Database Tuning, ITU, Spring 2009

Rasmus Pagh

March 12, 2009

1 Transaction case study

The relation `Seats(seatID,class,reserved)` is used to handle seat reservations in an airplane. It contains one tuple for each seat, and the attribute `reserved` is 0 or 1 depending on whether a seat is free or booked. 15 minutes before departure, the booking of business class seats is closed, by transferring any free business class seats to "economy plus" customers. The below transaction (written in Oracle SQL) makes the transfer of seats, using the two relations `FreeBusinessSeats(seatID,reserved)` and `Upgrades(seatID)` to store intermediate results.

```
1. DELETE FROM FreeBusinessSeats;
2. DELETE FROM Upgrades;
3. INSERT INTO FreeBusinessSeats (SELECT seatID, reserved FROM Seats
                                  WHERE class='business' AND reserved=0);
4. INSERT INTO Upgrades (SELECT *
                         FROM (SELECT seatID FROM Seats
                               WHERE class='economy plus' AND reserved=1)
                         WHERE rownum<=(SELECT COUNT(*) FROM FreeBusinessSeats));
5. UPDATE FreeBusinessSeats SET reserved=1
   WHERE rownum<=(SELECT COUNT(*) FROM Upgrades);
6. UPDATE Seats SET reserved=0 WHERE seatID IN (SELECT * FROM Upgrades);
7. UPDATE Seats SET reserved=1
   WHERE (seatID,1) IN (SELECT seatID,reserved FROM FreebusinessSeats);
```

Explanation: The first two lines delete any old intermediate results. Line 3 inserts the free business class seats in the relation `FreeBusinessSeats`. Line 4 chooses reservations from "economy plus" that are to be upgraded. The number of upgrades is kept below the number of free seats by use of the `rownum` variable, which returns the row number of the current row in the relation. Line 5 marks the right number of free seats in `FreeBusinessSeats` as reserved. In line 6 and 7, the information on the new reservations are transferred to the `Seats` relation.

a) Assume that the above transaction runs at SQL isolation level `READ COMMITTED`, implemented using locking as discussed in the lecture (shared locks released after each statement, exclusive locks obtained using 2PL). Argue that if, at the same time, a reservation is made for a business class seat (i.e. a transaction that changes a value of `reserved` from 0 to 1), there may be a double booking, that is, the number of reserved seats is smaller than the number of passengers.

b) Because of the above problem, it seems like a good idea to consider a higher isolation level. We consider SQLs `REPEATABLE READ` and `SERIALIZABLE`. Again, we imagine a straightforward locking implementation (where `REPEATABLE READ` would allow adding new tuples, and `SERIALIZABLE` would not). Consider for each of these isolation levels whether a double booking may occur. Argue for your answer.

c) Think of a better way of implementing the transaction, based on explicit row locking.

2 Tangled transactions

Suppose that user A has an empty relation `Primes(p INT)`. Then users A and B issue the below statements, in this order. The start and end of transactions are indicated by horizontal lines.

A	B
`SELECT * FROM Primes;`	
	`SELECT * FROM A.Primes;`
`INSERT INTO Primes VALUES (2);`	
	`INSERT INTO A.Primes VALUES (3);`
`SELECT * FROM Primes;`	
	`SELECT * FROM A.Primes;`
`DELETE FROM Primes WHERE p=3;`	
	`DELETE FROM Primes WHERE p=2;`
`SELECT * FROM Primes;`	
	`SELECT * FROM A.Primes;`
`COMMIT;`	
`----------`	
`SELECT * FROM Primes;`	
	`SELECT * FROM A.Primes;`
	`COMMIT;`
	`----------`
`SELECT * FROM Primes;`	
	`SELECT * FROM A.Primes;`

- Explain what may be seen by user A and user B for each of the three SQL isolation levels `READ COMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.
- What state would the database be in after each of the possible serial schedule?
- What happens if the transactions run at different isolation levels?