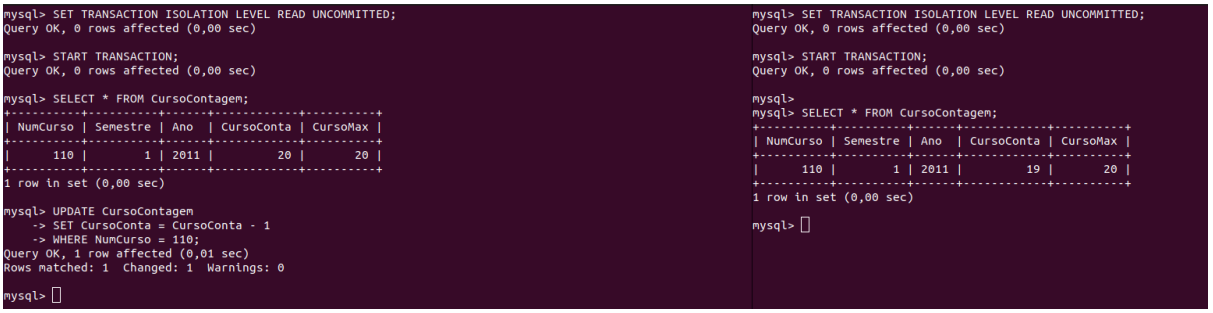
Nome: Harrison Caetano Candido
RA: 156264

1) Faça o exercício que está na folha em anexo e explique o que ocorre em cada teste, indicando , se for o caso, o erro de controle de concorrência que ocorre.
R:

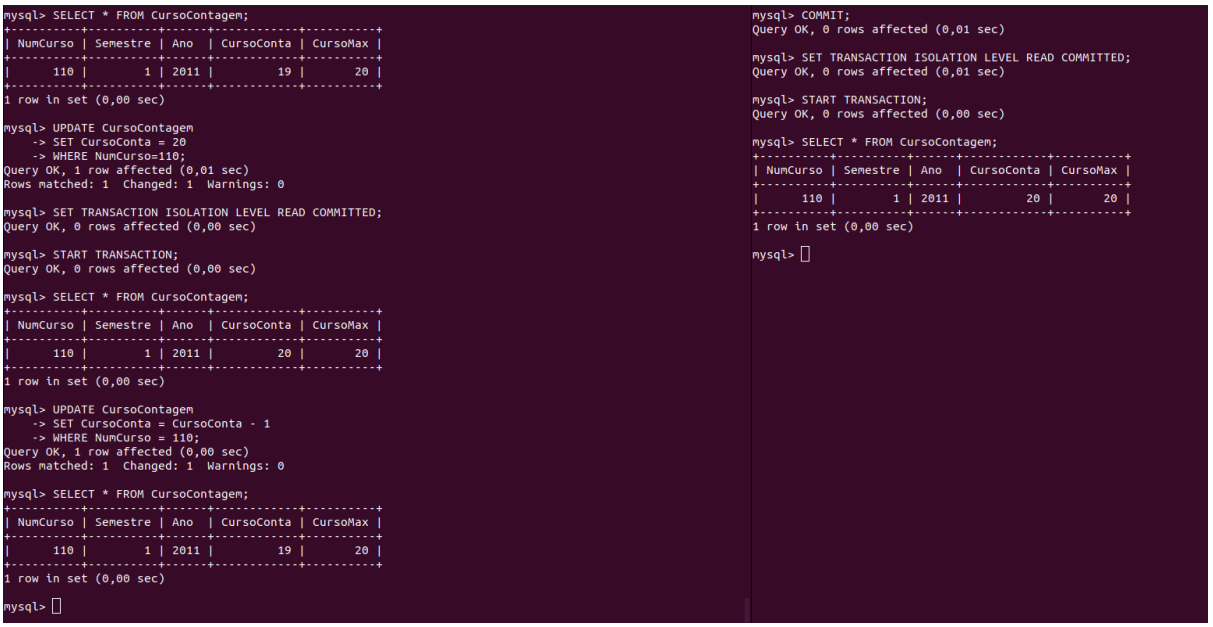
TESTE 1:

O nível de isolamento READ UNCOMMITED pode gerar um “DIRTY READ” ou “BLIND OVERWRITING” na transação 2, pois esta pode ler um dado que foi alterado, mas que ainda não foi commitado, ou seja, se esses dados lidos forem utilizados para operações críticas, pode haver inconsistência no processo.



TESTE 2:

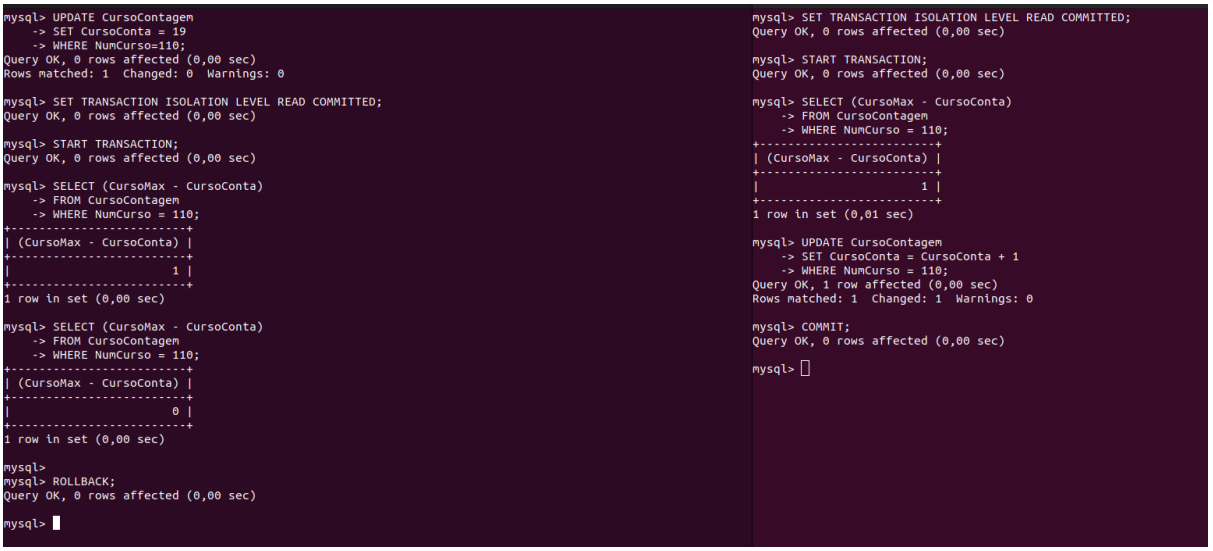
O nível de isolamento READ COMMITED pode gerar um NON REPETEABLE READ na transação 2, visto que se for necessário repetir um select na mesma transação, algumas linhas recuperadas no primeiro SELECT podem não ser iguais às recuperadas posteriormente a transação 1 realizar COMMIT.



TESTE 3:

Aqui ocorre a LOST UPDATE, em que as duas transações realizam WRITE no mesmo item de dado e um dos WRITEs é perdido, no caso, o UPDATE da transação 2 pode vir a ser perdido diante de um UPDATE da transação 1.

Também podemos dizer que uma NON REPEATABLE READ pode ocorrer se a transação 1 fizer uma leitura após o COMMIT da transação 2.



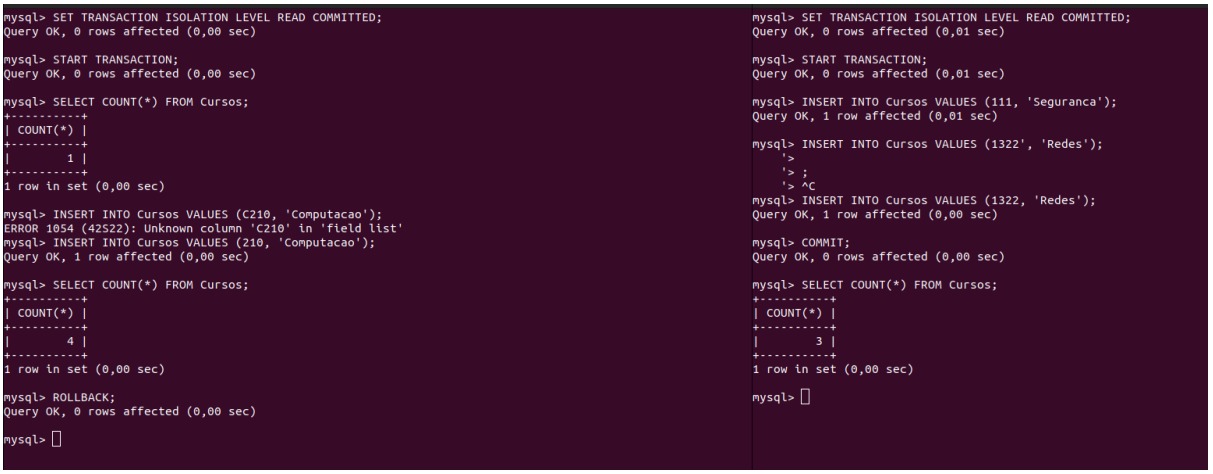
```
mysql> UPDATE CursoContagem
    -> SET CursoConta = 19
    -> WHERE NumCurso=110;
Query OK, 0 rows affected (0,00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0,00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0,00 sec)

mysql> SELECT (CursoMax - CursoConta)
    -> FROM CursoContagem
    -> WHERE NumCurso = 110;
+--------------------------+
| (CursoMax - CursoConta) |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0,00 sec)

mysql> SELECT (CursoMax - CursoConta)
    -> FROM CursoContagem
    -> WHERE NumCurso = 110;
+--------------------------+
| (CursoMax - CursoConta) |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set (0,00 sec)

mysql>
mysql> ROLLBACK;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

```
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0,00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0,00 sec)

mysql> SELECT (CursoMax - CursoConta)
    -> FROM CursoContagem
    -> WHERE NumCurso = 110;
+--------------------------+
| (CursoMax - CursoConta) |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0,01 sec)

mysql> UPDATE CursoContagem
    -> SET CursoConta = CursoConta + 1
    -> WHERE NumCurso = 110;
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

TESTE 4:

Na transação 1 ocorre o PHANTOM PROBLEM, uma vez que se repetido um SELECT aparecem novas linhas após o COMMIT feito na transação 2.

Também pode ocorrer o NON REPEATABLE READ, quando a transação 1 der mais de um SELECT após COMMITs feitos pela transação 2.



```
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0,00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0,00 sec)

mysql> SELECT COUNT(*) FROM Cursos;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0,00 sec)

mysql> INSERT INTO Cursos VALUES (C210, 'Computacao');
ERROR 1054 (42522): Unknown column 'C210' in 'field list'
mysql> INSERT INTO Cursos VALUES (210, 'Computacao');
Query OK, 1 row affected (0,00 sec)

mysql> SELECT COUNT(*) FROM Cursos;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0,00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0,00 sec)

mysql>
```

```
mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0,01 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0,01 sec)

mysql> INSERT INTO Cursos VALUES (111, 'Seguranca');
Query OK, 1 row affected (0,01 sec)

mysql> INSERT INTO Cursos VALUES (1322', 'Redes');
    '>
    '> ;
    '> ^C
mysql> INSERT INTO Cursos VALUES (1322, 'Redes');
Query OK, 1 row affected (0,00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0,00 sec)

mysql> SELECT COUNT(*) FROM Cursos;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0,00 sec)

mysql>
```

2) Faça uma pesquisa sobre:
- Escolha 1 SGBD relacional e um SGBD No-SQL e escreva sobre cada um implementa os níveis de isolamento. No máximo 2 parágrafos para cada SBGD escolhido.
- EX: Oracle e MongoDb ou PosgtgreSQL e GoogleBigTable
R:

ORACLE SQL

O Oracle implementa os níveis de isolamento padrão do SQL (READ COMMITTED, SERIALIZABLE, READ ONLY e READ WRITE), com foco em concorrência e consistência. Diferentemente de outros SGBDs, o Oracle não possui um nível READ UNCOMMITTED, pois prioriza a integridade dos dados. O nível padrão (READ COMMITTED) evita dirty

reads, enquanto SERIALIZABLE garante isolamento total, usando snapshots e bloqueios para evitar phantom reads e *non-repeatable reads*. O Oracle também oferece MVCC (Multi-Version Concurrency Control), permitindo leituras consistentes sem bloquear escritas, ideal para ambientes com alta concorrência.

Mongo DB

O MongoDB, como banco NoSQL, adota uma abordagem flexível ao isolamento. Para operações em um único documento, ele garante atomicidade, mas em transações multidocumento (suportadas desde a versão 4.0), usa snapshot isolation, onde cada transação opera em uma "foto" consistente dos dados no momento de seu início. O MongoDB não implementa níveis de isolamento tradicionais como em bancos relacionais; em vez disso, permite ajustes via write concerns (ex.: "majority") para controlar consistência vs. disponibilidade. Isso o torna eficiente para escalabilidade horizontal, mas menos rigoroso em cenários que exigem isolamento estrito como o Oracle.

3) Explique a figura em anexo.

R: O programador que está no computador deletou por acidente todas as linhas da tabela “CLIENTE” do banco de dados, provavelmente com o comando DELETE FROM table_name ou TRUNCATE TABLE table_name. Visto que a maioria dos SGBDs de empresas possuem o commit automático desabilitado por padrão, é sempre necessário dar COMMIT após as operações feitas de uma transação, se não operações de WRITE não são persistidas no banco de dados. A piada está na preocupação do programador de ter cometido um erro e quando outro programador o pergunta se ele deu commit na esperança de ainda ser possível dar um ROLLBACK, o primeiro programador simplesmente termina a transação com COMMIT como se ele tivesse se esquecido do procedimento, e assim persiste as operações no banco de dados, que antes estavam numa camada de buffer.