Assignment Report: Distributed System with Django, Threads, and Multiple Databases

Submitted By:

Name: Pranali Kalokhe

1. Objective

The goal of this assignment was to create a Django project that simulates a distributed system by using **three separate SQLite databases** (users.db, products.db, and orders.db).
The requirements were:

- Create tables manually.
- Insert at least 10 records into each table.
- Insert data **concurrently using threads**.
- Perform **all validations in Python**.
- Display the output after insertion and verify data using SQLite.

2. Project Structure

```
E:\distributed_system_assignment\
│
├── distributed_system\
│   ├── settings.py          # Multiple database configuration
│
├── core\
│   ├── models.py             # Models for Users, Products, Orders
│   ├── management\
│   │   └── commands\
│   │       ├── init_db.py        # Create tables
│   │       └── insert_data.py     # Insert data concurrently using
threads
│
├── users.db                 # Users database
├── products.db                # Products database
├── orders.db                 # Orders database
```

3. Validation Rules

Validations are applied in Python, not in the database:

Table	Validation Rules
Users	Name must not be empty, Email must contain '@'
Products	Price must be greater than 0
Orders	Quantity must be greater than 0, User & Product IDs must exist

4. Inserted vs. Skipped Records

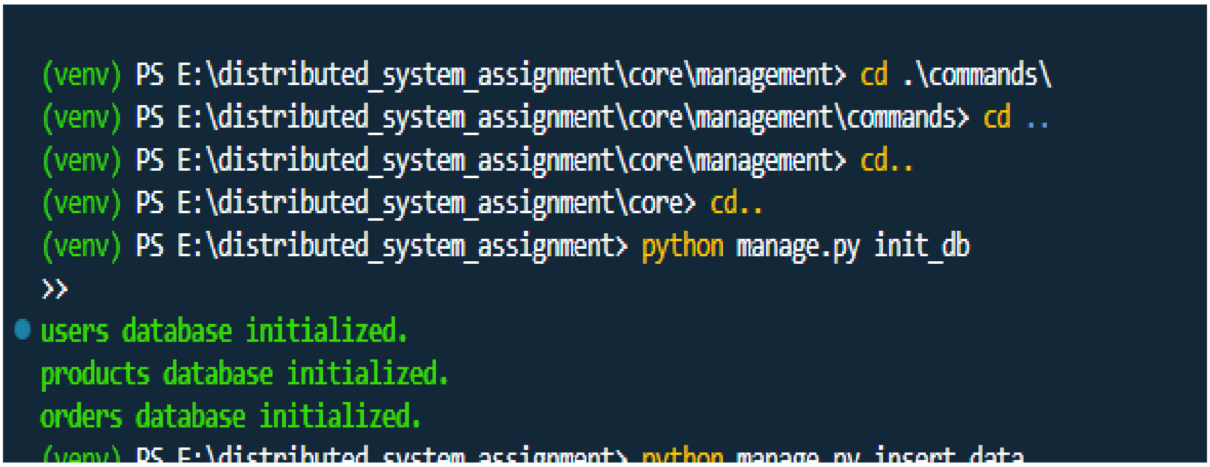
Table	Total Provided	Inserted	Skipped	Reason for Skipped Records
Users	10	9	1	User #10 → Empty name
Products	10	9	1	Product #10 → Negative price (-50)
Orders	10	7	3	Order #8 → Quantity 0 Order #9 → Quantity -1 Order #10 → Invalid product_id (11)

5. Execution Process

Step 1: Initialize Databases

Output:

users database initialized.
products database initialized.
orders database initialized.



```
(venv) PS E:\distributed_system_assignment\core\management> cd .\commands\
(venv) PS E:\distributed_system_assignment\core\management\commands> cd ..
(venv) PS E:\distributed_system_assignment\core\management> cd..
(venv) PS E:\distributed_system_assignment\core> cd..
(venv) PS E:\distributed_system_assignment> python manage.py init_db
>>
users database initialized.
products database initialized.
orders database initialized.
(venv) PS E:\distributed_system_assignment> python manage.py insert_data
```

Step 2: Insert Data Concurrently

python manage.py insert_data

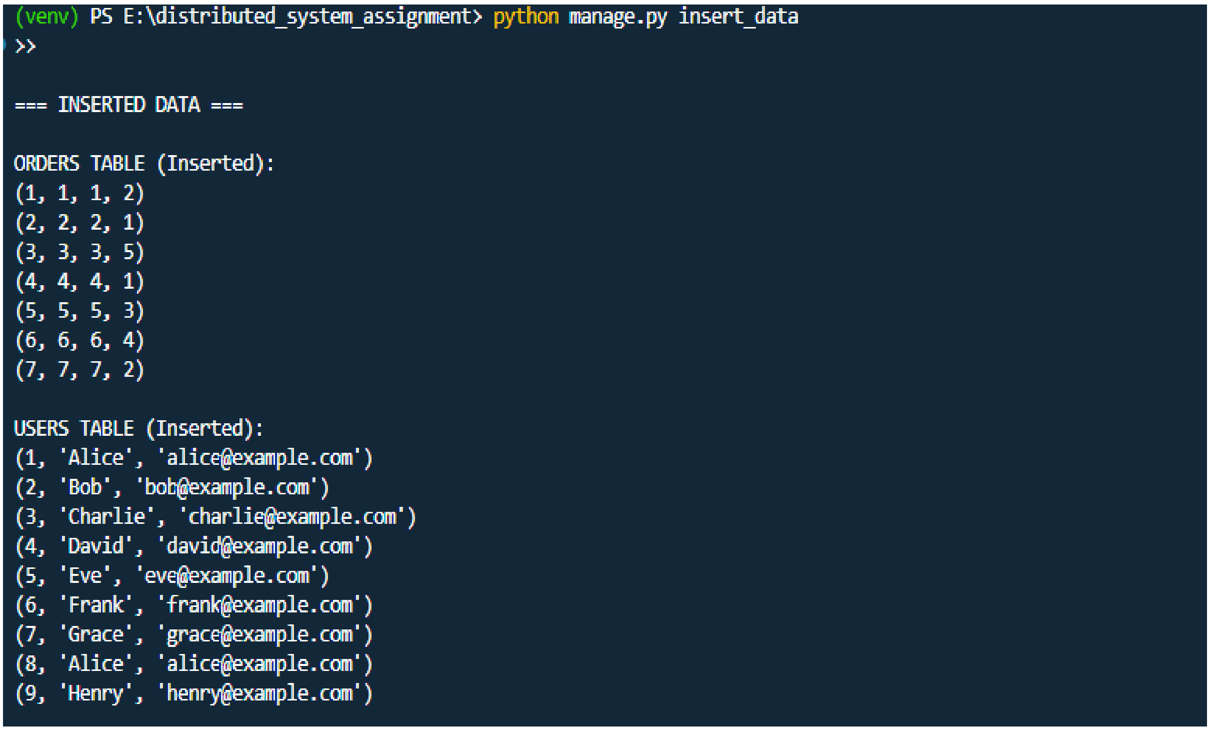
=== INSERTED DATA ===
(venv) PS E:\distributed_system_assignment> python manage.py insert_data
>>

Output:

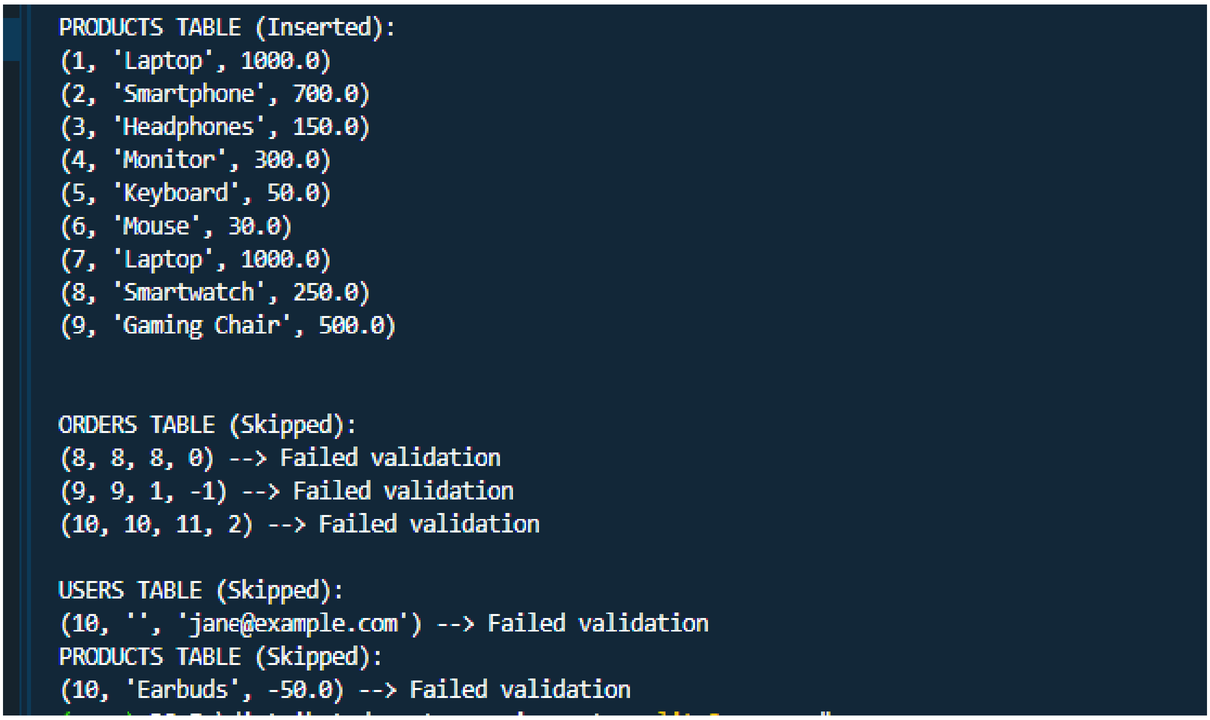
(venv) PS E:\distributed_system_assignment> python manage.py insert_data

```
>>

=== INSERTED DATA ===

ORDERS TABLE (Inserted):
(1, 1, 1, 2)
(2, 2, 2, 1)
(3, 3, 3, 5)
(4, 4, 4, 1)
(5, 5, 5, 3)
(6, 6, 6, 4)
(7, 7, 7, 2)

USERS TABLE (Inserted):
(1, 'Alice', 'alice@example.com')
(2, 'Bob', 'bob@example.com')
(3, 'Charlie', 'charlie@example.com')
(4, 'David', 'david@example.com')
(5, 'Eve', 'eve@example.com')
(6, 'Frank', 'frank@example.com')
(7, 'Grace', 'grace@example.com')
(8, 'Alice', 'alice@example.com')
(9, 'Henry', 'henry@example.com')

PRODUCTS TABLE (Inserted):
```

```
(1, 'Laptop', 1000.0)
(2, 'Smartphone', 700.0)
(3, 'Headphones', 150.0)
(4, 'Monitor', 300.0)
(5, 'Keyboard', 50.0)
(6, 'Mouse', 30.0)
(7, 'Laptop', 1000.0)
(8, 'Smartwatch', 250.0)
(9, 'Gaming Chair', 500.0)


ORDERS TABLE (Skipped):
(8, 8, 8, 0) --> Failed validation
(9, 9, 1, -1) --> Failed validation
(10, 10, 11, 2) --> Failed validation

USERS TABLE (Skipped):
(10, '', 'jane@example.com') --> Failed validation

PRODUCTS TABLE (Skipped):
(10, 'Earbuds', -50.0) --> Failed validation
```



```
(venv) PS E:\distributed_system_assignment> python manage.py insert_data
>>

=== INSERTED DATA ===

ORDERS TABLE (Inserted):
(1, 1, 1, 2)
(2, 2, 2, 1)
(3, 3, 3, 5)
(4, 4, 4, 1)
(5, 5, 5, 3)
(6, 6, 6, 4)
(7, 7, 7, 2)

USERS TABLE (Inserted):
(1, 'Alice', 'alice@example.com')
(2, 'Bob', 'bob@example.com')
(3, 'Charlie', 'charlie@example.com')
(4, 'David', 'david@example.com')
(5, 'Eve', 'eve@example.com')
(6, 'Frank', 'frank@example.com')
(7, 'Grace', 'grace@example.com')
(8, 'Alice', 'alice@example.com')
(9, 'Henry', 'henry@example.com')
```



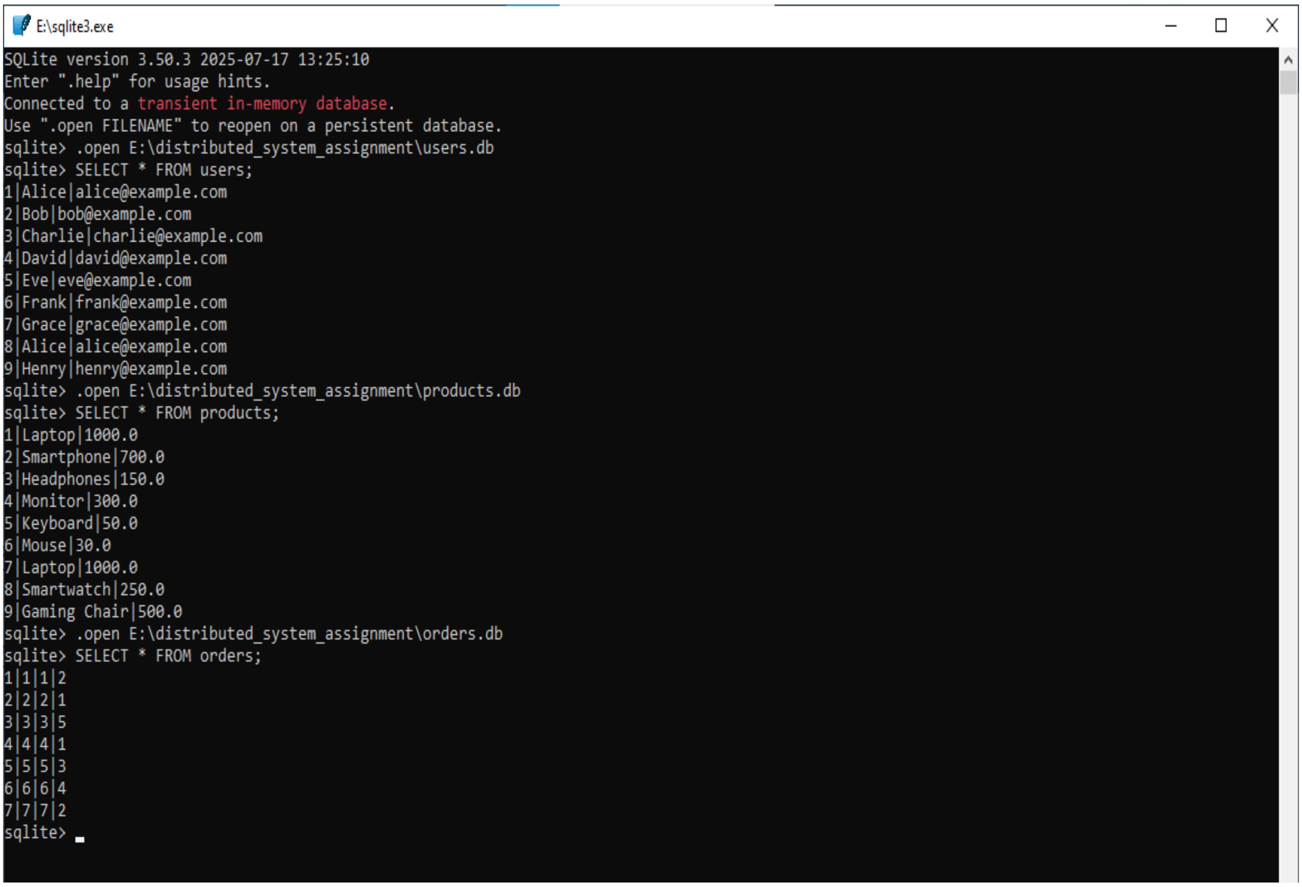
```
PRODUCTS TABLE (Inserted):
(1, 'Laptop', 1000.0)
(2, 'Smartphone', 700.0)
(3, 'Headphones', 150.0)
(4, 'Monitor', 300.0)
(5, 'Keyboard', 50.0)
(6, 'Mouse', 30.0)
(7, 'Laptop', 1000.0)
(8, 'Smartwatch', 250.0)
(9, 'Gaming Chair', 500.0)


ORDERS TABLE (Skipped):
(8, 8, 8, 0) --> Failed validation
(9, 9, 1, -1) --> Failed validation
(10, 10, 11, 2) --> Failed validation

USERS TABLE (Skipped):
(10, '', 'jane@example.com') --> Failed validation
PRODUCTS TABLE (Skipped):
(10, 'Earbuds', -50.0) --> Failed validation
```

6. Database Verification

Verification using SQLite:

```
sqlite> .open E:\distributed_system_assignment\users.db
sqlite> SELECT * FROM users;
1|Alice|alice@example.com
2|Bob|bob@example.com
3|Charlie|charlie@example.com
4|David|david@example.com
5|Eve|eve@example.com
6|Frank|frank@example.com
7|Grace|grace@example.com
8|Alice|alice@example.com
9|Henry|henry@example.com
sqlite> .open E:\distributed_system_assignment\products.db
sqlite> SELECT * FROM products;
1|Laptop|1000.0
2|Smartphone|700.0
3|Headphones|150.0
4|Monitor|300.0
5|Keyboard|50.0
6|Mouse|30.0
7|Laptop|1000.0
8|Smartwatch|250.0
9|Gaming Chair|500.0
```

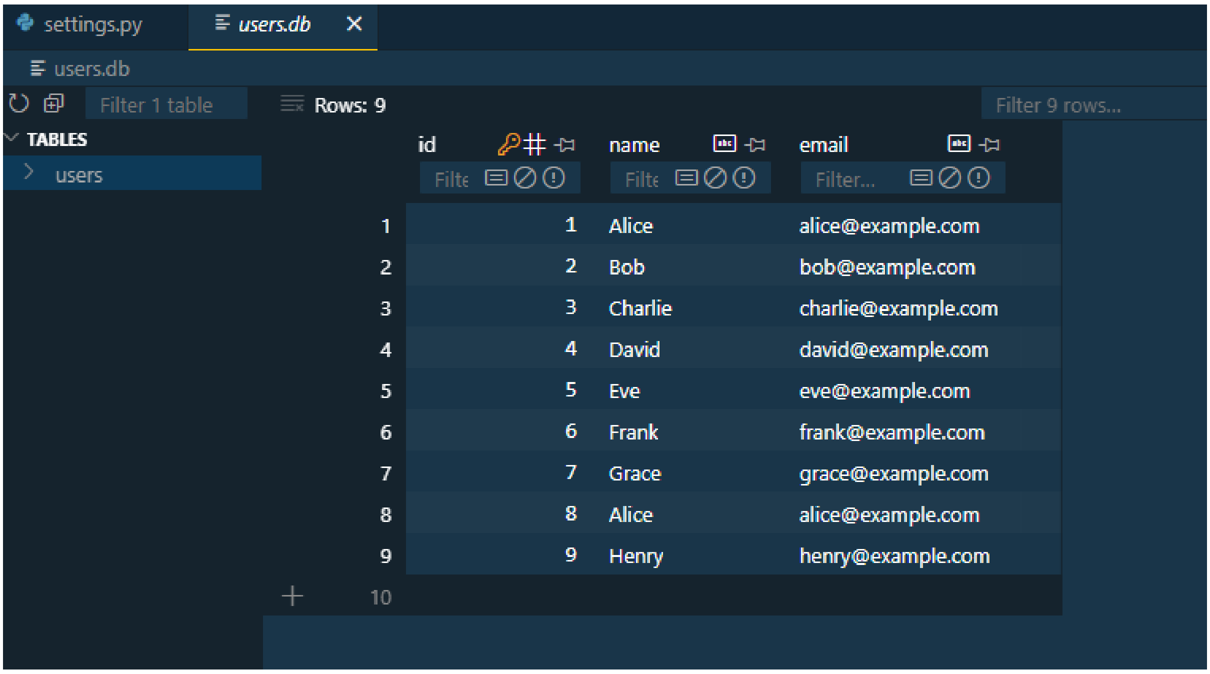
```
sqlite> .open E:\distributed_system_assignment\orders.db
sqlite> SELECT * FROM orders;
1|1|1|2
2|2|2|1
3|3|3|5
4|4|4|1
5|5|5|3
6|6|6|4
7|7|7|2
sqlite>
```



```
SQLite version 3.50.3 2025-07-17 13:25:10
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open E:\distributed_system_assignment\users.db
sqlite> SELECT * FROM users;
1|Alice|alice@example.com
2|Bob|bob@example.com
3|Charlie|charlie@example.com
4|David|david@example.com
5|Eve|eve@example.com
6|Frank|frank@example.com
7|Grace|grace@example.com
8|Alice|alice@example.com
9|Henry|henry@example.com
sqlite> .open E:\distributed_system_assignment\products.db
sqlite> SELECT * FROM products;
1|Laptop|1000.0
2|Smartphone|700.0
3|Headphones|150.0
4|Monitor|300.0
5|Keyboard|50.0
6|Mouse|30.0
7|Laptop|1000.0
8|Smartwatch|250.0
9|Gaming Chair|500.0
sqlite> .open E:\distributed_system_assignment\orders.db
sqlite> SELECT * FROM orders;
1|1|1|2
2|2|2|1
3|3|3|5
4|4|4|1
5|5|5|3
6|6|6|4
7|7|7|2
sqlite>
```

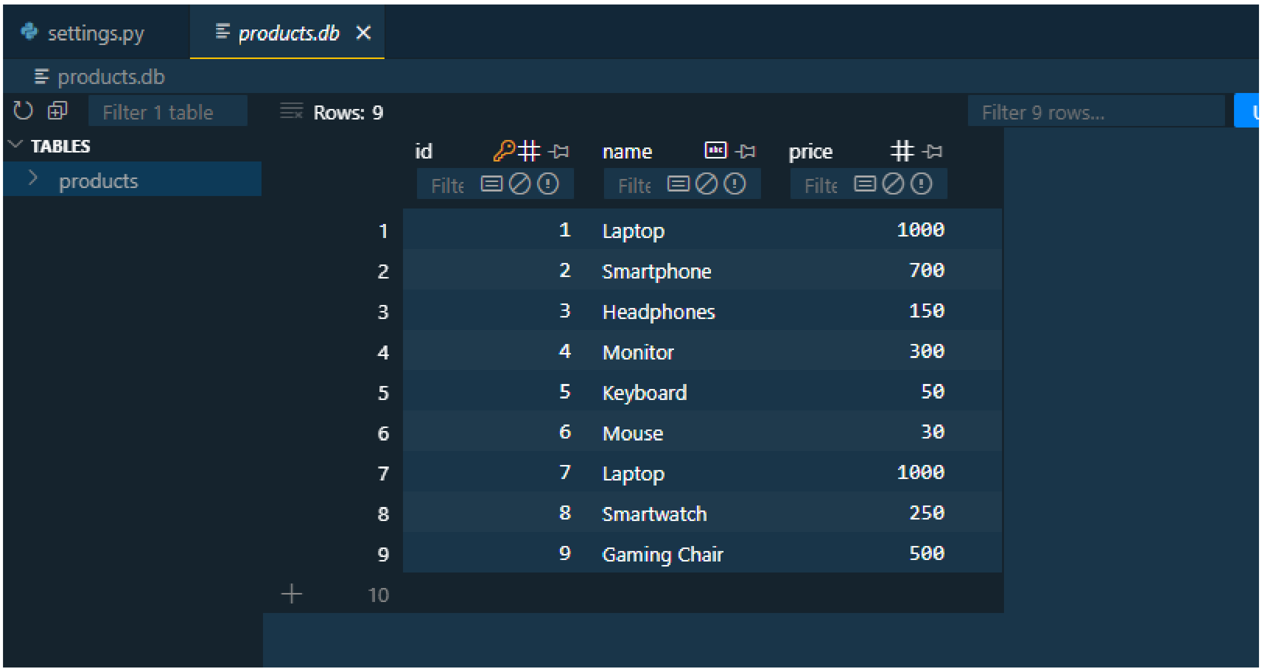
7. Screenshots

□ **Users Table** – SQLite query result



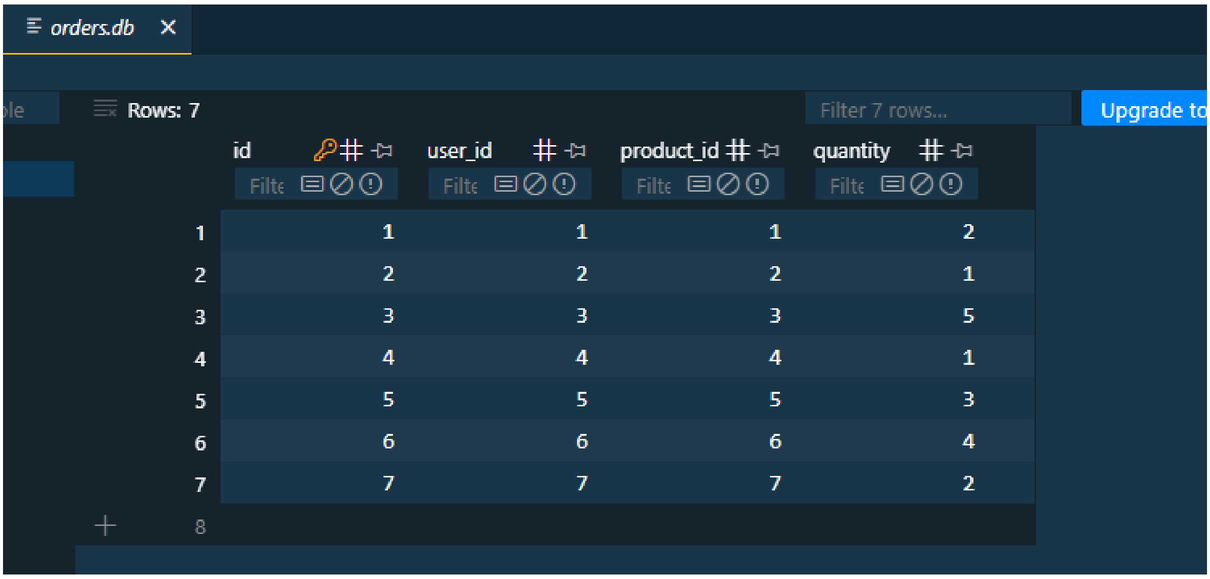
	id	name	email
1	1	Alice	alice@example.com
2	2	Bob	bob@example.com
3	3	Charlie	charlie@example.com
4	4	David	david@example.com
5	5	Eve	eve@example.com
6	6	Frank	frank@example.com
7	7	Grace	grace@example.com
8	8	Alice	alice@example.com
9	9	Henry	henry@example.com
+ 10			

□ **Products Table** – SQLite query result



	id	name	price
1	1	Laptop	1000
2	2	Smartphone	700
3	3	Headphones	150
4	4	Monitor	300
5	5	Keyboard	50
6	6	Mouse	30
7	7	Laptop	1000
8	8	Smartwatch	250
9	9	Gaming Chair	500
+ 10			

□ **Orders Table** – SQLite query result



	id	user_id	product_id	quantity
1	1	1	1	2
2	2	2	2	1
3	3	3	3	5
4	4	4	4	1
5	5	5	5	3
6	6	6	6	4
7	7	7	7	2
8				

8. Conclusion

- Implemented a Django project with **three separate SQLite databases**.
- Performed data insertion concurrently using **threads**.
- Applied **validations in Python**, ensuring only valid data is stored.
- Verified the final records using **SQLite queries**.
- Final Result: **9 users, 9 products, 7 orders (invalid records skipped as per rules)**.