CHEN Ruizhuo 3036032865 PA2

Lesson 1. Transaction isolation

The reservation transaction result is available only in booking1 session(NB_RESERVED_SEATS = 5,NB_AVAILABLE_SEATS=45), but not in control session(NB_RESERVED_SEATS = 0,NB_AVAILABLE_SEATS = 50).

Transactions are carried out in isolation until committed. DBMS ensures any 2 concurrent transactions do not "interfere". Each transactions executes as if ran by itself, concurrent accesses have no effect on transaction's behavior (no race conditions).

Lesson 2. Commit and rollback

After rollback the current transaction (in booking1), it is shown in both booking1 and control sessions that client1 does not buy the ticket(NB_RESERVED_SEATS = 0), and the number of available seats for the show does not decrease(NB_AVAILABLE_SEATS = 50).

So rollback is to undo work done in the current transaction, and back to the last committed transaction.

After committing the transaction(in booking1),it is show in control session that client1 buys the ticket(NB_RESERVED_SEATS = 5), and the number of available seats for the show decreases(NB_AVAILABLE_SEATS = 45).

Commit enables changes in a transaction to be implemented in the database. Once committing the transaction, the written will be permanent.

Lesson 3. Atomicity

After executing the transaction in booking1, and then crash the system, it is shown in control sessions that client1 does not buy the ticket(NB_RESERVED_SEATS = 0), and the number of available seats for the show does not decrease(NB_AVAILABLE_SEATS = 50).

Transaction in booking1 before crash hasn't change the database.

Issue a commit command in booking1, it is shown in control sessions that client1 does not buy the ticket(NB_RESERVED_SEATS = 0), and the number of available seats for the show does not decrease(NB_AVAILABLE_SEATS = 50).

System has atomicity: All actions in the transaction happen, or none happen. When system crash while transaction in progress, the transaction will be treat as 'abort'. DBMS typically undo the actions of aborted / failed transactions.

Lesson 4. Transaction on hold

After making a reservation in booking1, the reservation in booking2 can't be executed, it is in waiting list.

Each element has an unique lock, each transaction must acquire that lock before reading / writing that element. A transaction must obtain an X (exclusive) lock before writing. Before booking1 commit the transaction, client 1 and show 1 are locked due to written, any conflicting transactions are blocked waiting for this transaction. If transaction hold a conflicting lock, system will put requester into wait queue.

After committing the the booking1 session, the reservation in booking2 can be executed.
Session releases locks after committing. The waiting session can get lock once it is released.

At this point, we need execute the following request which hasn't been done. Then commit.
After commit, rollback won't work. So both booking1 and booking2 are executed. It is shown in control sessions that client1 bought 10 tickets(NB_RESERVED_SEATS = 10), client2 bought 2 tickets(NB_RESERVED_SEATS = 2), and the number of available seats for the show does not decrease(NB_AVAILABLE_SEATS = 38).
When transaction is on hold, it only enters the waiting queue, but once lock is released, it can be executed without aborting.

Lesson 5. Incomplete isolation = possible inconsistency
It is shown in control sessions that client1 bought 5 tickets(NB_RESERVED_SEATS = 5), client2 bought 2 tickets(NB_RESERVED_SEATS = 2), and the number of available seats for the show only decrease by 2 (NB_AVAILABLE_SEATS = 48).
There is no locks on read. Isolation levels is READ COMMITTED. Between reading and writing process, data may have been changed by other sessions. The value of the same object operated on by two sessions is determined by the last session.

Lesson 6. Full isolation
After committing the booking1 session, we can't update the client2, The transaction cannot be accessed continuously.
When isolation levels is 'Serializable', which assumes a static DB. It use strict 2PL+index-tree protocol. This means that once one session is working on the database, no other session can change it. As long as there are other transactions between the start and end of the serializable transaction that modify and commit the changes, the access cannot be done.

Lesson 7. Deadlocks
Update show in booking2, it can execute normally.

After updating show in booking2, the update operation by booking1 in show will wait.

After updating client1's reservation seats in booking1, the update operation by booking2 in client1 will wait.

By deadlock avoidance approach of wait-die, booking1 has higher priorities. Since booking2 holds the lock of show, booking1 will in wait queue. Since booking1 holds the lock of client, booking2 will aborts.
At the beginning of a transaction, a timestamp is assigned to it. Even it rolls back, the timestamp will maintain. The earlier transaction will has higher priority. If the data applied by Transaction1 is held by Transactio2. Only if Transaction1>Transaction2, Transaction1 will wait. Otherwise Transaction1 will rollback.

By deadlock avoidance approach of wound-wait, booking1 has higher priorities. Since booking2

holds the lock of show, booking1 also wants the lock of show, then booking2 will aborts. Since booking1 holds the lock of client, booking2 will in wait queue.
At the beginning of a transaction, a timestamp is assigned to it.Even it rolls back, the timestamp will maintain.The earlier transaction will has higher priority. If the data applied by Transaction1 is held by Transactio2. Only if Transaction1<Transaction2, Transaction1 will wait. Otherwise Transaction2 will rollback.

Lesson 8. Explicit locking

After booking1 completes the query, the query statement of booking2 enters the wait queue.
'FOR UPDATE' means once one session query one row, it will be locked to prevent other session to query. It means there is a lock on read. We can't query the table in booking2 with 'FOR UPDATE', since it need to wait booking1 release the lock, then lock the record.
If booking1 is committed, there is an error in booking2. The transaction cannot be accessed continuously. The result is the same with lesson 6.

Lesson 9. Read-only transaction

After setting the transaction mode to READ ONLY in the booking sessions, we can read the table but can't make any changes like delete, update and insert.

Lesson 10. Another look at Oracle's SERIALIZABLE isolation level

Both clients get the seats.
Under the isolation level of 'serializable', different sessions can operate in the same table before commit. Once one session commit the operation, other sessions can't make any changes but still can commit what has done.