資料庫管理 HW04

B12508026 戴偉璿

November 16, 2025

1. To check if PostgreSQL can avoid dirty read, I design two transactions:

   - Transaction A: Update balance to 999 of account_id 1

```sql
begin;
update accounts set balance = 999 where account_id = 1;
commit;
```

   - Transaction B: Read the record.

```sql
begin; select * from accounts where account_id = 1; commit;


```

   The execution steps are as follows:

   (a) Transaction A begins.

   (b) Transaction A updates balance to 999 of account_id 1, but does not commit yet.

   (c) Transaction B begins.

   (d) Transaction B reads the record of account_id 1.

   (e) Transaction B gets the old balance (not 999), which means dirty read is avoided.

   (f) Transaction B commits.

   (g) Transaction A commits.

   (h) Transaction B begins.

   (i) Transaction B reads the record of account_id 1.

   (j) Transaction B gets the new balance (999) after Transaction A commits.

   (k) Transaction B commits.

   Following are the screenshots of each step. Left panel shows Transaction A, right panel shows Transaction B. Figure 1 shows the original status of the accounts table, we can see the balance of account_id 1 is 1000. Figure 2 shows Transaction A updates balance to

999 of account_id 1 but does not commit yet, so that Transaction B still reads the old balance (1000). Figure 3 shows Transaction A commits, and then Transaction B reads the new balance (999) of account_id 1. You can determine the execution order by the system time shown in the top of each figure.

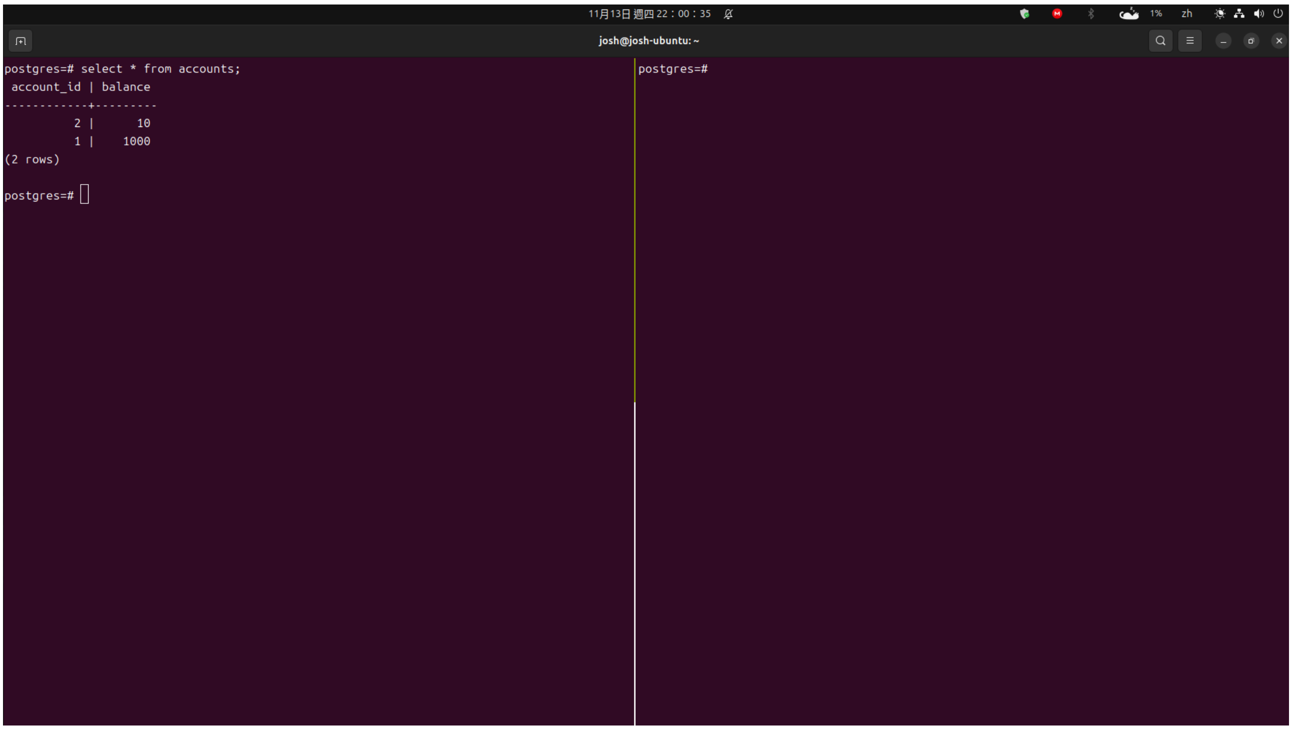

Figure 1: Orginal status of the accounts table

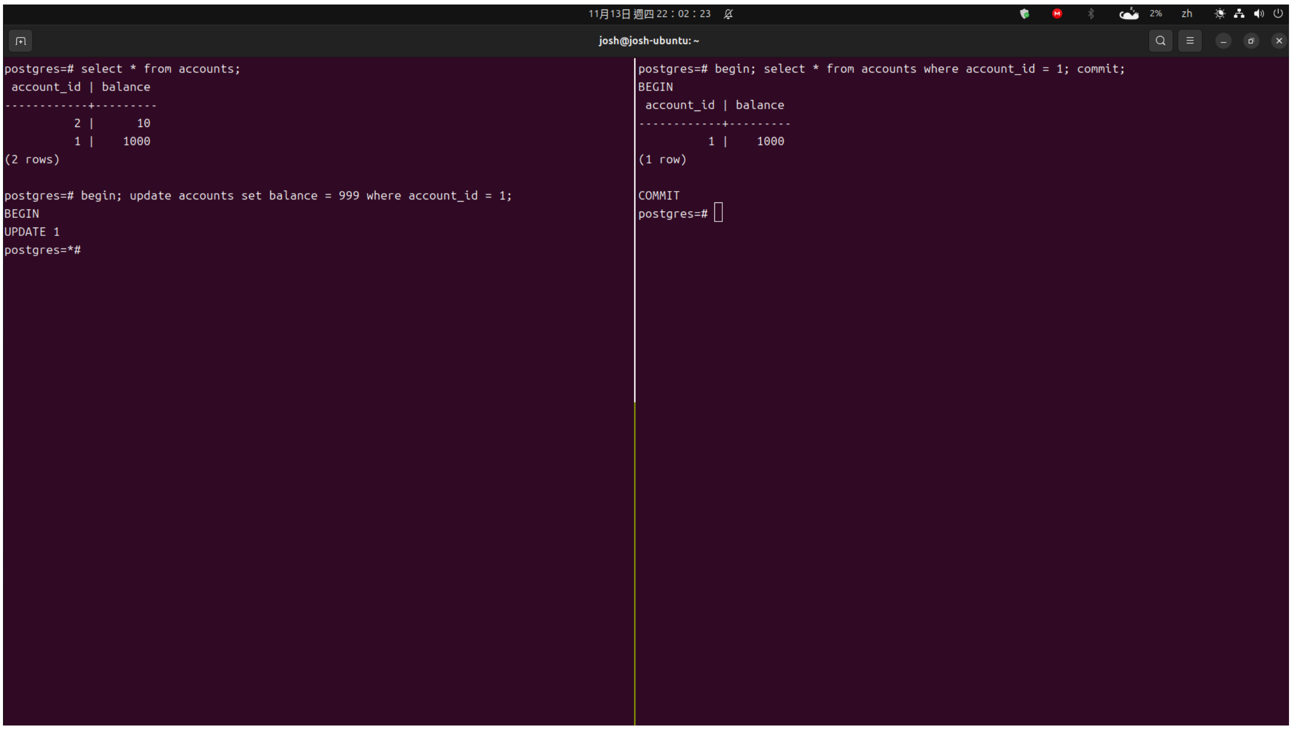

Figure 2: Transaction A updates balance to 999 of account_id 1, but does not commit yet

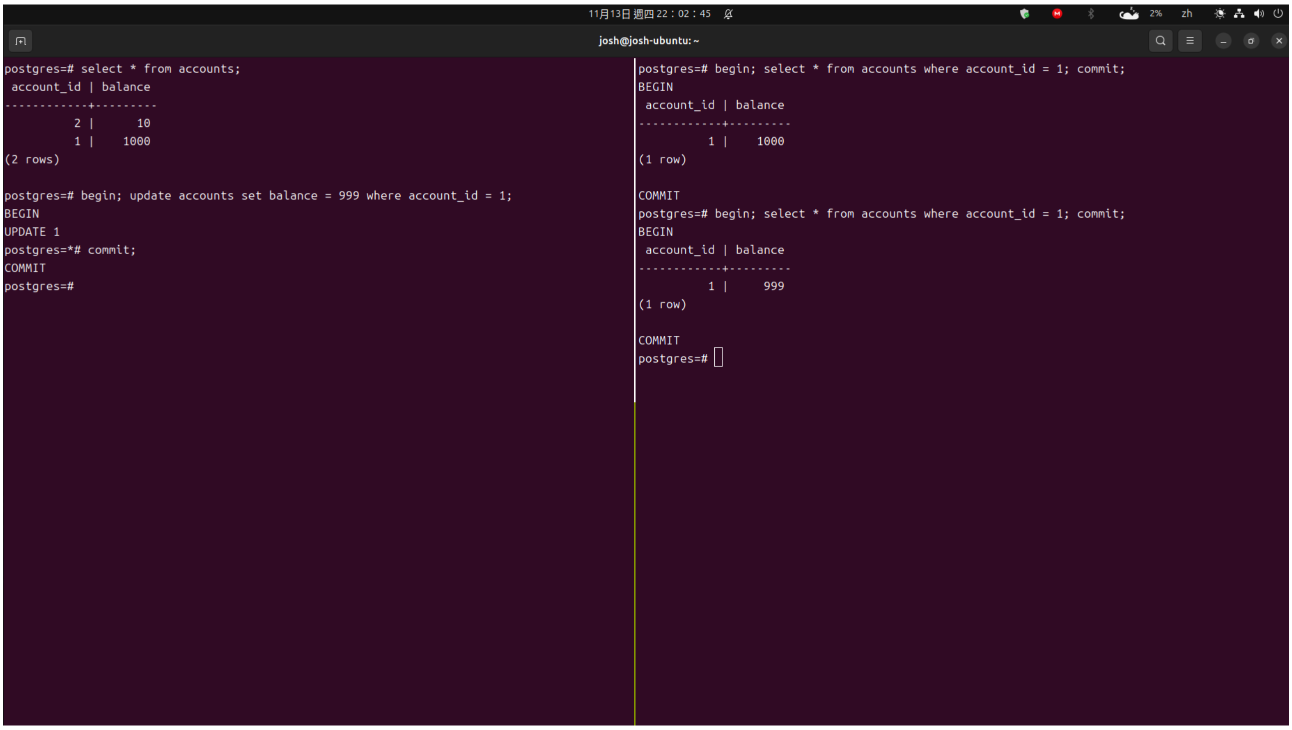

Figure 3: Transaction A commits, Transaction B reads the new balance (999) of account_id 1

With the experiments above, we can see that PostgreSQL can avoid dirty read.

2. (a) A conflict occurs when two transactions access the same data item and at least one of the acceses is a write operation. For item X, three conflicts occurs between $\{O_{11}, O_{23}\}$, $\{O_{17}, O_{21}\}$, $\{O_{17}, O_{23}\}$; for itme Y, there are no conflicts; for item Z, two conflicts occurs between $\{O_{15}, O_{24}\}$, $\{O_{15}, O_{26}\}$.

   (b) The serial schedule of $T_2 \to T_1$ is as follows:

$$O_{21} \to O_{23} \to O_{24} \to O_{26} \to O_{11} \to O_{12} \to O_{13} \to O_{15} \to O_{17}.$$

   To analyze the conflicting operation pairs, we can discuss by data items:

   For item X, the operation order must obey: $O_{21} \prec O_{23} \prec O_{17}$, $O_{23} \prec O_{11} \prec O_{17}$

   For item Y, there are no conflicts, so there is no constraint.

   For item Z, the operation order must obey: $O_{24} \prec O_{26} \prec O_{15}$

   Therefore, one such conflict-equivalent non-serial schedule is:

$$O_{21} \to O_{23} \to O_{11} \to O_{24} \to O_{26} \to O_{12} \to O_{13} \to O_{15} \to O_{17}.$$

   (c) The serial schedule of $T_1 \to T_2$ is as follows:

$$O_{11} \to O_{12} \to O_{13} \to O_{15} \to O_{17} \to O_{21} \to O_{23} \to O_{24} \to O_{26}.$$

   Consider the last operation of T_1 and the first operation of T_2, we have $O_{17} \prec O_{21}$ because O_{17} is the write operation and they access the same data item X.

To maintain the order within the transactions, O_{17} must be the last operation in the schedule and O_{21} must be the first operation in the schedule. From the analyze above, O_{21} must after O_{17}. Therefore, there is no such conflict-equivalent non-serial schedule.