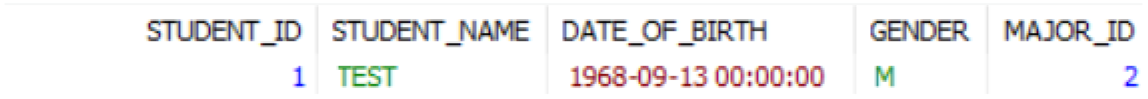
Laboratory 4 - Transactions

Author: Piotr Copek

Date: 23.04.2025

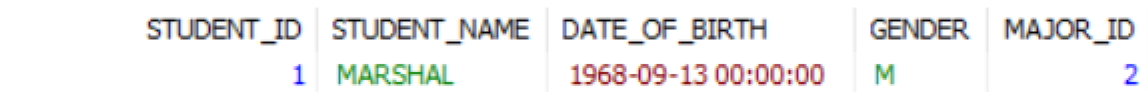
Dirty Read

Check again what Transaction 2 sees. Did it read uncommitted data? How could this be prevented?



STUDENT_ID	STUDENT_NAME	DATE_OF_BIRTH	GENDER	MAJOR_ID
1	TEST	1968-09-13 00:00:00	M	2

[Figure 1] - Result before `unrolling`.



STUDENT_ID	STUDENT_NAME	DATE_OF_BIRTH	GENDER	MAJOR_ID
1	MARSHAL	1968-09-13 00:00:00	M	2

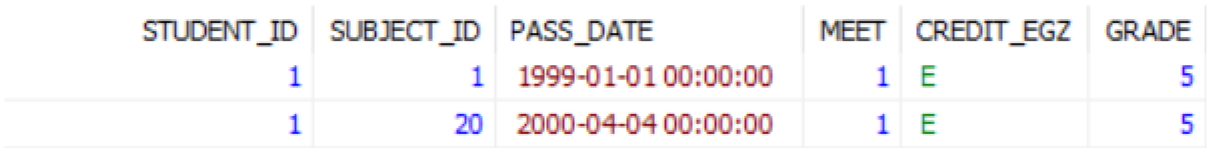
[Figure 2] - Result after `unrolling`.

Transaction 2 read uncommitted data of the temporary change `student_name = 'TEST'`, which was later rolled back in Transaction 1.

To prevent this from happening we could set the isolation level to `READ COMMITTED` or higher. This ensures a transaction only sees committed data.

Non-Repeatable Read

In which session did you set the READ COMMITTED isolation level and why? Did the data read in Transaction 1 change between the first, second, and third query? If so, why did that happen, and how could it be prevented?



STUDENT_ID	SUBJECT_ID	PASS_DATE	MEET	CREDIT_EGZ	GRADE
1	1	1999-01-01 00:00:00	1	E	5
1	20	2000-04-04 00:00:00	1	E	5

[Figure 3] - Result after `committing`.

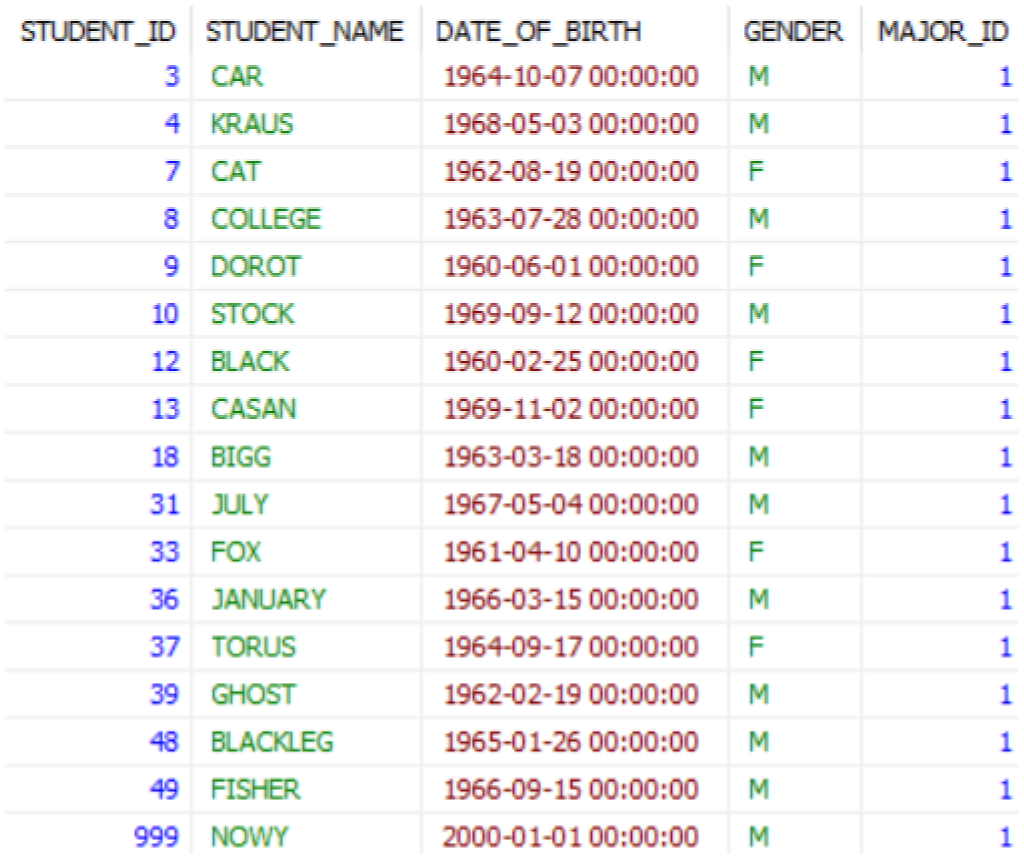
`READ COMMITTED` should be set in Transaction 1 because it determines what that transaction can "see". Transaction 2 doesn’t need this isolation level for the test.

Indeed the data changed after Transaction 2 committed its update.

To prevent this from happening we could use `REPEATABLE READ` isolation in Transaction 1 to ensure consistent reads.

Phantom Reads

Did the new student appear in the results? Do you think this is correct? If not, what would you propose to prevent it?



STUDENT_ID	STUDENT_NAME	DATE_OF_BIRTH	GENDER	MAJOR_ID
3	CAR	1964-10-07 00:00:00	M	1
4	KRAUS	1968-05-03 00:00:00	M	1
7	CAT	1962-08-19 00:00:00	F	1
8	COLLEGE	1963-07-28 00:00:00	M	1
9	DOROT	1960-06-01 00:00:00	F	1
10	STOCK	1969-09-12 00:00:00	M	1
12	BLACK	1960-02-25 00:00:00	F	1
13	CASAN	1969-11-02 00:00:00	F	1
18	BIGG	1963-03-18 00:00:00	M	1
31	JULY	1967-05-04 00:00:00	M	1
33	FOX	1961-04-10 00:00:00	F	1
36	JANUARY	1966-03-15 00:00:00	M	1
37	TORUS	1964-09-17 00:00:00	F	1
39	GHOST	1962-02-19 00:00:00	M	1
48	BLACKLEG	1965-01-26 00:00:00	M	1
49	FISHER	1966-09-15 00:00:00	M	1
999	NOWY	2000-01-01 00:00:00	M	1

[Figure 4] - Result after `committing` the `INSERT` of the new student.

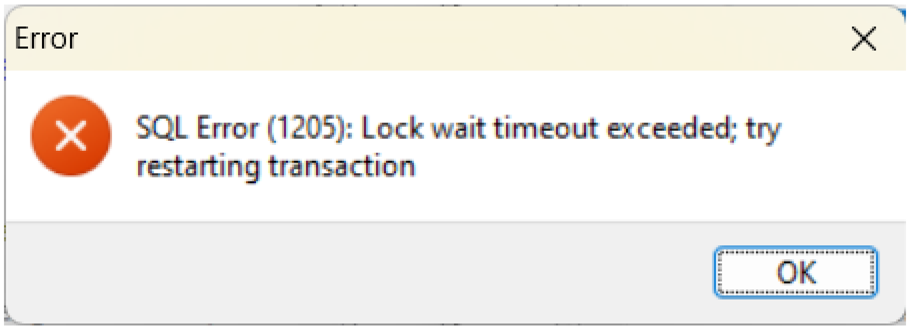
Indeed the new student `student_id = 999` appeared in the second query in Transaction 1.

To prevent this from happening we could use `SERIALIZABLE` isolation to lock the range of rows matching `major_id = 1`, preventing inserts until Transaction 1 completes.

Deadlocks

What happened in step 5? Describe how to avoid deadlocks.

Deadlock indeed occurred. Transaction 1 held a lock on `employee_id = 1` and waited for `employee_id = 2`, while Transaction 2 held a lock on `employee_id = 2` and waited for `employee_id = 1`. MySQL detected this and aborted one transaction.



```
SQL Error (1205): Lock wait timeout exceeded; try
restarting transaction
```

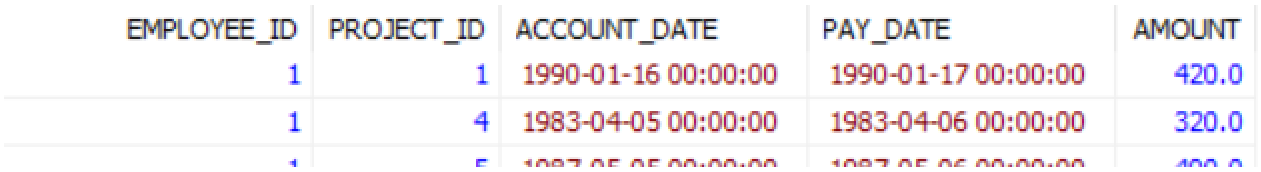
[Figure 5] - Warning about deadlock.

There are few things we could do to prevent this from happening:

- Always access tables in the same order.
- Implement retry logic in applications after deadlocks.
- Use short-lived transactions to reduce contention.

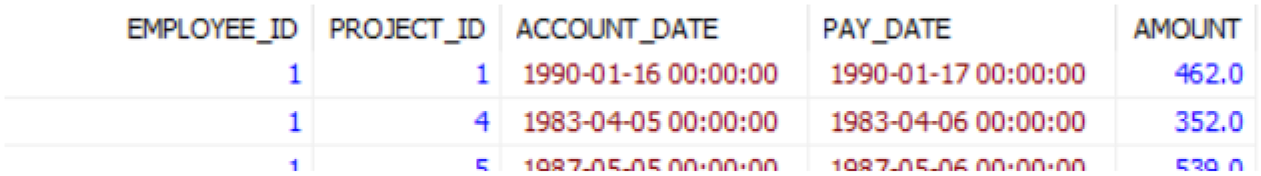
Blocking Reads

Does Transaction 2 wait? Does it see the changed value?



EMPLOYEE_ID	PROJECT_ID	ACCOUNT_DATE	PAY_DATE	AMOUNT
1	1	1990-01-16 00:00:00	1990-01-17 00:00:00	420.0
1	4	1983-04-05 00:00:00	1983-04-06 00:00:00	320.0
1	5	1987-05-05 00:00:00	1987-05-06 00:00:00	490.0

[Figure 6] - Values before committing.



EMPLOYEE_ID	PROJECT_ID	ACCOUNT_DATE	PAY_DATE	AMOUNT
1	1	1990-01-16 00:00:00	1990-01-17 00:00:00	462.0
1	4	1983-04-05 00:00:00	1983-04-06 00:00:00	352.0
1	5	1987-05-05 00:00:00	1987-05-06 00:00:00	539.0

[Figure 7] - Values after committing.

Were reads in Transaction 2 blocked? Do the results of queries (B) and (C) differ from (A)? Test the same exercise with isolation levels: READ COMMITTED and SERIALIZABLE. What are the differences?

At `REPEATABLE READ`:

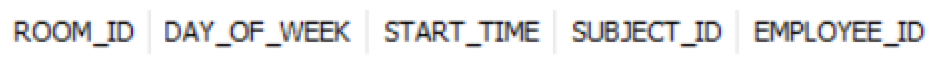
- Transaction 2 waited until Transaction 1 committed.
- Query B during Transaction 1 showed the old value. Query C after commit showed the updated value.

Isolation level differences:

- `READ COMMITTED` - Transaction 2 would see the new value immediately after Transaction 1 commits.
- `SERIALIZABLE` - Transaction 2 would wait until Transaction 1 completes, similar to `REPEATABLE READ`.

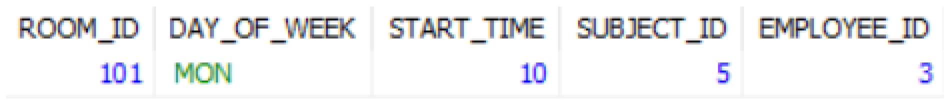
Using Various Isolation Levels for Transaction Testing

What were the differences in the retrieved values? What caused them?



ROOM_ID	DAY_OF_WEEK	START_TIME	SUBJECT_ID	EMPLOYEE_ID

[Figure 8] - Schedule before inserting.



ROOM_ID	DAY_OF_WEEK	START_TIME	SUBJECT_ID	EMPLOYEE_ID
101	MON	10	5	3

[Figure 9] - Schedule after committing insertion in Transaction 2.

Transaction 1 with `READ COMMITTED` - The second query saw the new row inserted by Transaction 2 after committing.

Transaction 2 `REPEATABLE READ` - Would not see the new row if it re-read the data in the same transaction.

`READ COMMITTED` allows seeing committed changes from other transactions, while `REPEATABLE READ` maintains a snapshot.