Documentation for Part 4

ACID- compliant distributed transactions using PostgreSQL

Atomicity:

Atomicity ensures that a transaction is treated as a single, indivisible unit. If any part of the transaction fails, the entire transaction is rolled back, preserving the consistency of the data.

In the project code, in case of any exception during the transaction (e.g., an error in executing a query), the entire distributed transaction is rolled back using **connection_a.rollback()** and **connection_b.rollback()**. This ensures that either all changes made by the transaction are applied or none at all.

Consistency:

Consistency ensures that a transaction brings the database from one valid state to another. It preserves the integrity constraints of the data.

In the project code, the SQL queries executed on Node A and Node B ensure that the changes made to the database are consistent with the business logic. For example, updating the product quantity and inserting order details are operations that maintain the consistency of the e-commerce platform.

Isolation:

Isolation ensures that the execution of transactions is isolated from each other, preventing interference between concurrent transactions.

In the project code, by setting **autocommit** to **False** for both Node A and Node B, the transactions are explicitly managed, and their effects are not immediately visible to other transactions. This helps in achieving a degree of isolation between transactions.

Durability:

Durability ensures that once a transaction is committed, its changes are permanent and survive any subsequent failures.

In the project code, the **commit** statements finalize the distributed transaction. Once the commit is successful, the changes made by the transaction are durable, and they will persist even if there are failures or system crashes.

```
Distributed transaction committed successfully.
Table: shipments
+---------------+------------+--------------------+
|   shipment_id |   order_id | shipping_address   |
+===============+============+====================+
|             1 |          1 | 123 Main Street    |
+---------------+------------+--------------------+

Table: order_status
+------------+----------+
|   order_id | status   |
+============+==========+
|          1 | Shipped  |
+------------+----------+
```

These elements collectively contribute to achieving ACID properties in the context of distributed transactions in an e-commerce platform.

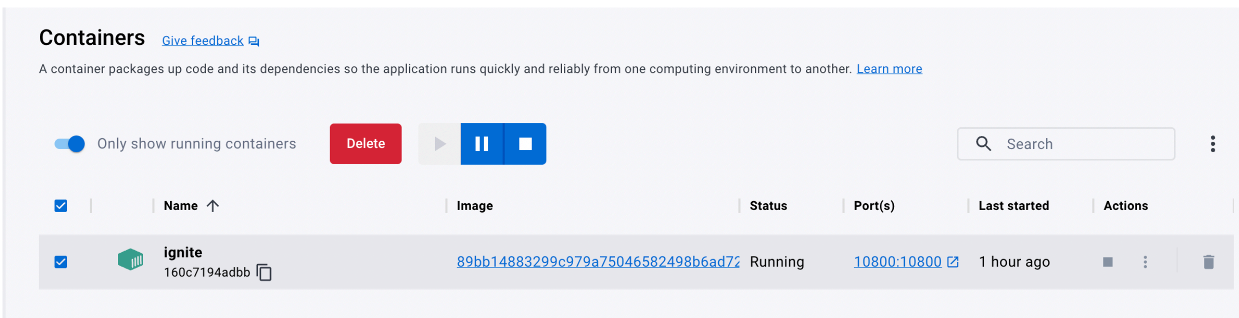
Concurrency control mechanism developed using Apache Ignite

Introduction:
This section documents the implementation of concurrency control in a distributed transaction management system using Apache Ignite. The primary goal is to manage simultaneous transactions effectively, ensuring data consistency and preventing conflicts in a concurrent environment.
System Overview:
The system utilizes Apache Ignite, a distributed database that provides high-performance, in-memory data storage and computing capabilities. Ignite supports transactional operations and offers various concurrency control mechanisms.

Pulled apache ignite image from docker container.

```
(base) priyadarshiniramakrishnan@Priyadarshinis-MacBook-Pro ~ % docker pull applem1support/ignite:2.12.0
2.12.0: Pulling from applem1support/ignite
0362ad1dd800: Pull complete
571218f61883: Pull complete
62afd1dfc9c3: Pull complete
9ea691a56377: Pull complete
86f4dd9c6161: Pull complete
0dd468c9d3bf: Pull complete
16cc616215ff: Pull complete
e0aad62c1238: Pull complete
774d49196324: Pull complete
ceded3bad7dc: Pull complete
Digest: sha256:800b125d95d384fc2a8dfe07c052e80cf9f0f210b5008ea8e29d0bd8029b459b
Status: Downloaded newer image for applem1support/ignite:2.12.0
docker.io/applem1support/ignite:2.12.0
(base) priyadarshiniramakrishnan@Priyadarshinis-MacBook-Pro ~ % docker restart ignite
ignite
(base) priyadarshiniramakrishnan@Priyadarshinis-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS                                                                  NAMES
2022dcb8e82c   applem1support/ignite:2.12.0   "/bin/sh -c $IGNITE_…"   14 minutes ago   Up 13 minutes   8080/tcp, 10800/tcp, 11211/tcp, 47100/tcp, 47500/tcp, 49112/tcp   ignite
(base) priyadarshiniramakrishnan@Priyadarshinis-MacBook-Pro ~ % docker run -d --name ignite -p 10800:10800 applem1support/ignite
Unable to find image 'applem1support/ignite:latest' locally
docker: Error response from daemon: manifest for applem1support/ignite:latest not found: manifest unknown: manifest unknown.
See 'docker run --help'.
(base) priyadarshiniramakrishnan@Priyadarshinis-MacBook-Pro ~ % docker run -d --name ignite -p 10800:10800 89bb14883299c979a75046582498b6ad72691c003fab71d10e5d756f646cb876
160c7194adbb4fe23e8e9530aab5fcb236d1fc13467e27832e58149a81ccb99d
(base) priyadarshiniramakrishnan@Priyadarshinis-MacBook-Pro ~ % brew services start postgresql@16
```

Started the apache ignite server.



<u>Concurrency Control Implementation:</u>

What is Concurrency Control?

- Concurrency control is a database management technique that ensures data integrity and consistency when multiple transactions occur at the same time.
- The primary goal of concurrency control is to manage conflicts and dependencies between concurrent transactions to prevent data anomalies and integrity issues.

Why is Concurrency Control Important?

- **Data Integrity**: Without proper concurrency control, simultaneous transactions can lead to data inconsistencies. For example, two transactions might simultaneously attempt to update the same record, leading to conflicts or incorrect data.
- **Isolation**: It ensures that transactions are isolated from each other, meaning the operations of one transaction are not visible to other transactions until they are completed.
- **System Performance**: Efficient concurrency control can improve the overall performance of a database system by managing access to data resources effectively.

Optimistic Concurrency Control approach (which is used in our implementation):

- Transactions execute without locking resources but validate the transaction at the end. If a conflict is detected, the transaction is rolled back and retried.

Code Overview:

The Python script implements concurrency control by managing simultaneous transactions on an Apache Ignite cluster. It utilizes the pyignite module to interact with Ignite.

Key Features

- **Optimistic Transactions**: The system employs optimistic concurrency control for transactions. It uses TransactionConcurrency.OPTIMISTIC and TransactionIsolation.SERIALIZABLE to ensure the highest level of isolation.
- **Conflict Handling**: In case of a conflict (detected by an OptimisticException), the system retries the transaction, providing robustness against concurrent access conflicts.
- **Threaded Transaction Execution**: The script uses Python's threading module to simulate concurrent transactions, demonstrating the system's capability to handle multiple simultaneous operations.

Functionality

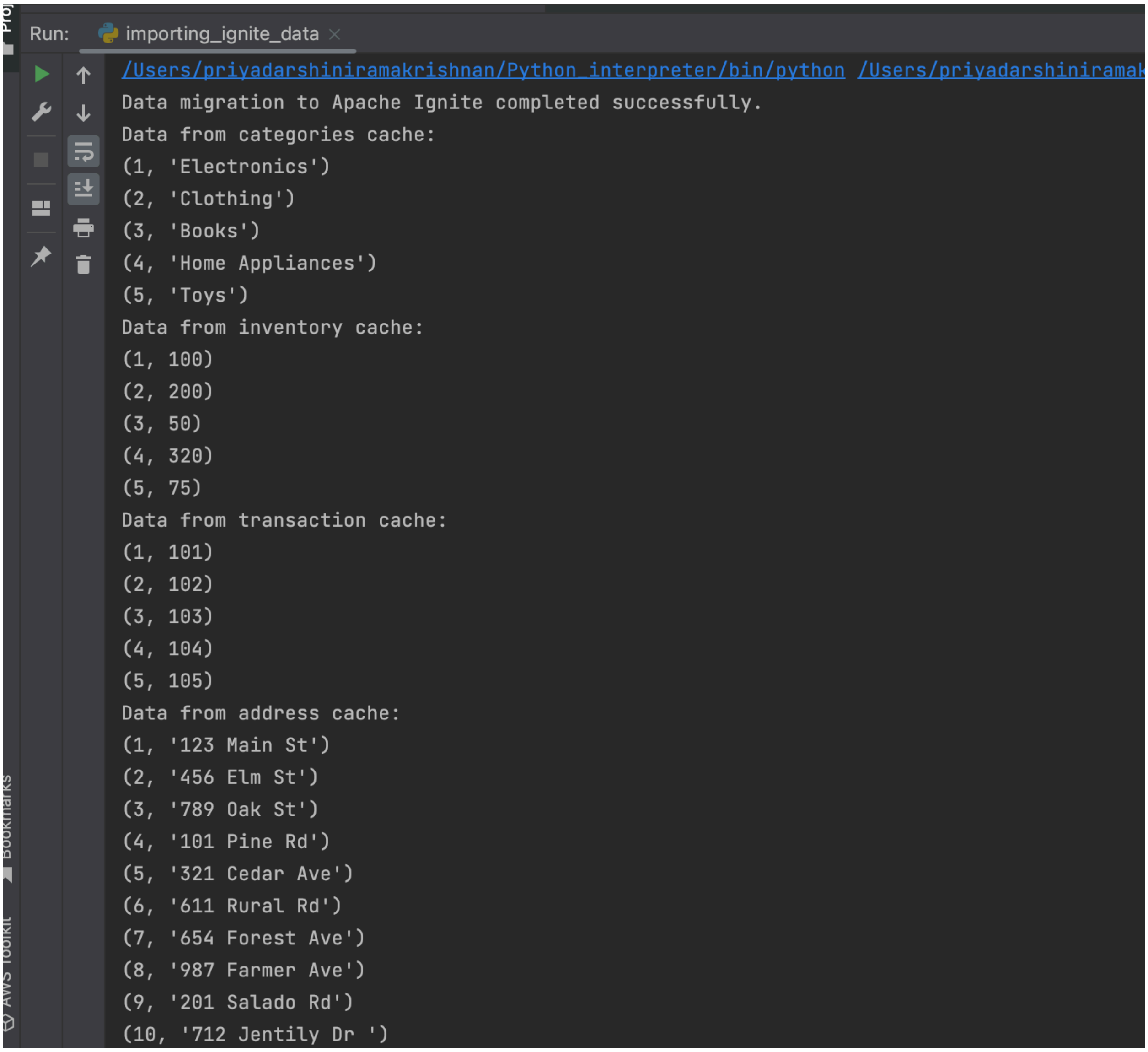
- connect_ignite **Function**: Connects to the Apache Ignite cluster.
- retrieve_data **Function**: Retrieves data from a specified cache in Ignite.
- perform_transaction **Function**: Handles the transaction logic, including conflict detection and retry mechanism. Updates the inventory data in a transactional manner.
- **Main Execution Flow**: Establishes a connection to the Ignite cluster. Performs concurrent transactions on the same inventory item to illustrate concurrency control.

Challenges and Solutions

- **Concurrency Conflicts**: A key challenge was handling conflicts arising from concurrent transactions. The solution implemented involves retrying the transaction in case of an OptimisticException.
- **Data Consistency**: Ensuring data consistency in a concurrent environment required careful design of the transaction logic and appropriate use of Ignite's transactional features.

Results and Discussion

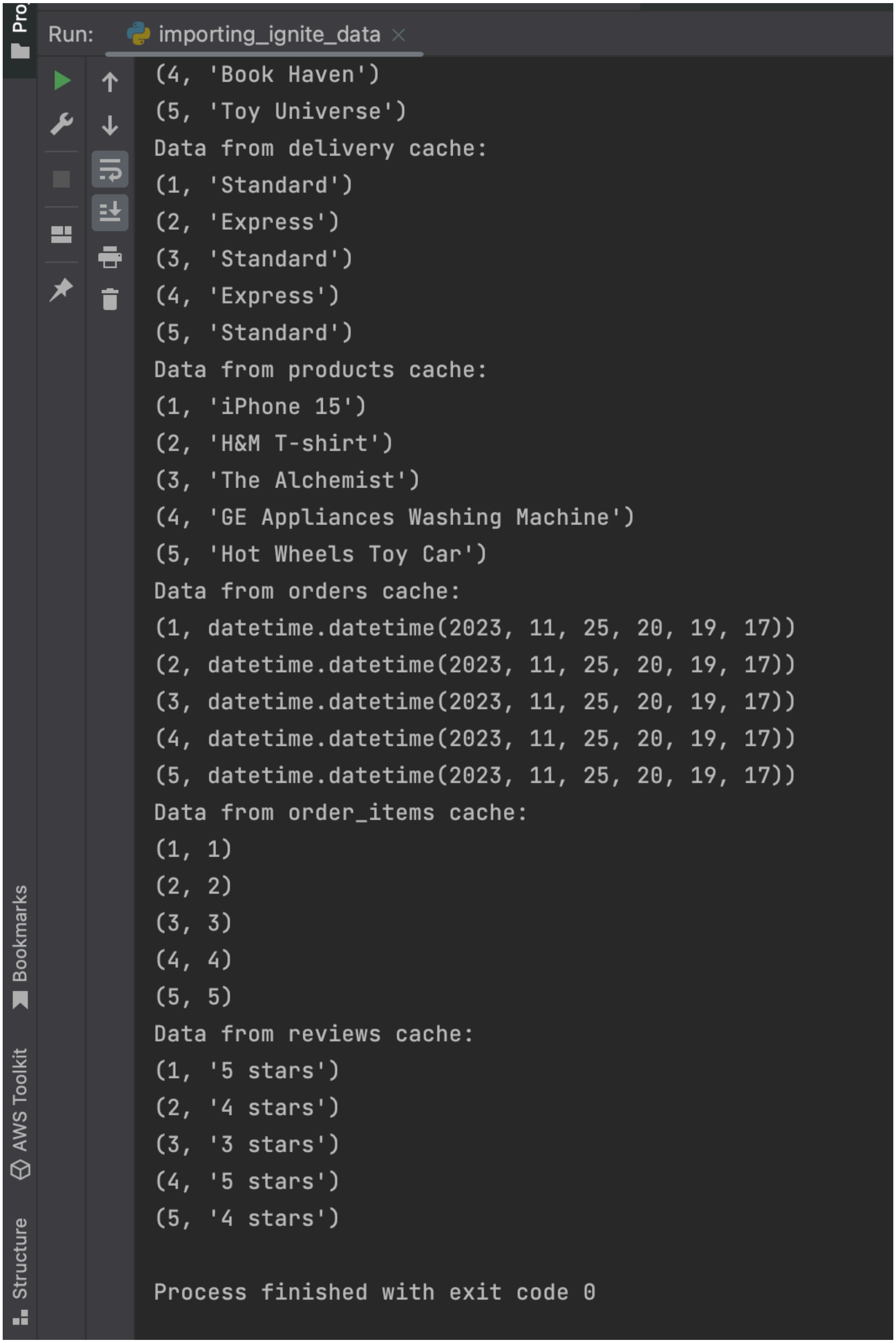
The implementation successfully demonstrates handling of concurrent transactions in Apache Ignite. The use of optimistic concurrency control, along with serializable isolation, ensures that transactions are executed reliably without data inconsistencies.
Data being added to apache ignite caches.



Data retrieved from caches.

Run: importing_ignite_data

```
Data from contact_details cache:
(1, '123-456-7890')
(2, '555-555-5555')
(3, '987-654-3210')
(4, '111-222-3333')
(5, '777-888-9999')
(6, '113-466-7880')
(7, '565-565-5665')
(8, '917-454-3110')
(9, '121-422-1353')
(10, '747-818-9397')
Data from customer cache:
(1, 'John')
(2, 'Jane')
(3, 'James')
(4, 'Emily')
(5, 'Michael')
Data from suppliers cache:
(1, 'TechCo')
(2, 'Fashion World')
(3, 'Furniture Emporium')
(4, 'Book Haven')
(5, 'Toy Universe')
Data from delivery cache:
(1, 'Standard')
(2, 'Express')
(3, 'Standard')
(4, 'Express')
(5, 'Standard')
```

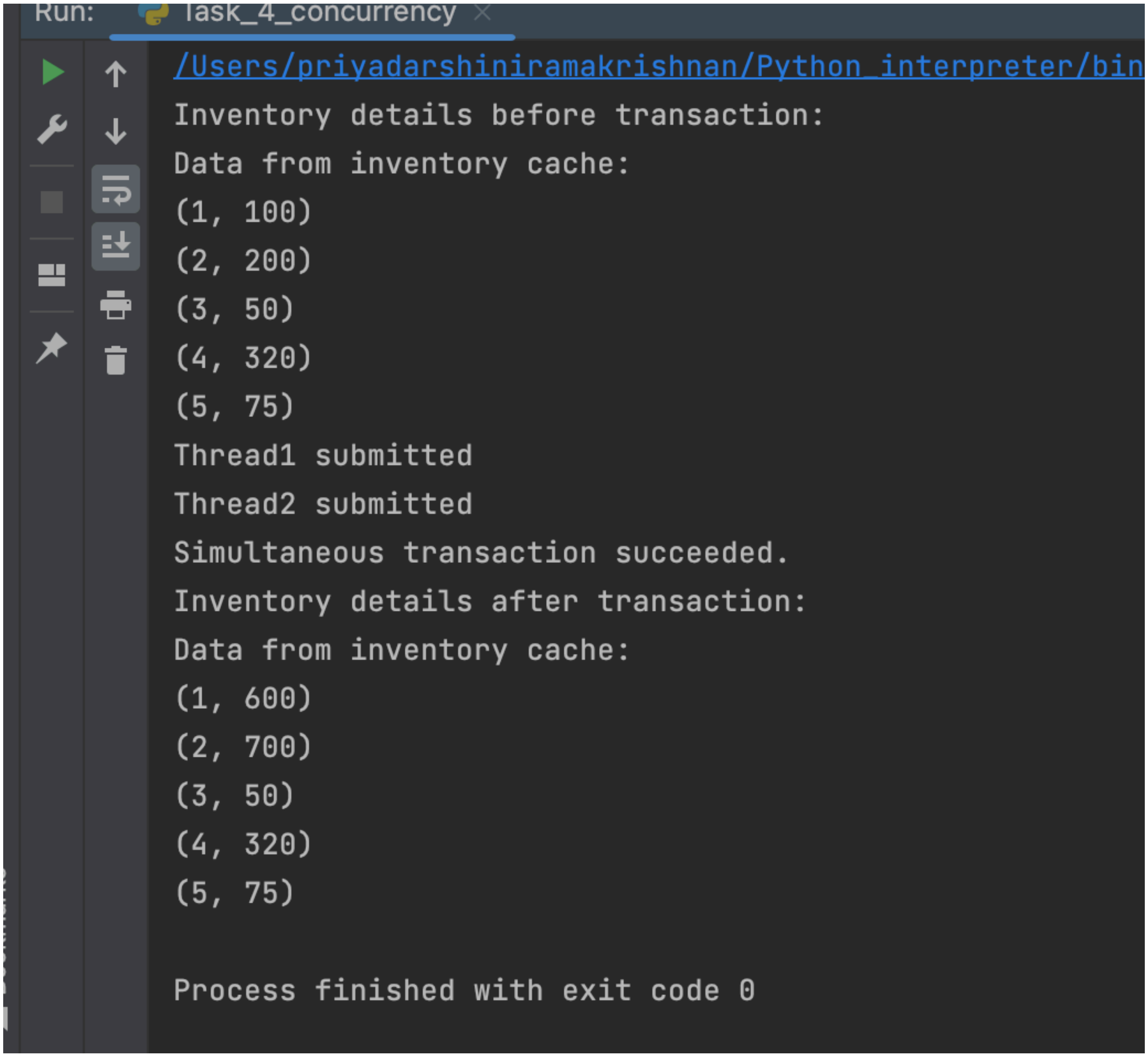



```
(4, 'Book Haven')
(5, 'Toy Universe')
Data from delivery cache:
(1, 'Standard')
(2, 'Express')
(3, 'Standard')
(4, 'Express')
(5, 'Standard')
Data from products cache:
(1, 'iPhone 15')
(2, 'H&M T-shirt')
(3, 'The Alchemist')
(4, 'GE Appliances Washing Machine')
(5, 'Hot Wheels Toy Car')
Data from orders cache:
(1, datetime.datetime(2023, 11, 25, 20, 19, 17))
(2, datetime.datetime(2023, 11, 25, 20, 19, 17))
(3, datetime.datetime(2023, 11, 25, 20, 19, 17))
(4, datetime.datetime(2023, 11, 25, 20, 19, 17))
(5, datetime.datetime(2023, 11, 25, 20, 19, 17))
Data from order_items cache:
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(5, 5)
Data from reviews cache:
(1, '5 stars')
(2, '4 stars')
(3, '3 stars')
(4, '5 stars')
(5, '4 stars')

Process finished with exit code 0
```

Created 2 threads and submitted parallel without affecting one other , which shows concurrency control has been achieved .

Run: Task_4_concurrency

```
/Users/priyadarshiniramakrishnan/Python_interpreter/bi
Inventory details before transaction:
Data from inventory cache:
(1, 100)
(2, 200)
(3, 50)
(4, 320)
(5, 75)
Thread1 submitted
Thread2 submitted
Simultaneous transaction succeeded.
Inventory details after transaction:
Data from inventory cache:
(1, 1100)
(2, 200)
(3, 50)
(4, 320)
(5, 75)

Process finished with exit code 0
```



Conclusion:

The system effectively showcases the use of Apache Ignite for concurrency control in a distributed transaction management scenario. The implementation highlights the capabilities of Ignite in handling complex transactional operations in a concurrent environment.