
 esac

```

D2

Tuesday, February 18, 2025 10:41 PM

File and Directory Management

1. `pwd` - Print the current working directory.
2. `ls` - List files and directories.
    - `ls -l` - List with detailed information.
    - `ls -a` - List all files, including hidden ones.
3. `cd <dir>` - Change directory.
    - `cd ..` - Move up one directory.
4. `mkdir <dir>` - Create a new directory.
5. `rmdir <dir>` - Remove an empty directory.
6. `rm <file>` - Remove a file.
    - `rm -r <dir>` - Remove a directory and its contents.
    - `rm -f <file>` - Force remove a file without confirmation
7. `cp <source> <destination>` - Copy a file or directory.
    - `cp -r <source> <destination>` - Copy directories recursively.
8. `mv <source> <destination>` - Move or rename files or directories.
9. `touch <file>` - Create an empty file or update the timestamp of an existing file.
10. `cat <file>` - View the content of a file.
11. `more <file>` - View content of a file, page by page..
12. `less <file>` - View content of a file, with scrolling ability
13. `head <file>` - View the first 10 lines of a file
14. `tail <file>` - View the last 10 lines of a file

- `tail -f <file>` - View the end of a file and monitor for changes

File Permissions

1. `chmod <permissions> <file>` - Change file permissions.
    - Example: `chmod 755 <file>`
2. `chown <user>:<group> <file>` - Change the owner and group of a file
3. `chgrp <group> <file>` - Change the group of a file

Process Management

1. `ps` - List running processes
    - `ps aux` - Display all running processes

When you run ps aux you will see the following on o\p:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  16980  1128 ?        Ss   Feb12   0:03 /sbin/init (it's a process)
root         2  0.0  0.0      0     0 ?        S    Feb12   0:00 [kthreadd]
user      1201  0.3  1.2  48584 12936 tty1     Ss+  09:45   0:14 bash
user      1345  0.0  0.5  21520  4564 tty1     S+   09:47   0:00 vim content.txt
root      9999  0.0  0.1  45678  1892 ?        S    10:00   0:00 apache2
```

1. `top` - Show active processes in real time.
2. `kill <pid>` - Terminate a process by its PID.
    - `kill -9 <pid>` - Force kill a process.
3. `bg` - Resume a suspended process in the background.
4. `fg` - Bring a background process to the foreground.

5. `htop` - Interactive process viewer (if installed).

System Information

1. `uname -r` - Show the kernel version.
2. `hostname` - Display the system's hostname.
3. `uptime` - Show how long the system has been running.
4. `df` - Display disk space usage.
   - `df -h` - Display disk space in human-readable format.
5. `free` - Show memory usage.
   - `free -h` - Display memory usage in human-readable format.
6. `top` - Display real-time process information.
7. `dmesg` - Display system log messages.
8. `lscpu` - Display CPU architecture information.
9. `lsblk` - List information about block devices (hard drives, SSDs, etc.).

Networking

1. `ping <host>` - Ping a host to check connectivity.
2. `ifconfig` - Show or configure network interfaces (older versions).
3. `ip a` - Display network interfaces and addresses (newer versions).
4. `netstat` - Display network connections, routing tables, and interface statistics.
   - `netstat -tuln` - Show listening ports.
5. `traceroute <host>` - Trace the route packets take to reach a host.
6. `curl <url>` - Fetch data from a URL.
7. `wget <url>` - Download files from the web.

File Compression

1. `tar -czvf <file.tar.gz> <directory>` - Compress a directory into a `.tar.gz` archive.
2. `tar -xzvf <file.tar.gz>` - Extract a `.tar.gz` archive.
3. `zip <file.zip> <file>` - Create a ZIP archive.
4. `unzip <file.zip>` - Extract a ZIP archive.

User Management

1. `whoami` - Display the current logged-in user.
2. `useradd <username>` - Create a new user.
3. `usermod` - Modify user information.
   - Example: `usermod -aG <group> <username>` to add a user to a group.
4. `passwd <username>` - Change the password for a user.
5. `groupadd <groupname>` - Create a new group.
6. `groups <username>` - Display the groups the user belongs to.
7. `id <username>` - Display user and group information.

Package Management (Debian-based, e.g., Ubuntu)

1. `apt update` - Update the package list.
2. `apt upgrade` - Upgrade installed packages.
3. `apt install <package>` - Install a package.
4. `apt remove <package>` - Remove a package.
5. `apt purge <package>` - Remove a package and its configuration files.
6. `apt search <package>` - Search for a package.

Archiving & System Backup

1. `rsync -av <source> <destination>` - Synchronize files and directories.
2. `tar -czvf <archive.tar.gz> <folder>` - Archive and compress a folder.
3. `scp <source> <destination>` - Securely copy files between machines over SSH.

Others

1. `man <command>` - Display the manual page for a command.
2. `history` - Display the command history.
3. `clear` - Clear the terminal screen.
4. `alias <name>='<command>'` - Create an alias for a command.
5. `echo <text>` - Display a line of text.

1. **`vi content.txt`** :
    - **`vi`** is a text editor in Linux (and other Unix-like systems).
    - This command opens the file **`content.txt`** in the `vi` editor. If the file doesn't exist, `vi` will create it when you save.
    - To edit the file, press `i` to go into insert mode, make your changes, and then press `Esc` to exit insert mode. To save the file, type `:w` and press `Enter`. To exit `vi`, type `:q` and press `Enter`. If you want to do both (save and exit), type `:wq`.
    1. **`cat content.txt`**:
        - **`cat`** is a command used to display the contents of a file.
        - This command will show the contents of **`content.txt`** in the terminal. If the file is long, it will scroll by quickly; for longer files, you might prefer to use `more` or `less` for easier reading.
        1. **`ps aux`** :
            - **`ps`** is a command that shows the current running processes on the system.
            - The **aux** flags are options that provide more detailed information:
                - **a**: Shows processes for all users, not just the current user.
                - **u**: Shows the process owner (username) and other details.
                - **x**: Includes processes not associated with a terminal (background processes).
            - This command will output a list of all processes running on the system, including their process IDs (PID), CPU usage, memory usage, and more.

COMMANDS

Vi name.text
Sort -r name.text(sort alphabet in reverse order)
Sort -n name.text(sort numbers)
Sudo apt install ncal
Sudo apt install plocate
sudo apt install update

`plocate`: This is the name of the package to be installed. **`plocate`** is a tool used to **locate files** on your system, providing a faster, more memory-efficient alternative to the older `locate` command. It's a utility that helps you quickly find files by searching through a database of file paths.

All the things that are system default and you want to see their location you can use the command :   whereis java   (eg java)

Examples
Note to access :-
ncal 12 2025

cal 2025

cal 1990

Note:-
wc -l name.txt (count line)
wc -w name.txt(count word)
wc -c name.txt(count character)
wc*

`grep -Ril "saumtt"`:

- **`grep`**: This command is used for searching through files or input based on a pattern.
- **`-R`**: This option stands for **recursive**, meaning `grep` will search through files in all subdirectories of the current directory.
- **`-i`**: This option makes the search **case-insensitive**, meaning it will match **`"saumtt"`**, `"saumtt"`, `"SAUMTT"`, etc.
- **`-l`**: This option tells `grep` to only **list the filenames** where the pattern was found, instead of displaying the matching lines themselves.
- **`"saumtt"`**: This is the pattern that `grep` will search for in the files.

What this command does: It searches for files containing the word `"saumtt"` (case-insensitive) in the current directory and all its subdirectories, and it lists the names of those files.

2. `grep -Ril "saumtt" name.txt`:

This command is similar to the previous one, but here:

- It is **searching in the file `name.txt`** (instead of all files recursively in the directory).

What this command does: It searches for the case-insensitive occurrence of `"saumtt"` inside the file `name.txt` and lists the filename if a match is found. Since it's a single file (`name.txt`), it doesn't need recursion.

Other imp cmad

- **`history | grep git`**: Filters the command history to show only the commands that contain the word `"git"`.
- **`git pull`**: Pulls the latest changes from the remote repository and merges them with the current local branch.
- **`git reset --hard origin/main`**: Resets the current branch to exactly match the remote `main` branch, discarding any local changes.

cd /mnt/c/Users/srs33/

find -name "*.txt"

find . -type d

```
find . -name "*.txt"
```

This will search for all `.txt` files starting from the current directory (`.`)

`find . -type d`:

- **`find`**: This is used to search for files or directories.
- `.`: The dot (`.`) refers to the **current directory**. This tells `find` to start the search from the current directory and include its subdirectories.
- **`-type d`**: This option tells `find` to **only search for directories** (d stands for directory), not files.

What this command does: It will list all directories (including subdirectories) starting from the current directory.

TAR

Both **`tar`** and **`zip`** are commonly used for **compressing** and **archiving** files in Linux (and Unix-like systems). They differ in the way they handle compression and the files they create.

1. `tar` (Tape Archive)

- **`tar`** is a command used to **create** and **extract** archives (collections of files) in Linux and Unix-based systems. It doesn't compress files by default but can be combined with compression algorithms like **gzip** or **bzip2** to compress the archive.

Common `tar` Commands:

a. Create a tar archive:

```
tar -cvf archive.tar file1 file2 folder/
```

- **`-c`**: Create a new archive.
- **`-v`**: Verbose output, lists files being archived.
- **`-f`**: Specifies the name of the archive file (`archive.tar`).

Example: To create a `tar` archive of `file1`, `file2`, and the folder `folder/`, the command would be:

This creates an archive called `archive.tar` containing `file1`, `file2`, and the contents of `folder/`.

b. Extract a tar archive:

```
tar -xvf archive.tar
```

- **-x**: Extract files from an archive.
- **-v**: Verbose output (optional).
- **-f**: Specifies the archive file to extract.

Example: To extract the contents of `archive.tar`, you would use:

This extracts all the files from `archive.tar` into the current directory.

c. Create a compressed tar archive (with gzip):

```
tar -czvf archive.tar.gz file1 file2 folder/
```

- **-z**: Use **gzip** compression.
- This creates a **compressed archive** (`.tar.gz` or `.tgz`).

Example: To create a compressed `tar` archive with `gzip`

d. Extract a compressed tar archive (with gzip):

```
tar -xzvf archive.tar.gz
```

- **-z**: This tells `tar` to use `gzip` for decompression.

Example: To extract a `.tar.gz` archive

ZIP

zip

- **zip** is a **compression** tool used to package files into a single compressed file (ending in `.zip`).
- Unlike `tar`, `zip` compresses files **by default**, so you don't need to use an external compression tool.
- It is more widely known for creating **.zip** files, which are commonly used in **Windows** environments, though it works on Linux as well.

Create a zip archive:

```
zip archive.zip file1 file2 folder/
```

- **archive.zip**: The name of the zip file you want to create.
- **file1 file2 folder/**: The files and directories you want to add to the archive.

Example: To create a `.zip` archive containing `file1`, `file2`, and the `folder/`, you would use

b. Extract a zip archive:

```
unzip archive.zip
```

- **unzip**: The command used to extract a `.zip` archive.
- **archive.zip**: The zip file you want to extract.

Example: To extract the contents of `archive.zip`, you would run

c. Add a file to an existing zip archive:

```
zip archive.zip newfile
```

- This adds `newfile` to the existing `archive.zip` file.

d. View the contents of a zip archive:

```
unzip -l archive.zip
```

- `-l`: List the contents of the zip file without extracting them.

NOTE:- `alias j1="ls -lrt"` creates a **shortcut alias** for a commonly used command in Linux. Now j1 can be used to run ls-lrt command.

Note:- lsof will load all the files that are running in the background.

NOTE: ip a it will show the v4 and v6 address.

DISK ANALYZER

```
sudo apt install ncdu
```

```
ncdu .
```

- **ncdu** stands for **NCurses Disk Usage**. It's a disk usage analyzer tool that provides a more **interactive** way to check how disk space is being used on a directory and its subdirectories.

What this command does: It runs ncdu in the **current directory**, and it will show you a graphical representation of how disk space is being used, making it easy to identify which files or directories are consuming the most space.

Example Output: You might see something like:

```
  3.2 GiB [##########] /Documents

1.5 GiB [####      ] /Downloads

800 MiB [###       ] /Pictures
```

…

TMUX

```
Start using new session without opening new terminal.
```

Create Variable

```
Vi code.sh
{
namej="saumya"
echo $namej
}
Chmod 444 code.sh
Chmod 744 code.sh
./code.sh

saumya
```

[2]

var_1="saumya"

var_2="tripathi"

```
echo "$var_1$var_2"
```

[3]

```
var_1="saumya"
var_2="tripathi"
echo "$var_1 $var_2"
unset var_1
echo "$var_1"
readonly var_2
#var_2="saumyatripathi"
```

[4]

```
var_name="saumya"
var_age=23
echo " Name is $var_name and age is $var_age"
var_blood_group ="0-"
readonly var_blood_group
echo "Blood group is $var_blood_group"
echo "Error modifying readonly varaiable please dont modify it "
echo var_blood_group="b+"
echo
unset var_age
echo "Age is after unsetting $var_age"
```

[5]

```
time=$(date +%H)
echo $time
if [ $time -lt 12 ];then
message="Good morning user"
elif [ $time -lt 18 ];then
message="Good afternoon user"
else
message="Good evening user"
fi
echo "$message $time"
```

```
40 awk '{print $1}' data.txt
141 awk '{print $2}' data.txt
142 awk '{print $3}' data.txt
143 awk '{print $1 $3}' data.txt
144 awk '{print "name" $1, profession " $3}' data.txt
145 awk '{print "name" $1," profession " $3}' data.txt
146 awk '{print "AWS name " $1,"AWS profession " $3}' data.txt
147 awk '/Engineer/' data.txt
148 awk '/Enginner/' data.txt
149 awk '/enginner/' data.txt
150 awk '/Enginner/' data.txt
151 awk '/Enginner/ {print $1}' data.txt
152 awk '$2 >25 {print $1 , "is older than 25}' data.txt
153 awk '$2 >25 {print $1 , "is older than 25"}' data.txt
```

[6]

```
i=1
while [ $i -lt 5 ];
do
echo "saumya"
i=`expr $i + 1`
done
```

[7]

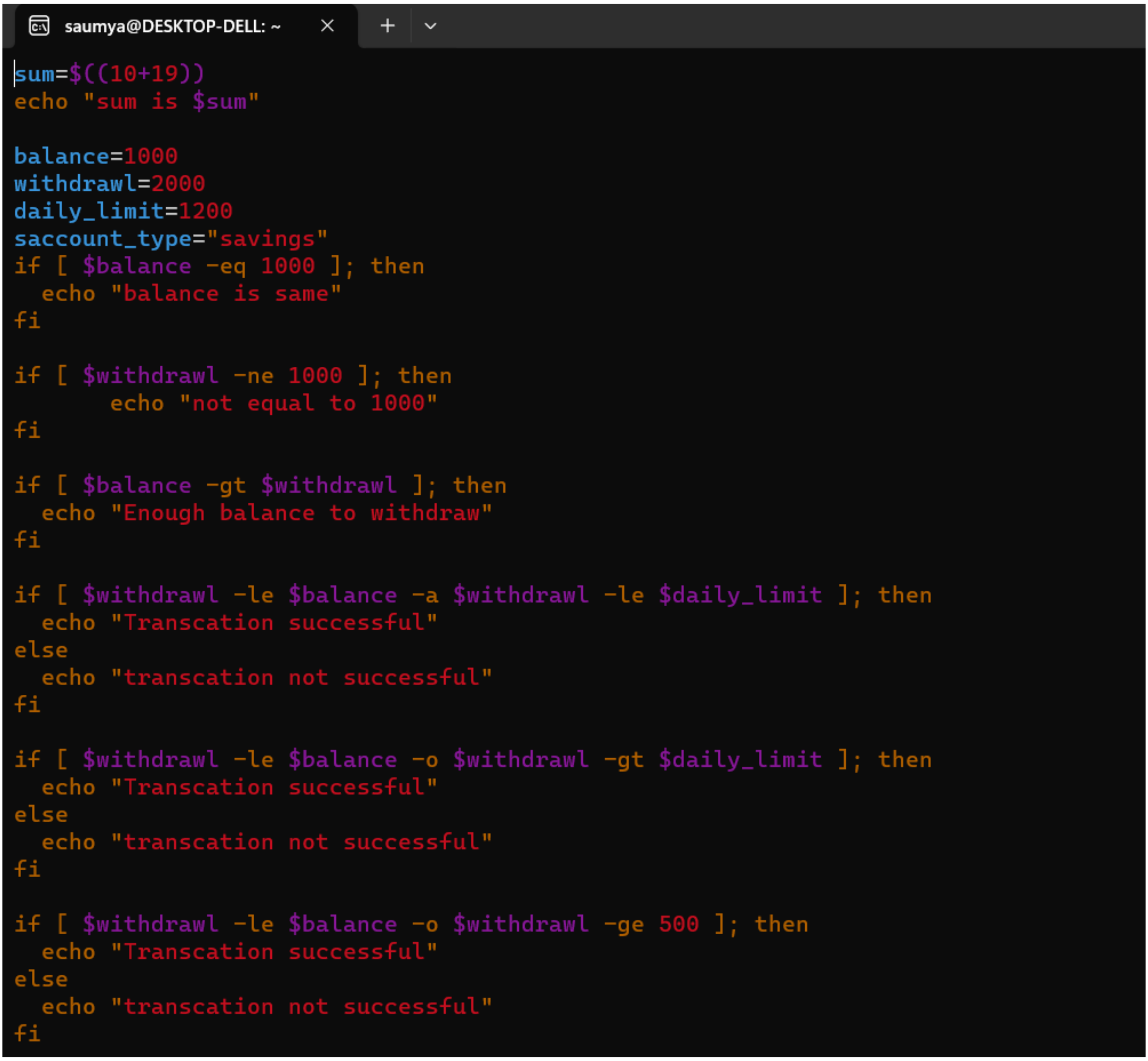
```
i=1
while [ $i -lt 5 ]
do
echo "saumya"
i=`expr $i + 1`
done
#a=0
for a in 1 2 3 4 5 6 7 8 9
do
if [ $a == 5 ]
then
break
```

```
fi
echo "iteration is $a"
done
```

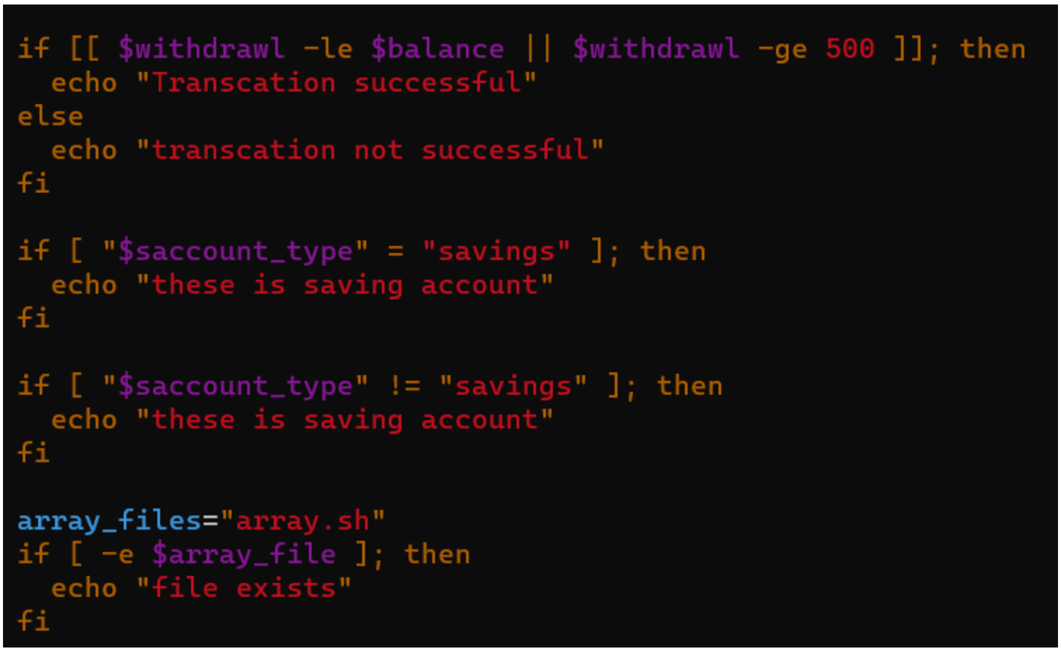
D3

19 February 2025 09:49

OPERATORS

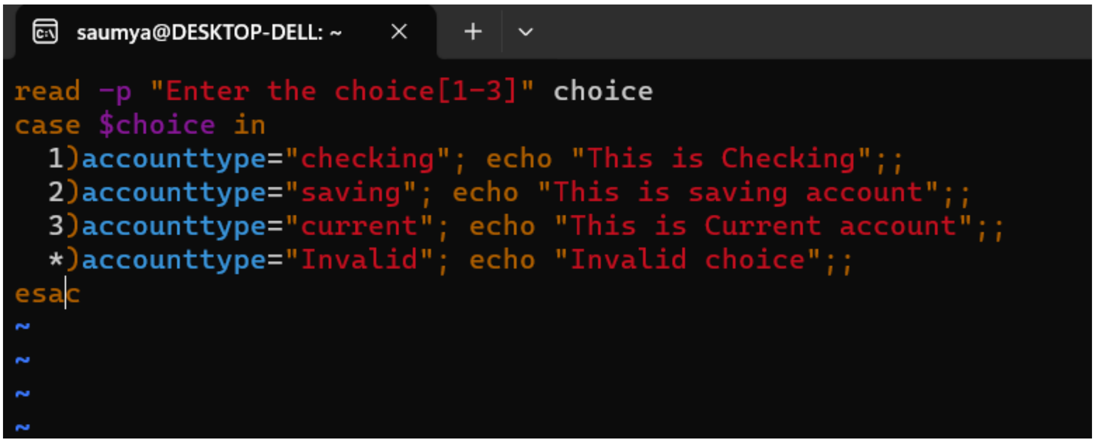


```
sum=$((10+19))
echo "sum is $sum"

balance=1000
withdrawl=2000
daily_limit=1200
saccount_type="savings"
if [ $balance -eq 1000 ]; then
  echo "balance is same"
fi

if [ $withdrawl -ne 1000 ]; then
       echo "not equal to 1000"
fi

if [ $balance -gt $withdrawl ]; then
  echo "Enough balance to withdraw"
fi

if [ $withdrawl -le $balance -a $withdrawl -le $daily_limit ]; then
  echo "Transcation successful"
else
  echo "transcation not successful"
fi

if [ $withdrawl -le $balance -o $withdrawl -gt $daily_limit ]; then
  echo "Transcation successful"
else
  echo "transcation not successful"
fi

if [ $withdrawl -le $balance -o $withdrawl -ge 500 ]; then
  echo "Transcation successful"
else
  echo "transcation not successful"
fi
```

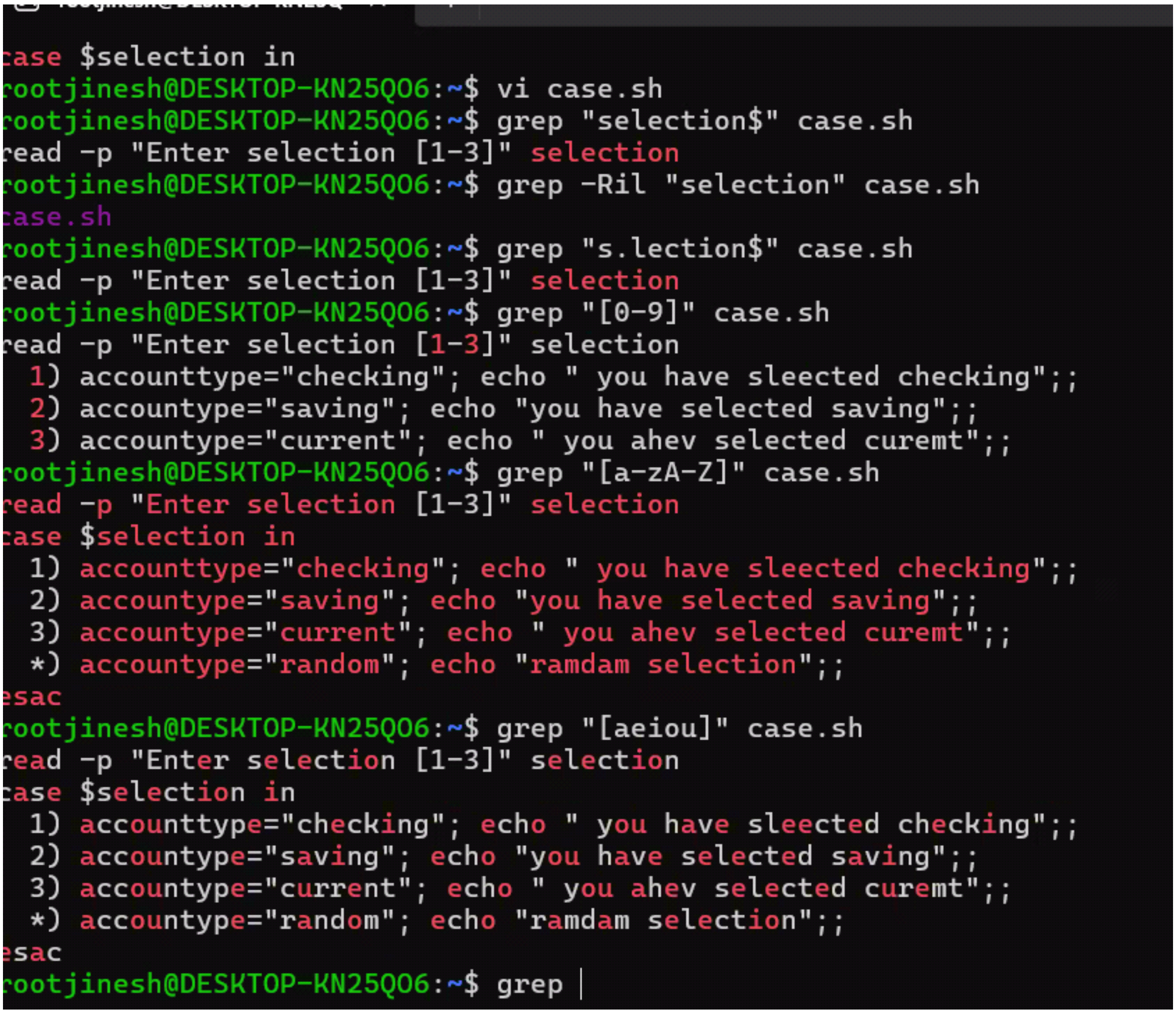


```
if [[ $withdrawl -le $balance || $withdrawl -ge 500 ]]; then
  echo "Transcation successful"
else
  echo "transcation not successful"
fi

if [ "$saccount_type" = "savings" ]; then
  echo "these is saving account"
fi

if [ "$saccount_type" != "savings" ]; then
  echo "these is saving account"
fi

array_files="array.sh"
if [ -e $array_file ]; then
  echo "file exists"
fi
```

CASE SELECTION



```
read -p "Enter the choice[1-3]" choice
case $choice in
  1)accounttype="checking"; echo "This is Checking";;
  2)accounttype="saving"; echo "This is saving account";;
  3)accounttype="current"; echo "This is Current account";;
  *)accounttype="Invalid"; echo "Invalid choice";;
esac
~
~
~
~
```



```
ase $selection in
ootjinesh@DESKTOP-KN25Q06:~$ vi case.sh
ootjinesh@DESKTOP-KN25Q06:~$ grep "selection$" case.sh
ead -p "Enter selection [1-3]" selection
ootjinesh@DESKTOP-KN25Q06:~$ grep -Ril "selection" case.sh
ase.sh
ootjinesh@DESKTOP-KN25Q06:~$ grep "s.lection$" case.sh
ead -p "Enter selection [1-3]" selection
ootjinesh@DESKTOP-KN25Q06:~$ grep "[0-9]" case.sh
ead -p "Enter selection [1-3]" selection
  1) accounttype="checking"; echo " you have sleected checking";;
  2) accountype="saving"; echo "you have selected saving";;
  3) accountype="current"; echo " you ahev selected curemt";;
ootjinesh@DESKTOP-KN25Q06:~$ grep "[a-zA-Z]" case.sh
ead -p "Enter selection [1-3]" selection
ase $selection in
  1) accounttype="checking"; echo " you have sleected checking";;
  2) accountype="saving"; echo "you have selected saving";;
  3) accountype="current"; echo " you ahev selected curemt";;
  *) accountype="random"; echo "ramdam selection";;
sac
ootjinesh@DESKTOP-KN25Q06:~$ grep "[aeiou]" case.sh
ead -p "Enter selection [1-3]" selection
ase $selection in
  1) accounttype="checking"; echo " you have sleected checking";;
  2) accountype="saving"; echo "you have selected saving";;
  3) accountype="current"; echo " you ahev selected curemt";;
  *) accountype="random"; echo "ramdam selection";;
sac
ootjinesh@DESKTOP-KN25Q06:~$ grep
```

1. grep "^read" file.sh

Purpose: Finds lines in file.sh that start with the word read.

Explanation: The ^ symbol is an anchor that matches the beginning of a line. This command will return any line in file.sh where read is the first word.

2. grep "^case" file.sh

Purpose: Finds lines in file.sh that start with the word case.

Explanation: The ^ symbol ensures the match is only at the beginning of the line. This command will return lines where case is the first word.

3. grep "selection$" file.sh

Purpose: Finds lines in file.sh that end with the word selection.

Explanation: The $ symbol is an anchor that matches the end of a line. This command will return lines that end with selection.

4. grep "s.lection" file.sh

Purpose: Finds lines in file.sh that contain s followed by any character and then lection.

Explanation: The . character is a wildcard that matches any single character. This command will match slection, selection, s-lection, etc.

5. grep "s..lection" file.sh

Purpose: Finds lines in file.sh where s is followed by any two characters and then lection.

Explanation: The .. (two dots) represent exactly two characters, so this command will match strings like slection, selection, s@lection, etc.

6. grep "[0-9]" file.sh

Purpose: Finds lines in file.sh that contain at least one digit.

Explanation: The [0-9] is a character class that matches any digit from 0 to 9. It will return lines with any numeric characters.

7. grep "[a-zA-Z]" file.sh

Purpose: Finds lines in file.sh that contain at least one alphabetic character (lowercase or uppercase).

Explanation: The [a-zA-Z] character class matches any lowercase (a-z) or uppercase (A-Z) letter. This command will return lines with alphabetic characters.

8. grep "[aeiou]" case.sh

Purpose: Finds lines in case.sh that contain at least one vowel (a, e, i, o, or u).

Explanation: The [aeiou] character class matches any of the vowels in the specified set. It will return lines with at least one vowel.

9. grep "s*n" case.sh

Purpose: Finds lines in case.sh where s is followed by zero or more s characters and ends with n.

Explanation: The * is a wildcard that matches zero or more occurrences of the preceding character (s). This command will match strings like sn, ssn, sssn, etc.

10. grep "se*n" case.sh

Purpose: Finds lines in case.sh where se is followed by zero or more e characters and ends with n.

Explanation: The * wildcard applies to the e, meaning any occurrence of e (including zero occurrences). This will match strings like sen, seen, seeeen, etc.

11. grep "selecti*n" case.sh

Purpose: Finds lines in case.sh where selecti is followed by zero or more i characters and ends with n.

Explanation: The * wildcard applies to i, meaning any number of i characters (including none). It will match selection, selectin, selectiiiion, etc.

12. grep "selection" case.sh

Purpose: Finds lines in case.sh that contain the exact word selection.

Explanation: This command looks for the exact string selection in the file. It will match any line where selection appears exactly as it is.

13. grep "sel.n" case.sh

Purpose: Finds lines in case.sh where sel is followed by any character and ends with n.

Explanation: The . wildcard matches any single character, so this will match strings like selin, selan, selxn, etc.

14. grep "selicti.n" case.sh

Purpose: Finds lines in case.sh where selicti is followed by any character and ends with n.

Explanation: The . wildcard matches any single character, so it will match strings like selictiin, selictian, etc.

15. grep "selecti.n" case.sh

Purpose: Finds lines in case.sh where selecti is followed by any character and ends with n.

Explanation: The . wildcard matches any single character, so it will match strings like selectin, selectin, etc.

16. grep "s*n" case.sh (repeated)

Purpose: Same as command #9. Finds lines where s is followed by zero or more s characters and ends with n.

17. grep "s*on" case.sh

Purpose: Finds lines in case.sh where s is followed by zero or more s characters and ends with on.

Explanation: The * wildcard applies to s, matching zero or more occurrences of s. This will match strings like son, sson, sss on, etc.

```
read -s -p "Enter password: " p
```

- **Purpose**: Reads a password input silently (i.e., the input will not be visible) and stores it in the p variable.
- **Explanation**:
  - `-s`: This option suppresses the echo of characters typed by the user, making it suitable for password entry, so the input isn't shown on the screen.
  - `-p "Enter password: "`: Displays the prompt "Enter password: ".
  - p: This is the variable where the input will be stored.
  - **Example**: The user will type a password, but the characters won't be displayed in the terminal.

```
read -t 5 -p "Quick 5 sec: " pin
```

- **Purpose**: Waits for user input for 5 seconds and assigns the input to the variable `pin`.
- **Explanation**:
  - `-t 5`: This option sets a timeout of 5 seconds. If the user doesn't provide input within this time frame, the script will proceed without waiting.
  - `-p "Quick 5 sec: "`: This option displays the prompt message "Quick 5 sec: ".
  - After 5 seconds, if the user doesn't input anything, the `pin` variable will remain empty.
- **Example**: If the user enters a pin within 5 seconds, it will be saved to the `pin` variable. If they don't, the script moves on after 5 seconds.

Key Concepts:

^: Anchors the match to the beginning of the line.

$: Anchors the match to the end of the line.

.: Matches any single character (except a newline).

*: Matches zero or more occurrences of the preceding character.

[ ]: Matches any character within the brackets.

grep: Searches for patterns in a file or input.

Sr. no.	Symbol	Description
1.	.	It is called a wild card character, It matches any one character other than the new line.
2.	^	It matches the start of the string.
3.	$	It matches the end of the string.
4.	*	It matches up to zero or more occurrences i.e. any number of times of the character of the string.
5.	\	It is used for escape following character.
6.	()	It is used to match or search for a set of regular expressions.
7.	?	It matches exactly one character in the string or stream.

D4

20 February 2025 15:07

grep -E "completed in [1-9][0-9]{X,Y}ms" logfile.txt

- grep: Command used for searching text within files.
- -E: Enables **extended regular expressions** (ERE), allowing for more advanced pattern matching.
- "completed in [1-9][0-9]{X,Y}ms": The pattern being searched for in the file (logfile.txt).

Regular Expression Breakdown:

- **completed in** → Matches the phrase literally.
- **[1-9]** → Matches a digit between 1 and 9 (ensuring numbers don't start with 0).
- **[0-9]{X,Y}** → Matches a number with a range of digits:
  - {X,Y} specifies the minimum (X) and maximum (Y) number of digits.
  - Example:
    - {3} → Matches exactly 3 digits (e.g., 278ms).
    - {2,} → Matches at least 2 digits (e.g., 21ms, 215ms, 2781ms).
    - {1,3} → Matches between 1 and 3 digits (e.g., 5ms, 27ms, 312ms).
- **ms** → Matches the literal "ms" (milliseconds).

1. **grep -E "completed in [1-9][0-9]{3}ms" logfile.txt**
   - Matches API request times that are exactly **3 digits** long.
   - Example matches: 278ms, 312ms, 215ms.
2. **grep -E "completed in [1-9][0-9]{2,}ms" logfile.txt**
   - Matches API request times that are at least **2 digits** long.
   - Example matches: 21ms, 278ms, 2156ms, 7245ms.
3. **grep -E "completed in [1-9][0-9]{1,}ms" logfile.txt**
   - Matches API request times that have at least **1 digit** (effectively any millisecond value).
   - Example matches: 5ms, 27ms, 312ms, 1878ms.
4. **grep -E "completed in [1-9][0-9]{1,4}ms" logfile.txt**
   - Matches API request times that are between **1 to 4 digits** long.
   - Example matches: 5ms, 278ms, 1878ms, 7245ms.
5. **grep -E "completed in [1-9][0-9]{1,3}ms" logfile.txt**
   - Matches API request times that are between **1 to 3 digits** long.
   - Example matches: 5ms, 27ms, 278ms, 312ms (but not 1878ms or 7245ms).

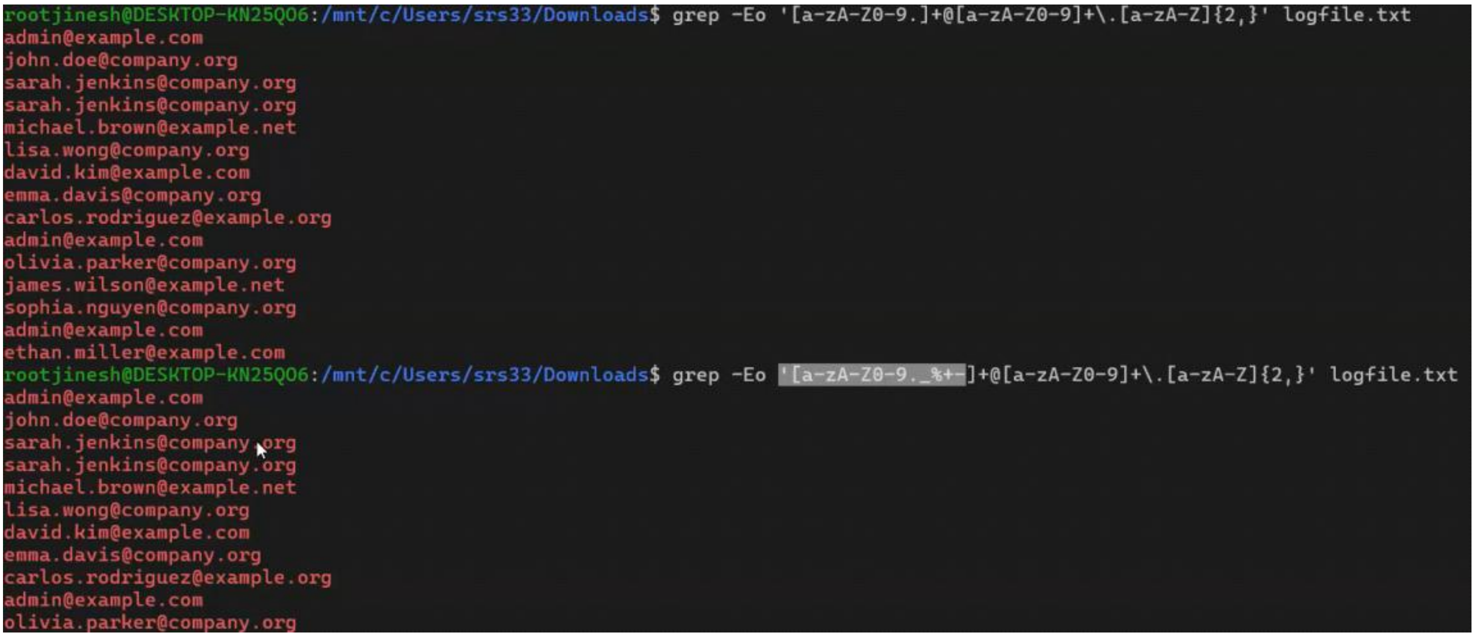
Command Syntax:

grep -E "CPU usage: .*[7-9][0-9]%" logfile.txt

- grep: Searches for patterns in a file.
- -E: Enables **extended regular expressions (ERE)** for more complex matching.
- "CPU usage: .*[7-9][0-9]%": The **pattern** being searched.
- logfile.txt: The log file being searched.

Regular Expression Breakdown:

1. **CPU usage:** → Matches this phrase literally.
2. **.*** → Matches **any characters (including spaces) zero or more times** before the percentage value.
3. **[7-9][0-9]%**:
   - [7-9] → Matches a digit between **7 and 9** (ensures CPU usage is at least 70%).
   - [0-9] → Matches any digit **0-9** (ensures two-digit numbers like 70, 85, 99, etc.).
   - % → Matches the literal percentage sign.



```
rootjinesh@DESKTOP-KN25Q06:/mnt/c/Users/srs33/Downloads$ grep -Eo '[a-zA-Z0-9.]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,}' logfile.txt
admin@example.com
john.doe@company.org
sarah.jenkins@company.org
sarah.jenkins@company.org
michael.brown@example.net
lisa.wong@company.org
david.kim@example.com
emma.davis@company.org
carlos.rodriguez@example.org
admin@example.com
olivia.parker@company.org
james.wilson@example.net
sophia.nguyen@company.org
admin@example.com
ethan.miller@example.com
rootjinesh@DESKTOP-KN25Q06:/mnt/c/Users/srs33/Downloads$ grep -Eo '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,}' logfile.txt
admin@example.com
john.doe@company.org
sarah.jenkins@company.org
sarah.jenkins@company.org
michael.brown@example.net
lisa.wong@company.org
david.kim@example.com
emma.davis@company.org
carlos.rodriguez@example.org
admin@example.com
olivia.parker@company.org
```

Ollama is an open-source tool designed to simplify the deployment and management of large language models (LLMs) locally on your machine. It provides an easy-to-use interface for running, fine-tuning, and experimenting with LLMs without requiring extensive technical expertise.

Key Features:

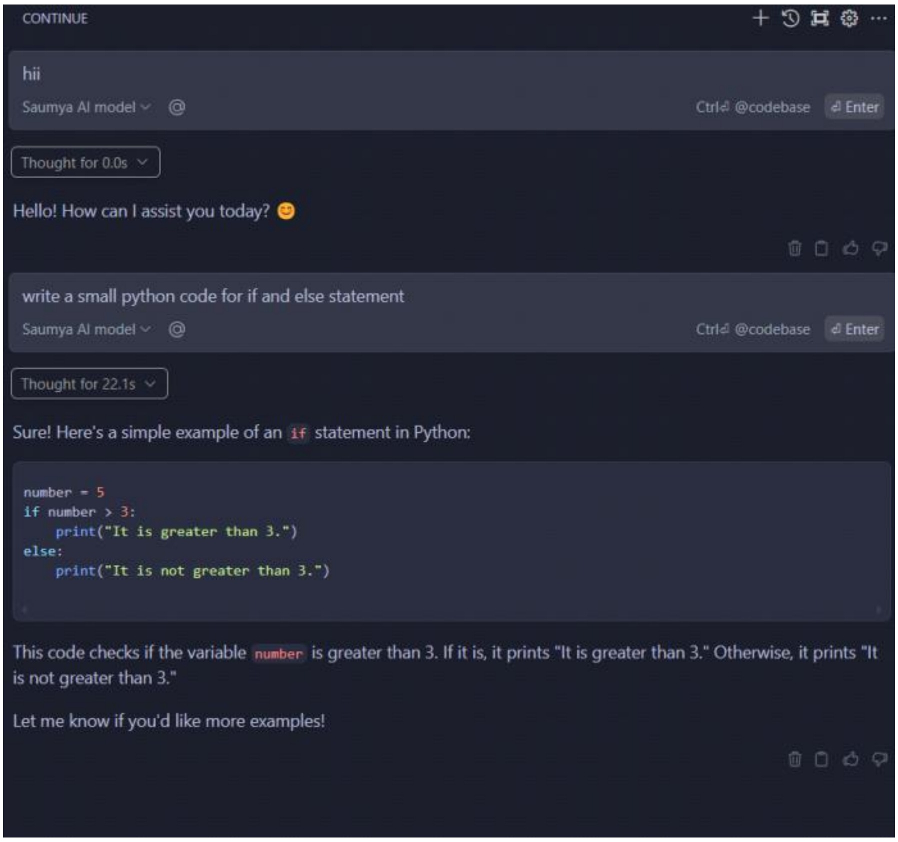
- Supports a variety of pre-trained models (e.g., LLaMA, GPT-based models).
- Enables local deployment, ensuring data privacy and security.
- Provides a command-line interface (CLI) for easy interaction.
- Allows customization and fine-tuning of models for specific tasks.

Primary Uses:

- **AI Development**: Experiment with and build applications using LLMs.
- **Research**: Test and analyze model performance for academic or professional research.
- **Custom Solutions**: Fine-tune models for specific use cases like chatbots, content generation, or data analysis.
- **Education**: Learn about LLMs and AI in a hands-on manner.
- **Offline AI**: Run AI models locally without relying on cloud services.

Advantages:

- **Privacy**: Data remains on your local machine.
- **Customization**: Tailor models to your specific needs.
- **Accessibility**: Simplifies the process of working with advanced AI models.



hii

Hello! How can I assist you today? 😊

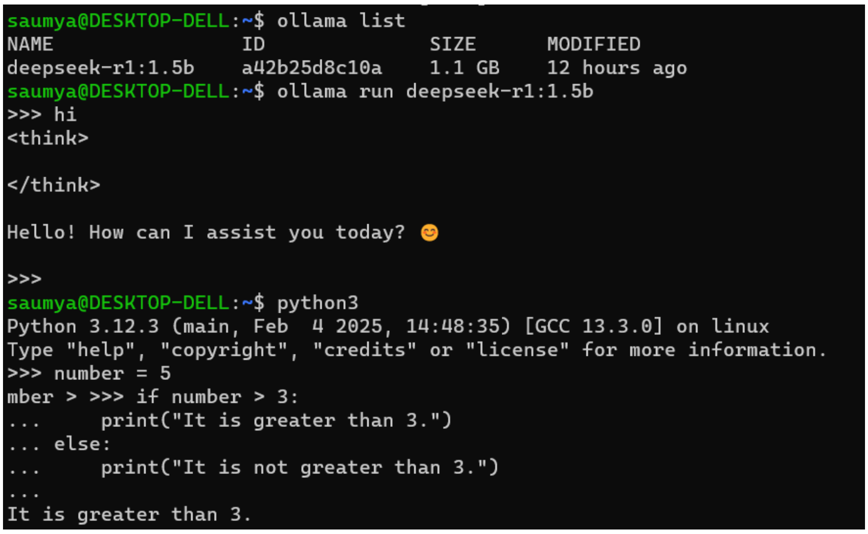
write a small python code for if and else statement

Sure! Here's a simple example of an `if` statement in Python:

```
number = 5
if number > 3:
    print("It is greater than 3.")
else:
    print("It is not greater than 3.")
```

This code checks if the variable `number` is greater than 3. If it is, it prints "It is greater than 3." Otherwise, it prints "It is not greater than 3."

Let me know if you'd like more examples!



```
saumya@DESKTOP-DELL:~$ ollama list
NAME                ID              SIZE      MODIFIED
deepseek-r1:1.5b    a42b25d8c10a    1.1 GB    12 hours ago
saumya@DESKTOP-DELL:~$ ollama run deepseek-r1:1.5b
>>> hi
<think>

</think>

Hello! How can I assist you today? 😊

>>>
saumya@DESKTOP-DELL:~$ python3
Python 3.12.3 (main, Feb  4 2025, 14:48:35) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> number = 5
mber > >>> if number > 3:
...     print("It is greater than 3.")
... else:
...     print("It is not greater than 3.")
...
It is greater than 3.
```

D1

26 February 2025 02:45

1. Virtual Environment
What: A separate space for your Python project to keep its dependencies isolated.

Why: Avoids conflicts between different projects.

How:

Create: python -m venv venv

Activate: source venv/bin/activate (macOS/Linux) or venv\Scripts\activate (Windows)

Deactivate: deactivate

2. .gitignore
What: A file that tells Git which files/folders to ignore.

Why: Keeps unnecessary files (like cache or logs) out of your Git repository.

Example:

Copy

```
__pycache__/
venv/
*.log
```

3. pyproject.toml
What: A config file for Python projects.

Why: Manages dependencies, build settings, and tool configurations.

Example:

toml
Copy

```
[build-system]
requires = ["setuptools", "wheel"]

[project]
name = "my_project"
version = "0.1.0"
```

4. wheel
What: A format for Python packages.

Why: Makes installing packages faster and easier.

How: Install a .whl file: pip install <package>.whl

5. build
What: A tool to create Python packages.

Why: Standardizes the process of building packages.

How:

Install: pip install build

Build: python -m build

```
n=10
for i in range(1,n):
    print(" "*(n-i),end=" ")
    for j in range (1,i):
        print(j,end=" ")
    print()
```

```
N=9
j=0

for i in range (0,N):
    for j in range (0,N-i):
        print(" ", end="")
    for j in range (0, i):
        print("*", end=" ")
    print("")
```

```
for i in range(0,N):
    for j in range(0,i):
        print(j ,end=" ")
    print("")
```

```
for i in range(0,N):
    for j in range(0,i):
        print(i ,end=" ")
    print("")
print()
```

```
for i in range(N,0,-1):
    for j in range(0,i):
        print(i ,end=" ")
    print("")
print("")
```

```
for i in range(1, N + 1):
    print(" " * (N - i), end=" ")
    print(f"{i} " * i)
```

```
print(" ".join(map(str,range(1,10))))
for i in range(1, N + 1):
    print(" " * (N - i)+" ".join(map(str,(1,i))))
```

D2

26 February 2025 02:51

Docker in Short

Docker is a platform for developing, shipping, and running applications using containers. It consists of three main parts:

1. Docker CLI: Command-line tool to interact with Docker.
2. Docker Daemon: Background service managing Docker objects (images, containers, etc.).
3. Docker REST API: Allows remote communication with the Docker daemon.

Key Concepts:
- Docker Image: A read-only template with instructions to create a container. It includes the app code, libraries, and dependencies.
  - Example: `ubuntu:20.04`.
- Docker Container: A running instance of an image. It's isolated, lightweight, and shares the host OS kernel.
  - Example: Run a container with `docker run -it ubuntu:20.04`.

Workflow:
1. Create a Dockerfile to define the app environment.
2. Build an image with `docker build`.
3. Run a container with `docker run`.

Docker ensures apps run consistently across environments by packaging everything into containers.

Image Commands
Pull an image:

--docker pull <image_name>:<tag>
Downloads an image from a registry (e.g., Docker Hub).

List images:

--docker images
Shows all downloaded images.

Remove an image:

--docker rmi <image_name>:<tag>
Deletes an image.

Container Commands

Run a container:

--docker run <image_name>:<tag>
Starts a container from an image. Add -d to run in the background.

List running containers:

```
--docker ps
```

Shows all running containers.

List all containers:

```
--docker ps -a
```

Shows all containers (running and stopped).

Stop a container:

```
--docker stop <container_name_or_id>
```

Stops a running container.

Start a stopped container:

```
--docker start <container_name_or_id>
```

Restarts a stopped container.

Remove a container:

```
--docker rm <container_name_or_id>
```

Deletes a stopped container.

View container logs:

```
--docker logs <container_name_or_id>
```

Displays logs from a container.

Run a command in a running container:

```
--docker exec -it <container_name_or_id> <command>
```

Executes a command inside a running container (e.g., bash).

System Commands

Check Docker version:

```
--docker --version
```

Shows the installed Docker version.

Clean up unused data:

```
--docker system prune
```

Removes stopped containers, unused images, and networks.