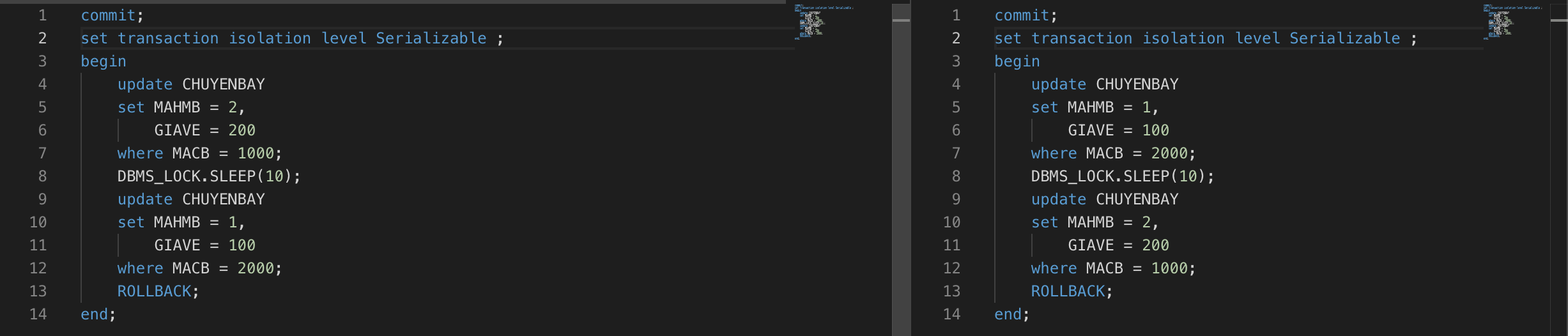
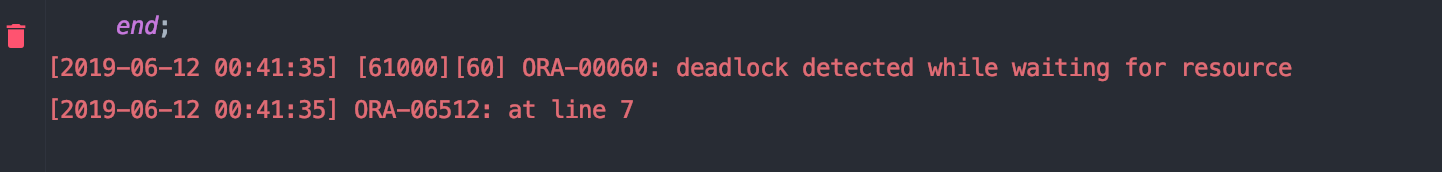
**5.5. Deadlock**

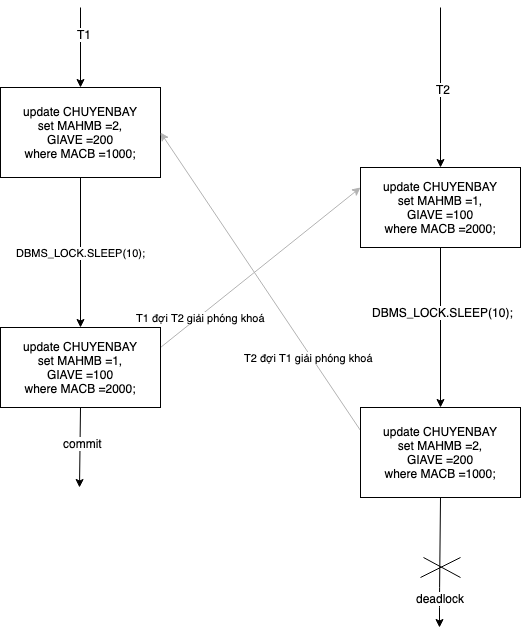
* *Tình huống*:

*+* T1 xem cập nhật chuyến bay ‘1000’ sau đó cập nhật chuyến bay ‘2000’.

+ T2 thực hiện cập nhật lần lượt chuyến bay ‘2000’ và ‘1000’.

+ Điều này khiến cho T1 chờ tài nguyên của T2, còn T2 chờ tài nguyên của T1 → Xảy ra deadlock.

* *Nguyên nhân*:



- *Hướng giải quyết*: Để giải quyết deadlock, hệ quản trị sẽ tự động rollback transaction có kích cỡ nhỏ hơn. Kích cỡ của 1 transaction được xác định bởi số dòng insert, update or delete. Trong trường hợp này, 2 transaction có kích cỡ bằng nhau nên hệ quản trị sẽ rollback transaction T2 thực hiện sau. Để ngăn ngừa deadlock, ta giảm mức cô lập xuống Repeatable Read.

- *Kết quả*: Danh sách vé sẽ không tính việc bán vé ở T2. Vì lúc này T1 dùng dữ liệu trước khi T2 thực thi.

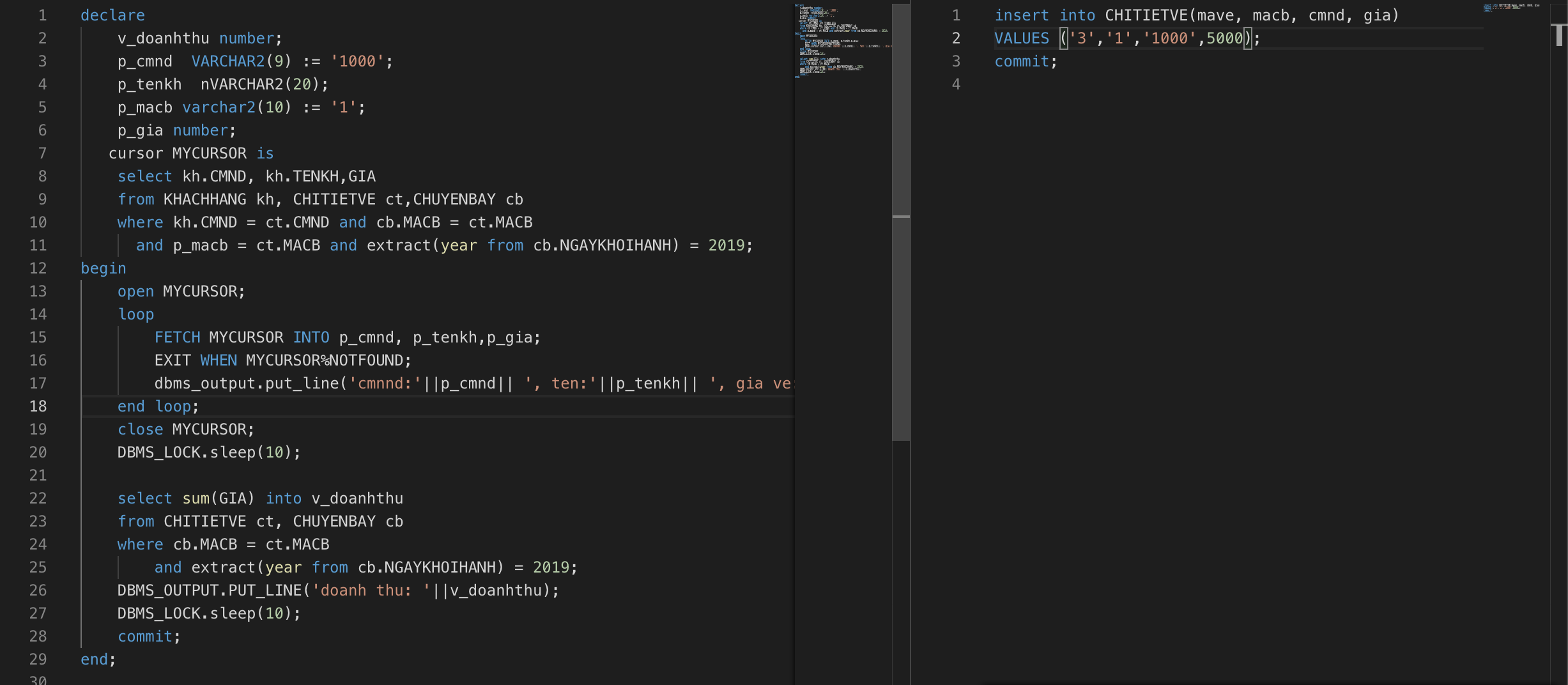
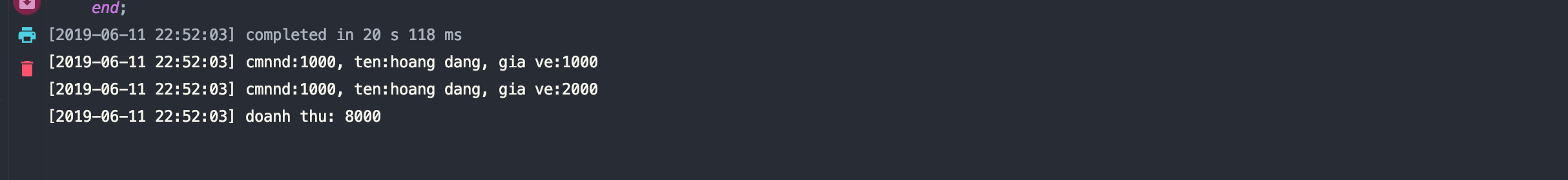
**5.6. Phantom**

- *Tình huống*:

+ T1 xem danh sách vé và tính doanh thu.

+ T2 thực hiện đặt vé (thêm vé mới).

+ Điều này làm T1 nhìn thấy dữ liệu bị sai. -> Xảy ra hiện tượng Phantom.

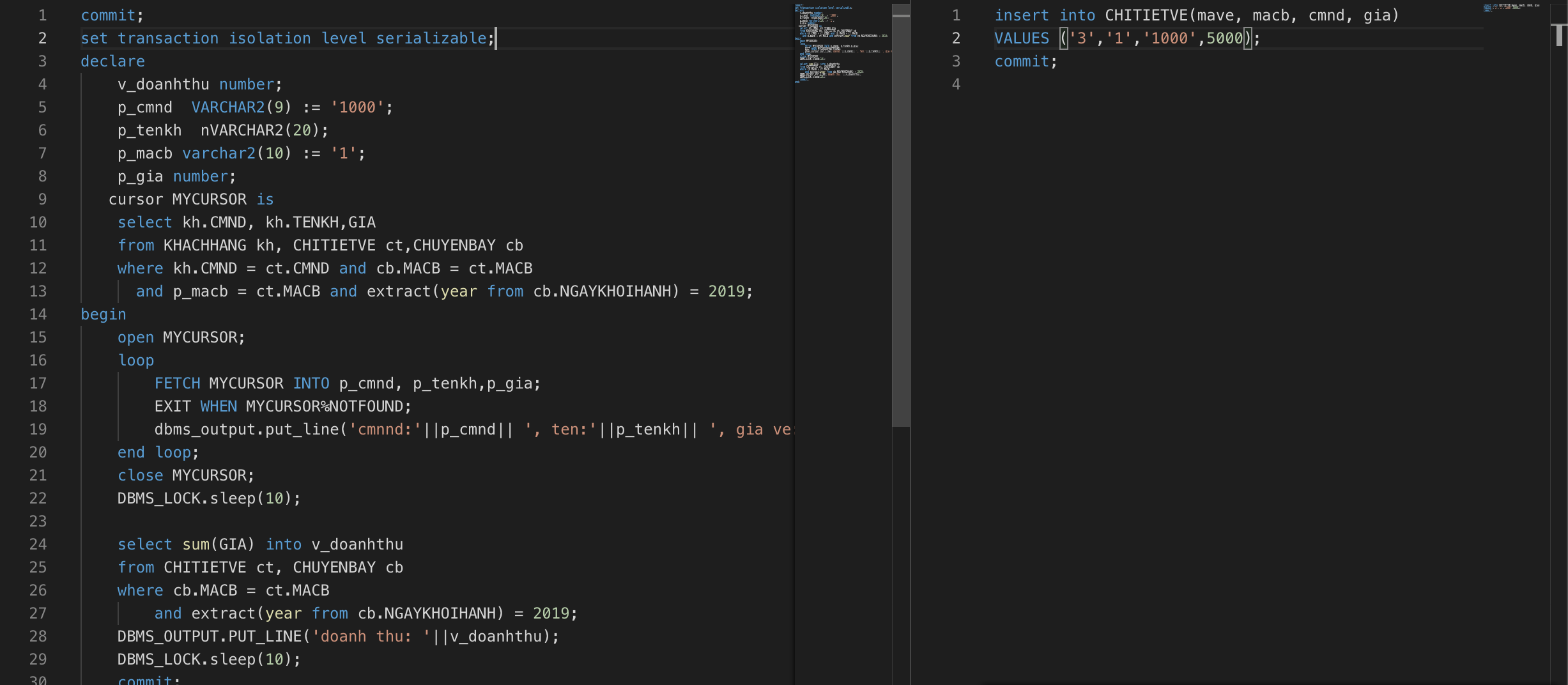
­

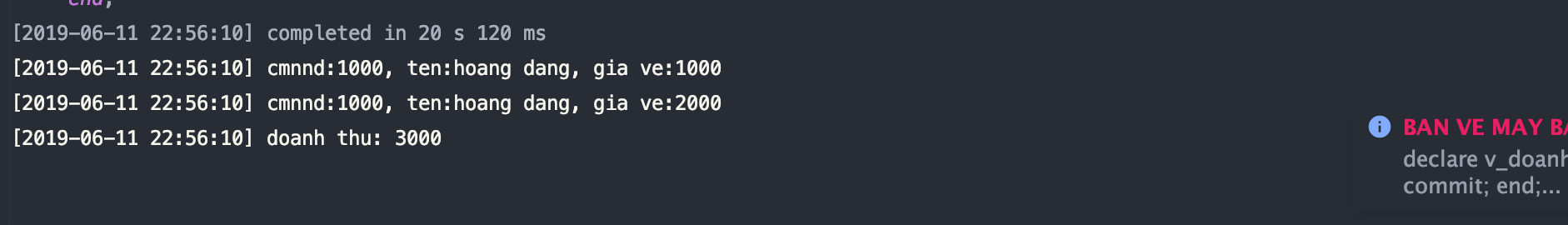
Tại đây phải có 3 dòng vé, tuy nhiên T1 chỉ thấy có 2 dòng vé và tổng doanh thu bị sai.

- *Nguyên nhân*: T1 thiết lập mức cô lập Read Committed nên khi tính tổng doanh thu thì sẽ tính cả số tiền vừa thu được từ việc bán thức ăn ở T2.

- *Hướng giải quyết*: Dùng mức cô lập Serializable. T2 phải chờ đến khi T1 thực hiện xong rồi mới thực hiện.

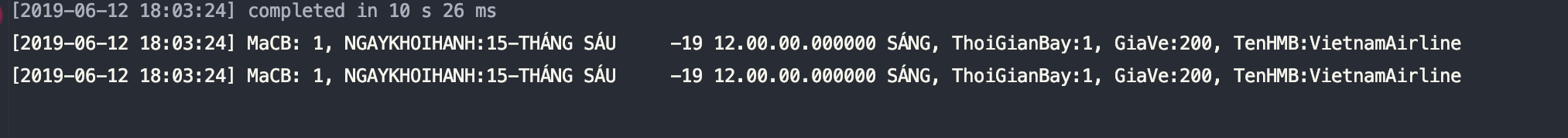
- *Kết quả*: Danh sách vé sẽ không tính việc bán vé ở T2. Vì lúc này T1 dùng dữ liệu trước khi T2 thực thi.

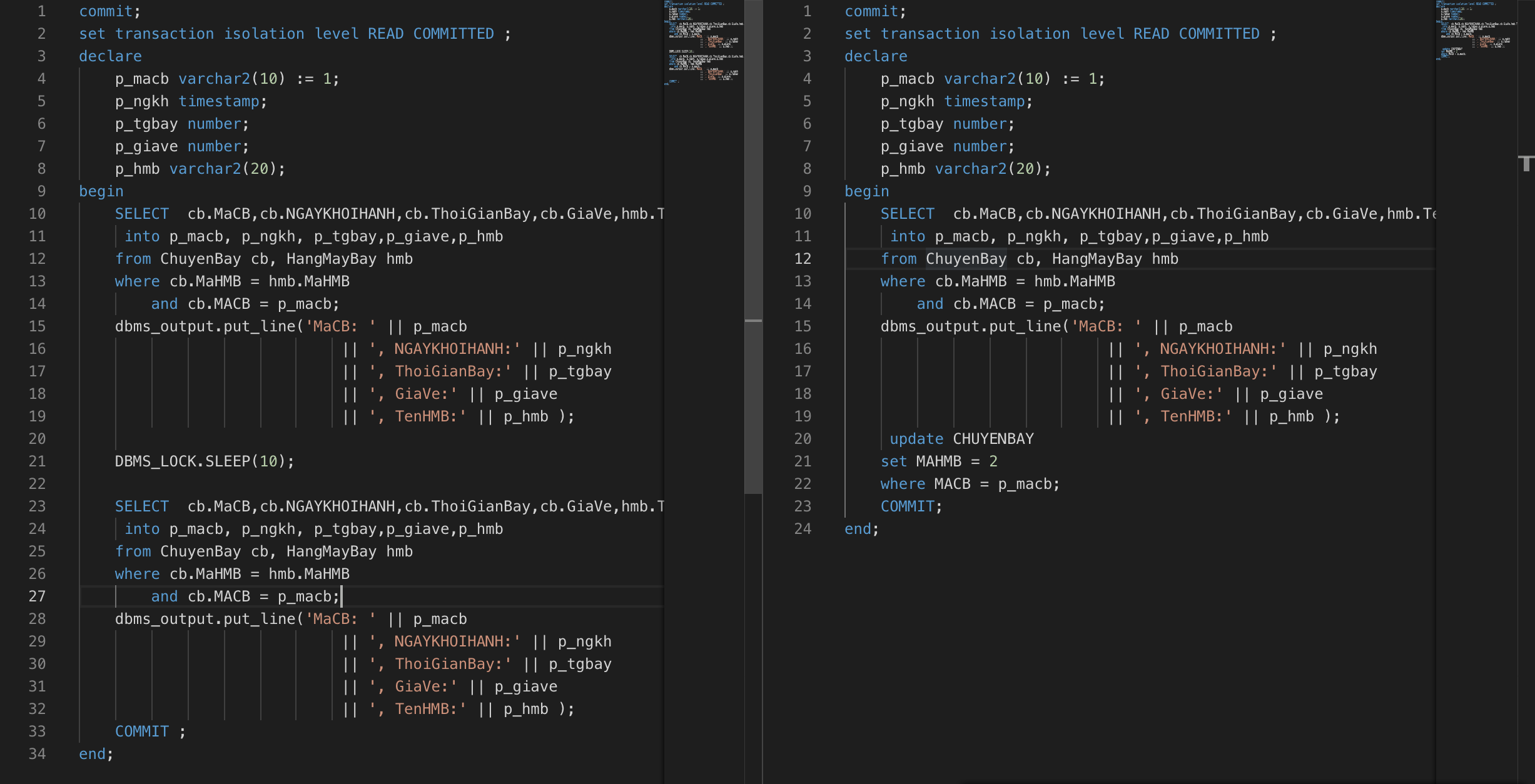




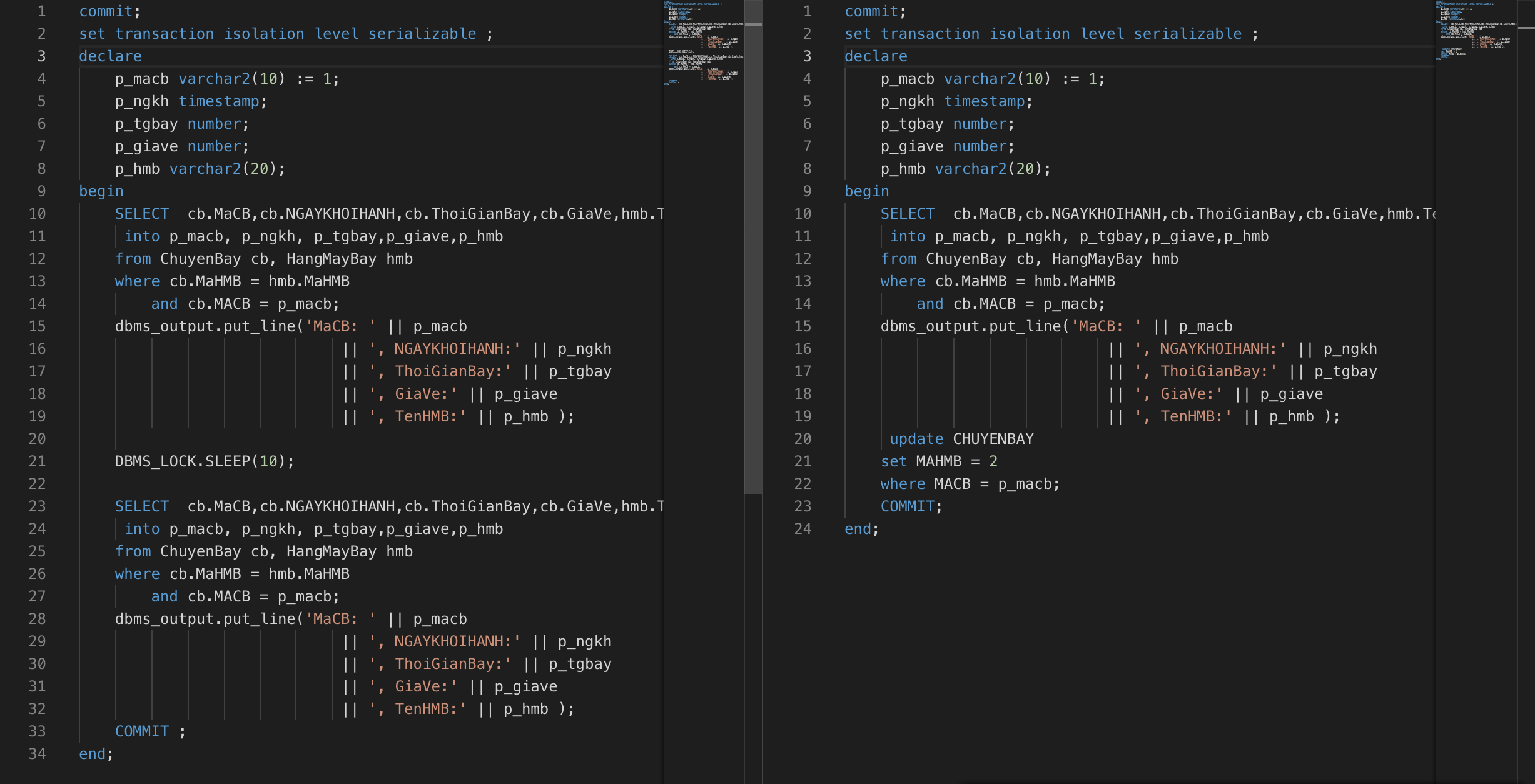
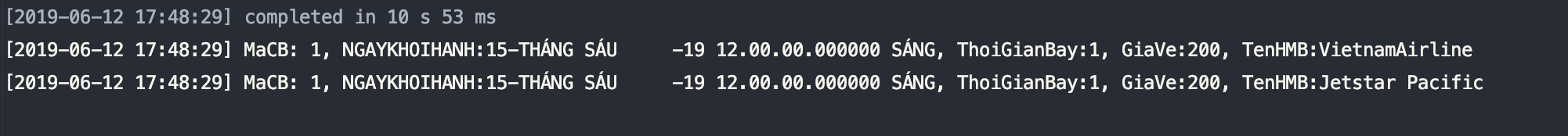
**5.7. Unrepeatable Read**

* *Tình huống:* Khi thay đổi hãng máy bay của một chuyến bay. Ban đầu T1 đọc thông tin chuyến bay, sau đó delay. T2 vào thay đổi thông tin chuyến bay, sau đó commit lên CSDL. T1 trở lại không thể đọc lại dữ liệu nhưng không thể nhìn thấy dữ liệu ban đầu.

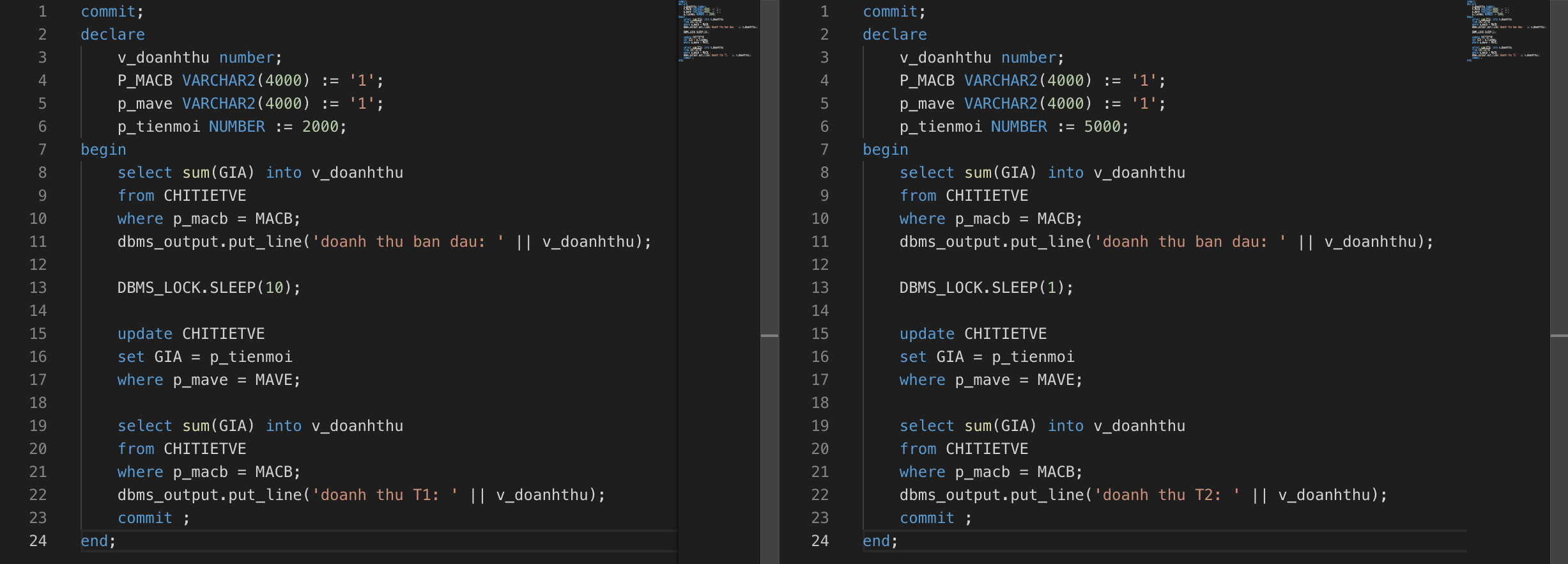
**

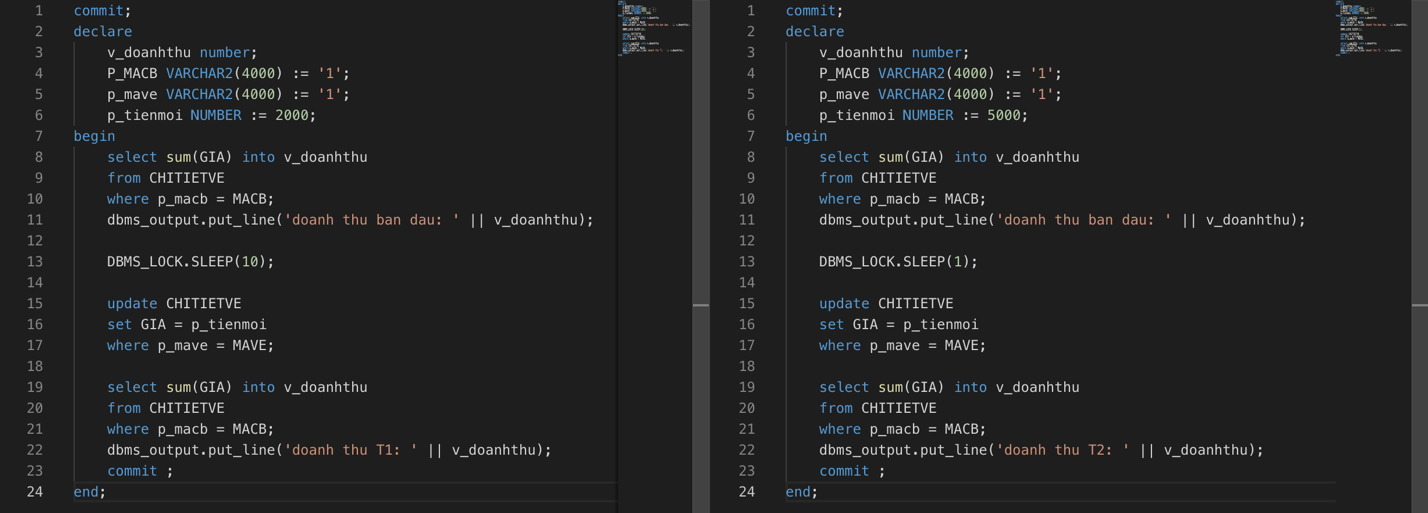


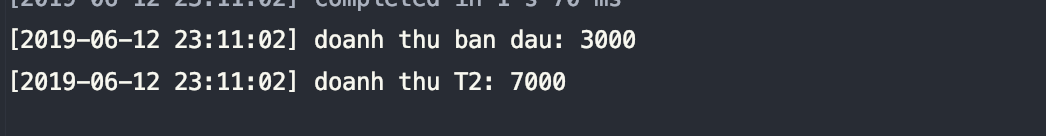
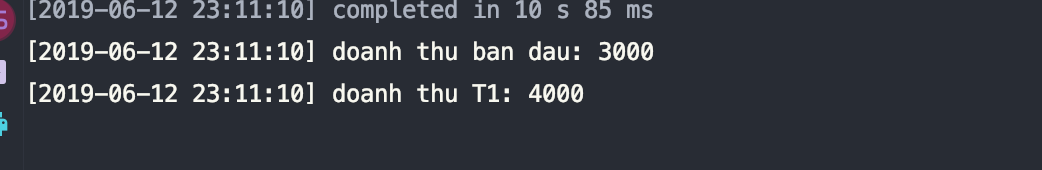
* *Nguyên nhân*: T1 thiết lập mức cô lập Read Committed nên mỗi câu lệnh SELECT trong cùng 1 thao tác sẽ thiết lập snapshot của chính nó dù những câu lệnh này đọc trên những đơn vị dữ liệu giống nhau.
* *Hướng giải quyết*: Dùng mức cô lập Serializable (hoặc Repeatable Read). Đối với Serializable, khi chạy dòng lệnh Select sẽ tạo Share lock và giữ Share lock này đến hết giao tác. Khi T2 vào Update thông tin. T2 phải đợi T1 thực hiện xong mới update. → Dữ liệu đúng.
* *Kết quả:*



**5.8. Lost Update**

* *Tình huống:* Khi cập nhật tổng giá tiền của 1 vé, T1 cập nhật lên 2000. T2 cập nhật lên 5000. Giả sử T1 bị delay 10s. Thì T2 thực hiện update trước. sau đó T1 lấy lại giá trị cũ để cập nhật. Trong trường hợp này. Cả hai đều tăng giá tiền. Nhưng sau khi cập nhật, dẫn đến kết quả tổng doanh thu sai. Mất dữ liệu cập nhật của T2.



* *Nguyên nhân*: T1 và T2 đọc giá trị doanh thu ban đầu. Tuy nhiên T1 không nhận biết được T2 đã cập nhật doanh thu mới, trong khi T1 vẫn sử dụng giá trị doanh thu đọc ban đầu. Vì vậy dữ liệu doanh thu đã bị cập nhật sai.
* *Hướng giải quyết*:

+ Cách 1: Sử dụng câu lệnh SELECT … FOR UPDATE thay cho câu lệnh SELECT thông thường để lấy giả trị doanh thu ban đầu. Kết quả: T1 đợi T2 thực hiện xong rồi mới vào đọc giá trị doanh thu, lúc này giá trị doanh thu chính là giá trị đã được T2 cập nhật. Sau đó mới tiến hành cập nhật doanh thu mới nên dữ liệu của T2 không bị mất. Nguyên nhân là vì câu lệnh SELECT … FOR UPDATE sẽ thiết lập khóa ghi nên nó không cho phép bất kỳ transaction nào khác vào đọc và thao tác dữ liệu trên nó, đảm bảo tính nhất quán dữ liệu.

+ Cách 2: Dùng mức cô lập Serializable. Khi T2 chạy dòng lệnh SELECT sẽ tạo Share lock và giữ Share lock này đến hết giao tác. T1 muốn thực hiện câu lệnh SELECT đầu tiên phải đợi cho đến khi T2 thực hiện xong. Kết quả: không bị mất dữ liệu cập nhật của T2, dữ liệu đúng.

**5.9. Dirty Read**

* *Tình huống:*

|  |  |  |
| --- | --- | --- |
| T1 | T2 | Giải thích |
| SELECT MaCB, GiaVe FROM ChuyenBay WHERE MaCB IN (‘100’, ‘200’, ‘300’);   |  |  | | --- | --- | | ChuyenBay | | | MaCB | GiaVe | | 100 | 1200000 | | 200 | 1000000 | |  | T1 thực hiện câu truy vấn cho chuyến bay ‘100’, ‘200’, ‘300’. Không tìm thấy chuyến bay ‘300’. |
| INSERT INTO ChuyenBay (MaCB, GiaVe) VALUES (300, 1300000); |  | T1 thêm 1 dòng dữ liệu vào bảng ChuyenBay, nhưng chưa commit. |
|  | SELECT MaCB, GiaVe FROM ChuyenBay WHERE MaCB IN (‘100’, ‘200’, ‘300’);   |  |  | | --- | --- | | ChuyenBay | | | MaCB | GiaVe | | 100 | 1200000 | | 200 | 1000000 | | 300 | 1300000 | | T2 thực hiện câu truy vấn cho chuyến bay ‘100’, ‘200’, ‘300’. |
|  | UPDATE ChuyenBay  SET GiaVe=990000  WHERE MaCB=’100’; | T2 cập nhật giá vé của chuyến bay ‘100’. |
| ROLLBACK; |  | T1 hủy giao tác. |
|  | COMMIT; | T2 commit dữ liệu. |
|  | SELECT MaCB, GiaVe FROM ChuyenBay WHERE MaCB IN (‘100’, ‘200’, ‘300’);   |  |  | | --- | --- | | ChuyenBay | | | MaCB | GiaVe | | 100 | 990000 | | 200 | 1000000 | | T2 thực hiện lại truy vấn cho chuyến bay ‘100’, ‘200’, ‘300’, nhưng không tìm thấy ‘300’. Dữ liệu ban đầu T2 đọc chưa được commit mà bị hủy bỏ nên ra kết quả dữ liệu không chính xác. |

* *Nguyên nhân*: Khi T1 cập nhật dữ liệu, T2 có thể đọc được dữ liệu vừa cập nhật của T1, tuy nhiên dữ liệu đó chưa được T1 commit (chế độ Read Uncommitted). Sau đó T1 hủy giao tác (rollback) nên dữ liệu T2 đọc được là vô nghĩa (dữ liệu rác). -> Xảy ra hiện tượng Dirty Read.
* *Hướng giải quyết*: Sử dụng mức cô lập Read Committed, Repeatable Read hoặc Serializable để giải quyết vấn đề này.
* *Kết quả:*

|  |  |  |
| --- | --- | --- |
| T1 | T2 | Giải thích |
| SELECT MaCB, GiaVe FROM ChuyenBay WHERE MaCB IN (‘100’, ‘200’, ‘300’);   |  |  | | --- | --- | | ChuyenBay | | | MaCB | GiaVe | | 100 | 1200000 | | 200 | 1000000 | |  | T1 thực hiện câu truy vấn cho chuyến bay ‘100’, ‘200’, ‘300’. Không tìm thấy chuyến bay ‘300’. |
| INSERT INTO ChuyenBay (MaCB, GiaVe) VALUES (300, 1300000); |  | T1 thêm 1 dòng dữ liệu vào bảng ChuyenBay, nhưng chưa commit. |
|  | SET TRANSACTION ISOLATION LEVEL READ COMMITTED;  --SET TRANSACTION ISOLATION LEVEL REPEATABLE READ/ SERIALIZABLE;  SELECT MaCB, GiaVe FROM ChuyenBay WHERE MaCB IN (‘100’, ‘200’, ‘300’);   |  |  | | --- | --- | | ChuyenBay | | | MaCB | GiaVe | | 100 | 1200000 | | 200 | 1000000 | | T2 đặt mức cô lập Read Committed (hoặc Repeatable Read hoặc Serializable).  T2 thực hiện câu truy vấn cho chuyến bay ‘100’, ‘200’, ‘300’, không tìm thấy chuyến bay ‘300’ vì dữ liệu chưa được T1 commit. |

* Tuy nhiên, Oracle sử dụng cơ chế Read Committed làm cơ chế mặc định (default) nên đã giải quyết được vấn đề Dirty Read.