《数据库系统及应用实践》课程实验报告

实验 7：事务处理

姓　　名: 李鹏达　　学　　号: 10225101460　　完成日期: 2024 年 6 月 26 日

1 实验目标

1. 学习和掌握 MySQL 数据库管理系统中事务处理相关的设置和基本操作；
2. 学习和理解事务处理过程中不同事务隔离级别所对应的异常，能够根据应用场景选择合适的事务隔离级别；

2 实验过程记录

在 Docker 容器中启动 MySQL 数据库。

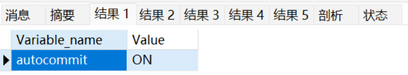
2.1 事务相关设置和基本操作

启动三个 MySQL 客户端（T_1, T_2, T_3），分别连接到数据库。

分别在不同客户端中执行下列语句，查看和修改系统变量的值；

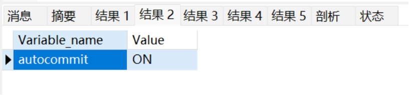
```sql
show global variables like 'autocommit'; -- T1
show session variables like 'autocommit'; -- T1
show global variables like '%transaction%'; -- T1
show session variables like '%transaction%'; --T1
set global autocommit = OFF; -- T1
set session autocommit = OFF; -- T1
select @@global.autocommit, @@session.autocommit; -- T1
select @@global.autocommit, @@session.autocommit; -- T2
```

结果如下：



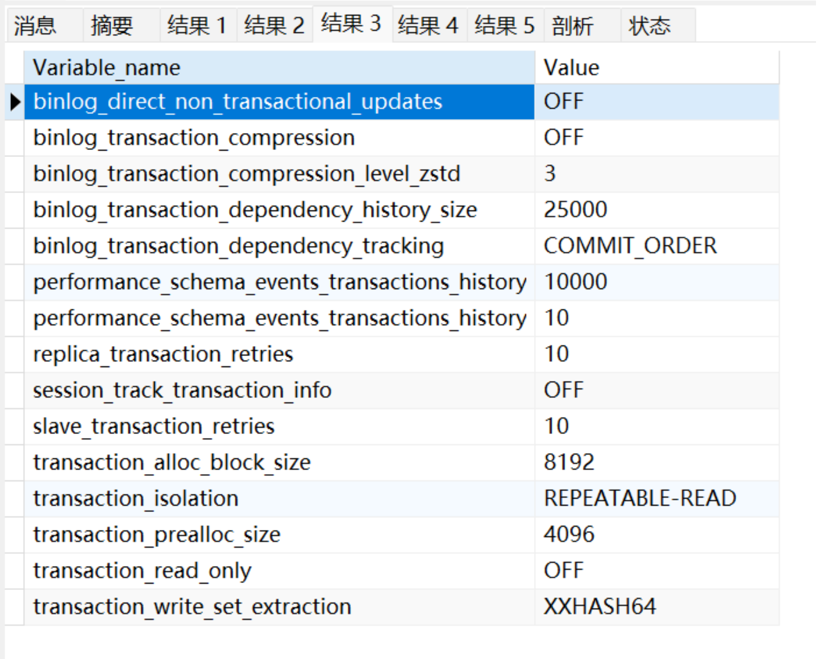
Variable_name	Value
autocommit	ON

图 1: T1 执行结果（1）



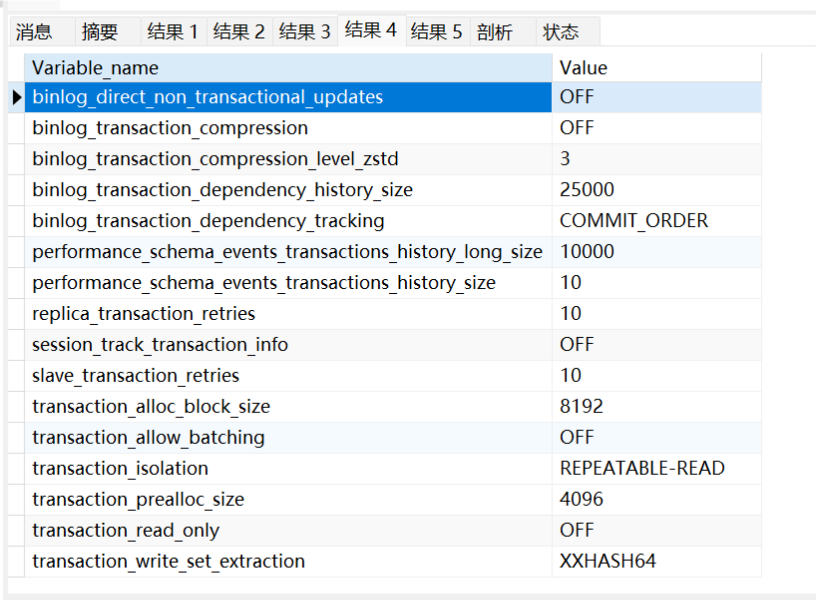
Variable_name	Value
autocommit	ON

图 2: T1 执行结果（2）



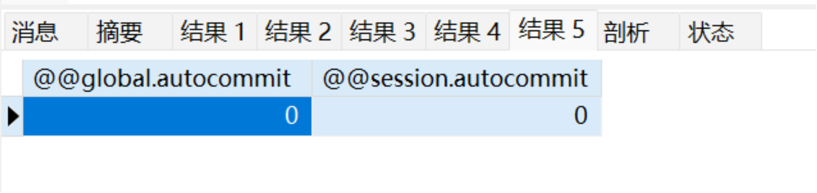
Variable_name	Value
binlog_direct_non_transactional_updates	OFF
binlog_transaction_compression	OFF
binlog_transaction_compression_level_zstd	3
binlog_transaction_dependency_history_size	25000
binlog_transaction_dependency_tracking	COMMIT_ORDER
performance_schema_events_transactions_history	10000
performance_schema_events_transactions_history	10
replica_transaction_retries	10
session_track_transaction_info	OFF
slave_transaction_retries	10
transaction_alloc_block_size	8192
transaction_isolation	REPEATABLE-READ
transaction_prealloc_size	4096
transaction_read_only	OFF
transaction_write_set_extraction	XXHASH64

图 3: T1 执行结果（3）



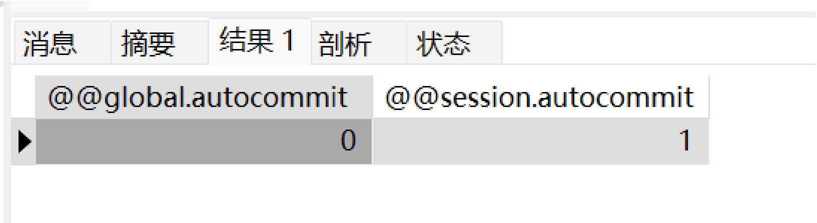
Variable_name	Value
binlog_direct_non_transactional_updates	OFF
binlog_transaction_compression	OFF
binlog_transaction_compression_level_zstd	3
binlog_transaction_dependency_history_size	25000
binlog_transaction_dependency_tracking	COMMIT_ORDER
performance_schema_events_transactions_history_long_size	10000
performance_schema_events_transactions_history_size	10
replica_transaction_retries	10
session_track_transaction_info	OFF
slave_transaction_retries	10
transaction_alloc_block_size	8192
transaction_allow_batching	OFF
transaction_isolation	REPEATABLE-READ
transaction_prealloc_size	4096
transaction_read_only	OFF
transaction_write_set_extraction	XXHASH64

图 4: T1 执行结果（4）



@@global.autocommit	@@session.autocommit
0	0

图 5: T1 执行结果（5）



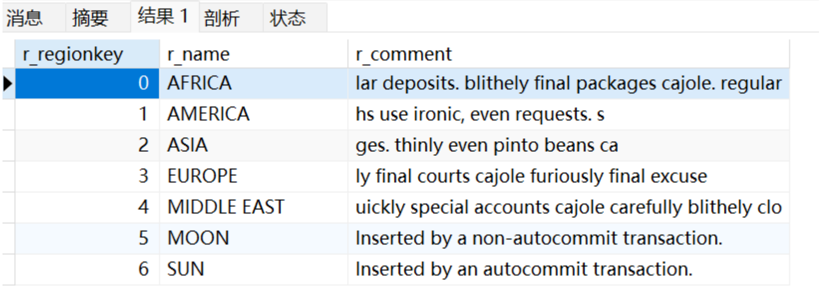
@@global.autocommit	@@session.autocommit
0	1

图 6: T2 执行结果

分别在不同客户端中执行下列语句，对比自动提交事务与手动提交事务的差别；

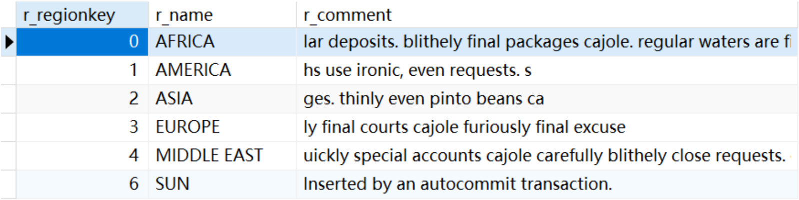
```sql
insert into region values(5,'MOON','Inserted by a non-autocommit transaction.'); -- T1
insert into region values(6,'SUN','Inserted by an autocommit transaction.'); -- T2
start transaction; -- T3
insert into region values(7,'STAR','Inserted by a rollback transaction.'); -- T3
select * from region; -- T1
select * from region; -- T2
select * from region; -- T3
commit; -- T1
rollback; -- T3
select * from region; -- T1
select * from region; -- T2
select * from region; -- T3
```

结果如下：



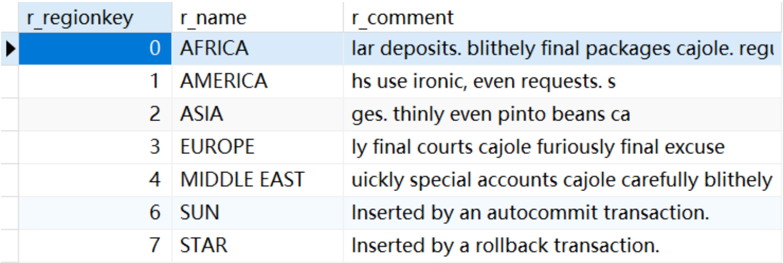
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely clo
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 7: T1 执行结果（1）



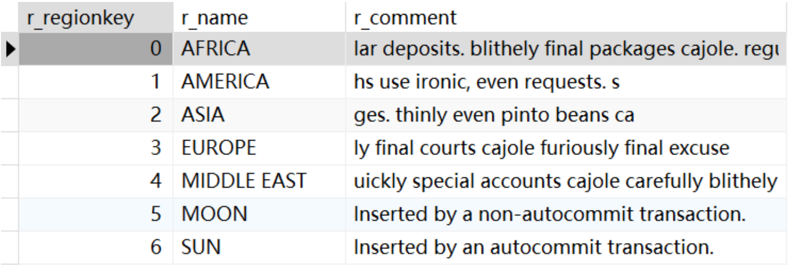
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
6	SUN	Inserted by an autocommit transaction.

图 8: T2 执行结果（1）



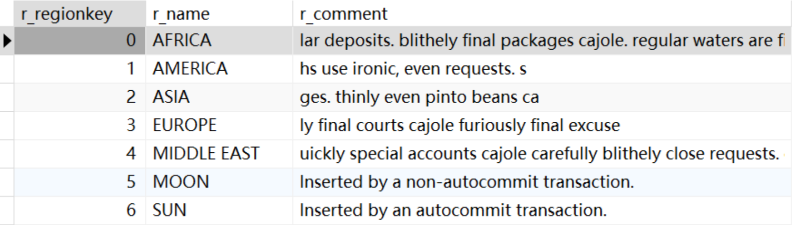
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
6	SUN	Inserted by an autocommit transaction.
7	STAR	Inserted by a rollback transaction.

图 9: T3 执行结果（1）



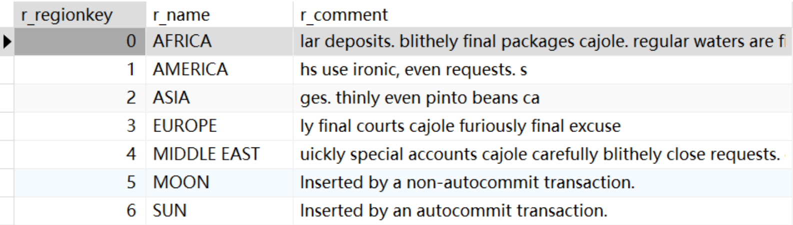
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 10: T1 执行结果（2）



r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 11: T2 执行结果（2）



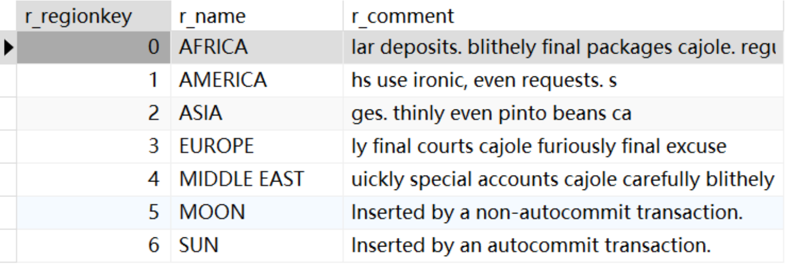
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 12: T3 执行结果（2）

分别在不同客户端中执行下列语句，关注不同语句对于事务的影响；

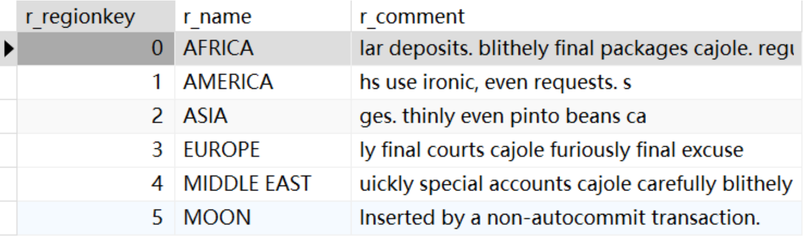
```
start transaction; -- T2
delete from region where r_regionkey = 6; -- T2
select * from region; -- T1
select * from region; -- T2
select * from region; -- T3
rollback; -- T2
select * from region; -- T1
select * from region; -- T2
select * from region; -- T3
start transaction; -- T2
delete from region where r_regionkey = 6; -- T2
select * from region; -- T1
select * from region; -- T2
select * from region; -- T3
analyze table orders; -- T2
rollback; -- T2
select * from region; -- T1
select * from region; -- T2
select * from region; -- T3
commit; --T1
```

结果如下：



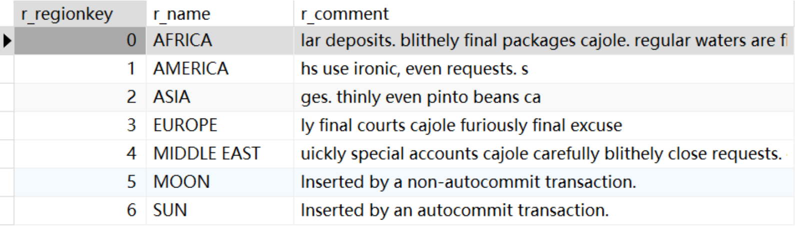
	r_regionkey	r_name	r_comment
▶	0	AFRICA	lar deposits. blithely final packages cajole. regu
	1	AMERICA	hs use ironic, even requests. s
	2	ASIA	ges. thinly even pinto beans ca
	3	EUROPE	ly final courts cajole furiously final excuse
	4	MIDDLE EAST	uickly special accounts cajole carefully blithely
	5	MOON	Inserted by a non-autocommit transaction.
	6	SUN	Inserted by an autocommit transaction.

图 13：T1 执行结果（1）



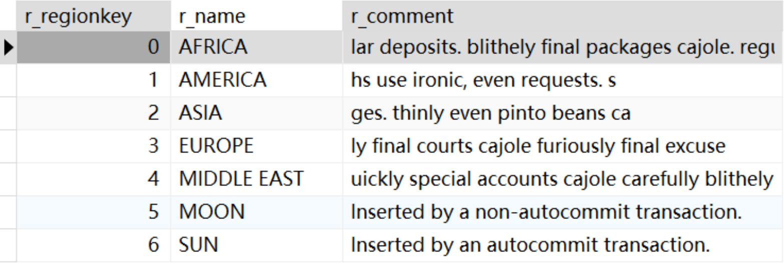
	r_regionkey	r_name	r_comment
▶	0	AFRICA	lar deposits. blithely final packages cajole. regu
	1	AMERICA	hs use ironic, even requests. s
	2	ASIA	ges. thinly even pinto beans ca
	3	EUROPE	ly final courts cajole furiously final excuse
	4	MIDDLE EAST	uickly special accounts cajole carefully blithely
	5	MOON	Inserted by a non-autocommit transaction.

图 14：T2 执行结果（1）



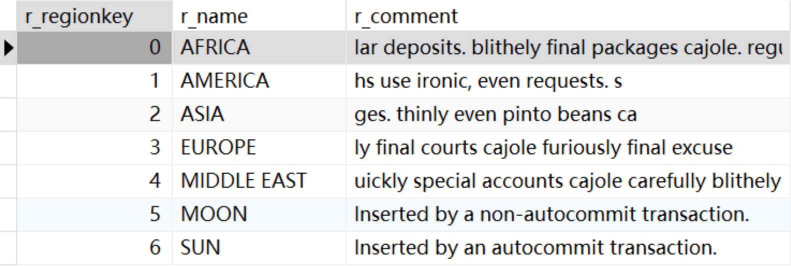
	r_regionkey	r_name	r_comment
▶	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
	1	AMERICA	hs use ironic, even requests. s
	2	ASIA	ges. thinly even pinto beans ca
	3	EUROPE	ly final courts cajole furiously final excuse
	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
	5	MOON	Inserted by a non-autocommit transaction.
	6	SUN	Inserted by an autocommit transaction.

图 15：T3 执行结果（1）



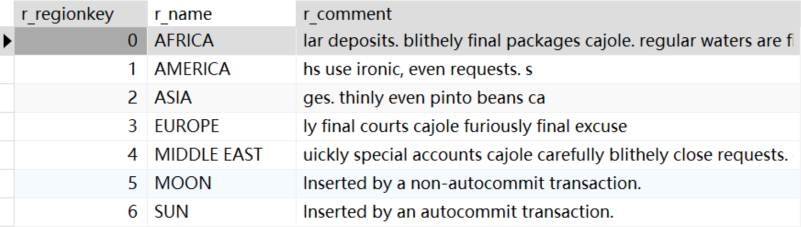
	r_regionkey	r_name	r_comment
▶	0	AFRICA	lar deposits. blithely final packages cajole. regu
	1	AMERICA	hs use ironic, even requests. s
	2	ASIA	ges. thinly even pinto beans ca
	3	EUROPE	ly final courts cajole furiously final excuse
	4	MIDDLE EAST	uickly special accounts cajole carefully blithely
	5	MOON	Inserted by a non-autocommit transaction.
	6	SUN	Inserted by an autocommit transaction.

图 16：T1 执行结果（2）



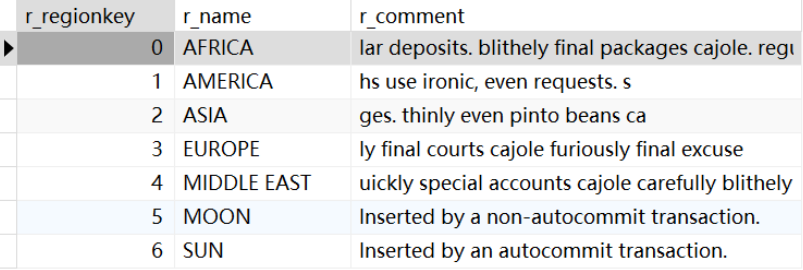
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 17：T2 执行结果（2）



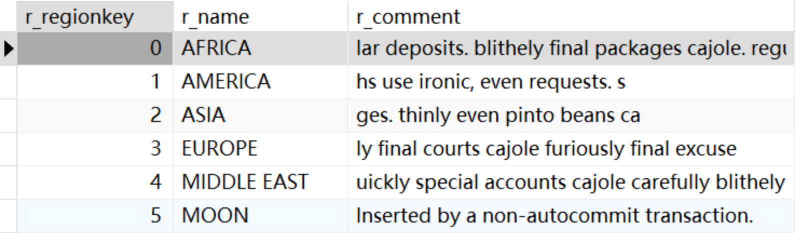
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are f
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 18：T3 执行结果（2）



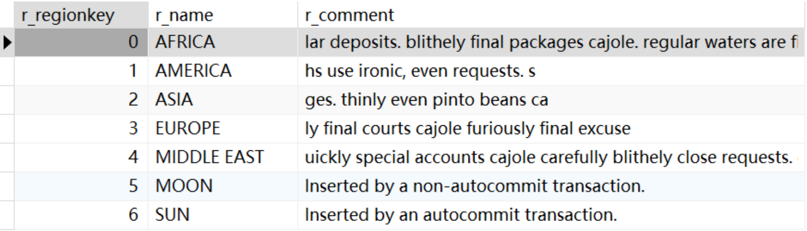
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 19：T1 执行结果（3）



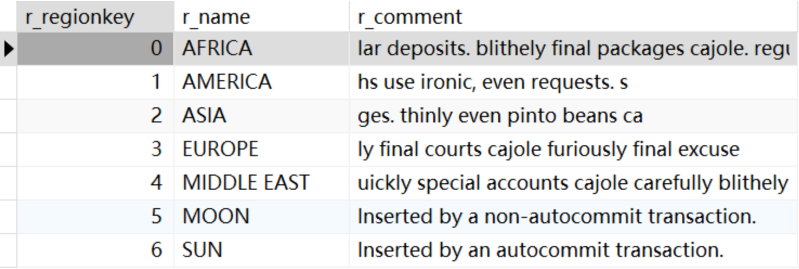
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
5	MOON	Inserted by a non-autocommit transaction.

图 20：T2 执行结果（3）



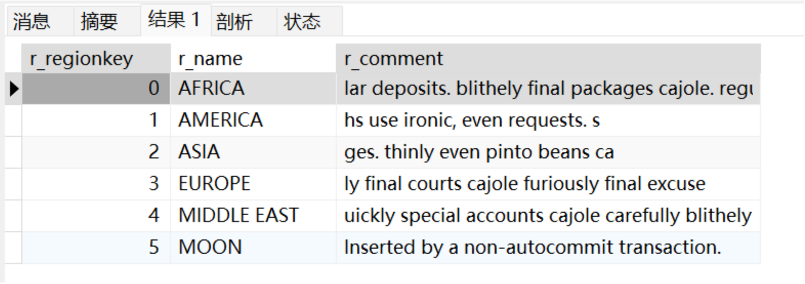
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 21：T3 执行结果（3）



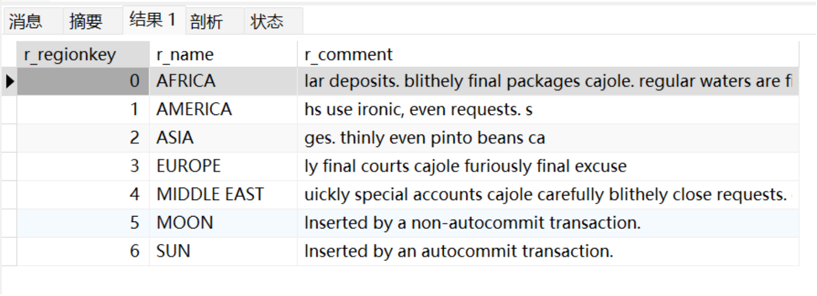
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 22：T1 执行结果（4）



r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regu
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely
5	MOON	Inserted by a non-autocommit transaction.

图 23：T2 执行结果（4）



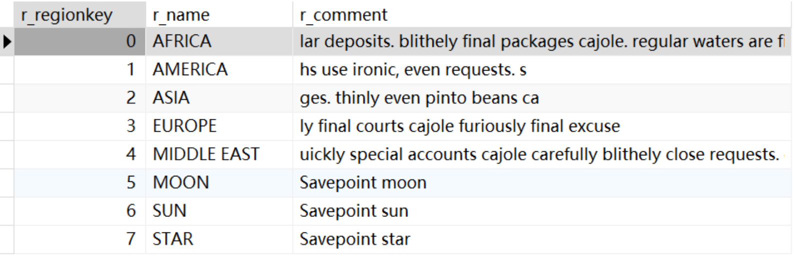
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Inserted by a non-autocommit transaction.
6	SUN	Inserted by an autocommit transaction.

图 24：T3 执行结果（4）

在客户端 T2 中执行下列语句，关注如何使用 SAVEPOINT；

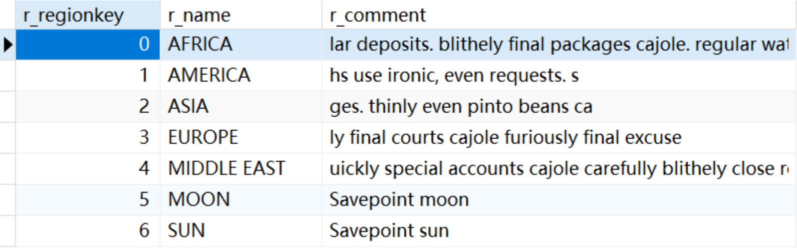
```sql
start transaction;
insert into region values(5,'MOON','Savepoint moon');
savepoint moon;
insert into region values(6,'SUN','Savepoint sun');
savepoint sun;
insert into region values(7,'STAR','Savepoint star');
savepoint star;
select * from region;
rollback to sun;
select * from region;
rollback to star;
select * from region;
rollback to moon;
select * from region;
commit;
```

结果如下：



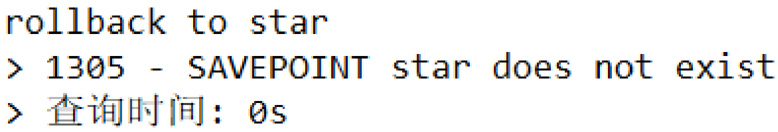
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Savepoint moon
6	SUN	Savepoint sun
7	STAR	Savepoint star

图 25: T2 执行结果（1）



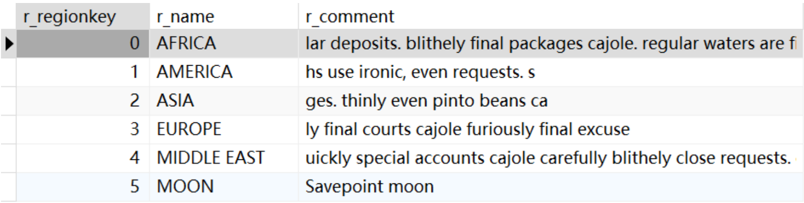
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular wat
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close r
5	MOON	Savepoint moon
6	SUN	Savepoint sun

图 26: T2 执行结果（2）



```
rollback to star
> 1305 - SAVEPOINT star does not exist
> 查询时间: 0s
```

图 27: T2 执行结果（3）



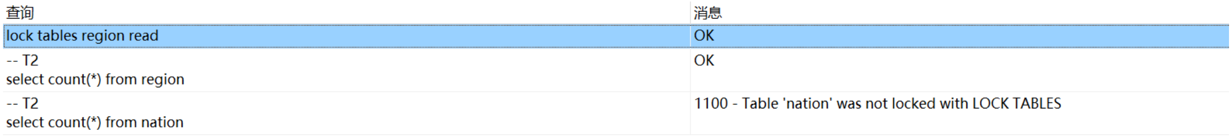
r_regionkey	r_name	r_comment
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are f
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.
5	MOON	Savepoint moon

图 28: T2 执行结果（4）

可以发现，当回滚到 sun 后，就无法再回滚到 star 了，因为 star 是在 sun 之后创建的。
分别在不同客户端中执行下列语句，关注如何使用 LOCK TABLES 语句；

```
lock tables region read; -- T2
select count(*) from region; -- T2
select count(*) from nation; -- T2
select * from region; -- T3
delete from region where r_regionkey = 5; -- T3
lock tables nation write, nation as n1 read; -- T2
insert into nation select n_nationkey+100, n_name, n_regionkey,n_comment
from nation; -- T2;
insert into nation select n_nationkey+100, n_name, n_regionkey,n_comment
from nation as n1; -- T2;
lock tables region read; -- T2
select * from region; -- T2
select * from region as r; -- T2
lock tables region as r read; -- T2
select * from region; -- T2
select * from region as r; -- T2
delete from nation where n_nationkey >= 100; -- T3
```

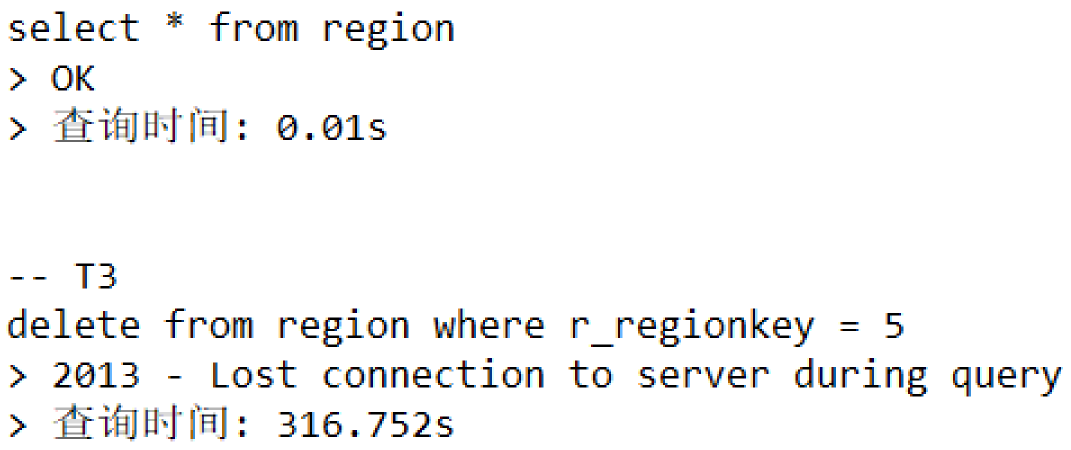
结果如下：



查询	消息
lock tables region read	OK
-- T2 select count(*) from region	OK
-- T2 select count(*) from nation	1100 - Table 'nation' was not locked with LOCK TABLES

图 29: T2 执行结果（1）

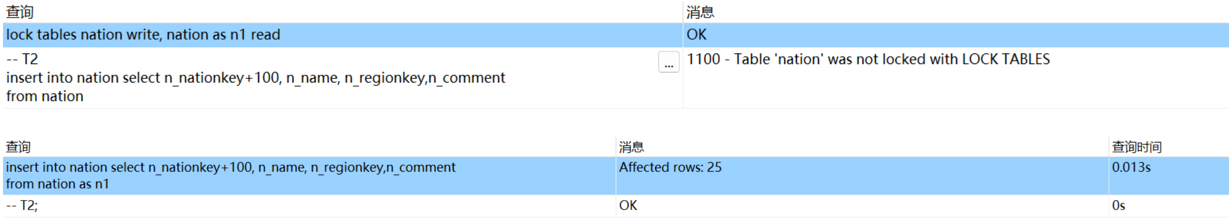
可以发现，当 T2 给 region 表加锁后，T2 可以读取 region 表，但无法读取 nation 表。



```
select * from region
> OK
> 查询时间: 0.01s


-- T3
delete from region where r_regionkey = 5
> 2013 - Lost connection to server during query
> 查询时间: 316.752s
```

图 30: T3 执行结果（1）

发现 T3 可以读取但无法删除 region 表中的数据，因为 region 表被 T2 锁住了。

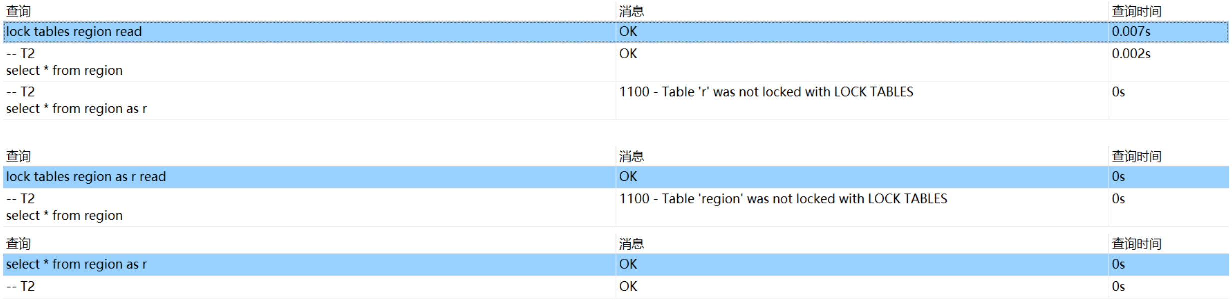


查询	消息
lock tables nation write, nation as n1 read	OK
-- T2 insert into nation select n_nationkey+100, n_name, n_regionkey,n_comment from nation	1100 - Table 'nation' was not locked with LOCK TABLES

查询	消息	查询时间
insert into nation select n_nationkey+100, n_name, n_regionkey,n_comment from nation as n1	Affected rows: 25	0.013s
-- T2;	OK	0s

图 31: T2 执行结果（2）

在获取 nation 表的写锁，和别名为 n1 的读锁后，在复杂查询中，必须使用别名来读取数据。



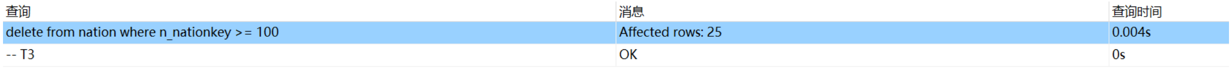
查询	消息	查询时间
lock tables region read	OK	0.007s
-- T2 select * from region	OK	0.002s
-- T2 select * from region as r	1100 - Table 'r' was not locked with LOCK TABLES	0s

查询	消息	查询时间
lock tables region as r read	OK	0s
-- T2 select * from region	1100 - Table 'region' was not locked with LOCK TABLES	0s

查询	消息	查询时间
select * from region as r	OK	0s
-- T2	OK	0s

图 32: T2 执行结果（3）

同时，获取锁时未使用别名，后续查询时则不能使用别名。使用别名获取锁后，后续查询也必须使用别名。



查询	消息	查询时间
delete from nation where n_nationkey >= 100	Affected rows: 25	0.004s
-- T3	OK	0s

图 33: T3 执行结果（2）

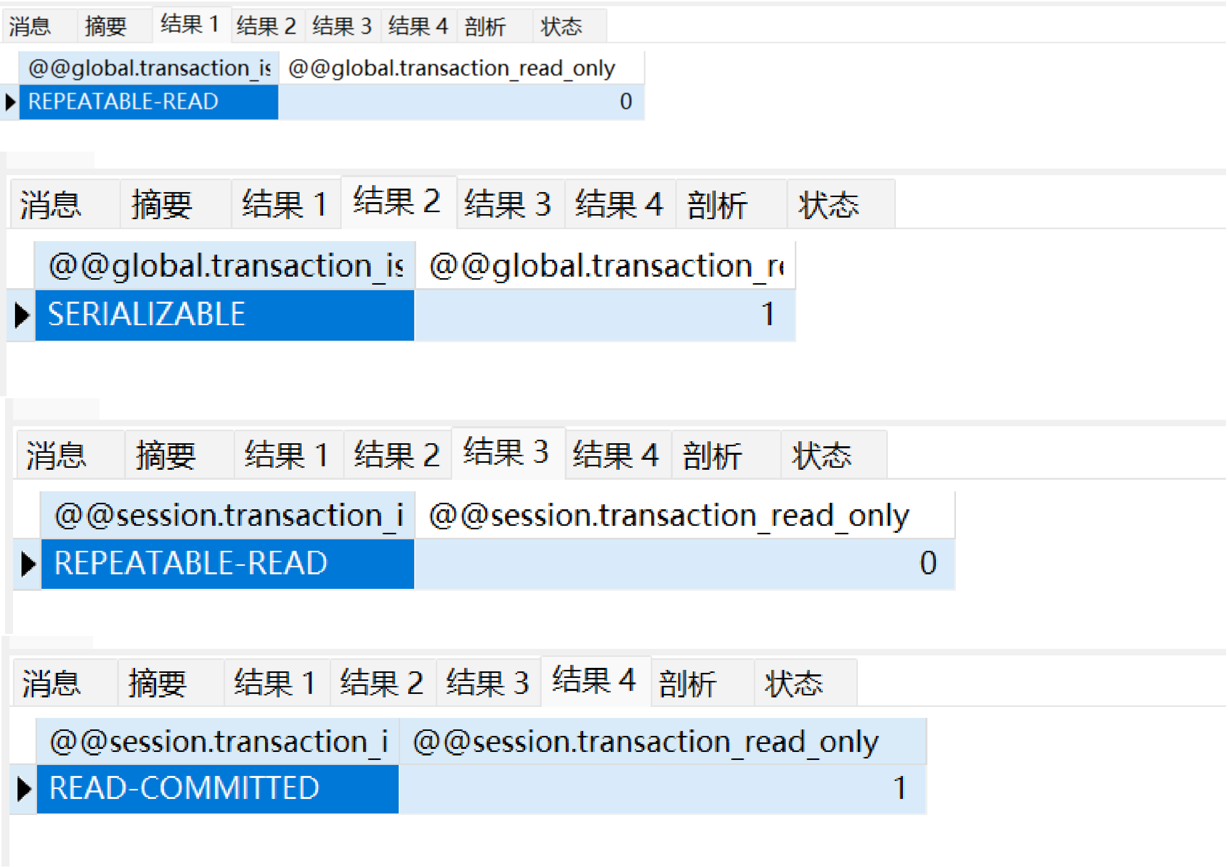
此时，T3 可以删除 nation 表中的数据。

在客户端 T2 中执行下列语句，关注如何设置事务属性；

```sql
select @@global.transaction_isolation, @@global.transaction_read_only;
set global transaction isolation level serializable;
set global transaction read only;
select @@global.transaction_isolation, @@global.transaction_read_only;
select @@session.transaction_isolation, @@session.transaction_read_only;
set @@session.transaction_isolation = 'read-committed';
set @@session.transaction_read_only = on;
select @@session.transaction_isolation, @@session.transaction_read_only;
start transaction;
select * from region;
insert into region values(5,'MOON','Read only?');
set transaction_read_only = off;
insert into region values(5,'MOON','Read only?');
commit;
start transaction;
select * from region;
insert into region values(5,'MOON','Read only?');
rollback;
start transaction;
select @@global.transaction_isolation, @@session.transaction_isolation;
set transaction isolation level serializable;
select @@global.transaction_isolation, @@session.transaction_isolation;
set session transaction isolation level serializable;
```

```sql
select @@global.transaction_isolation, @@session.transaction_isolation;
commit;
```

结果如下：



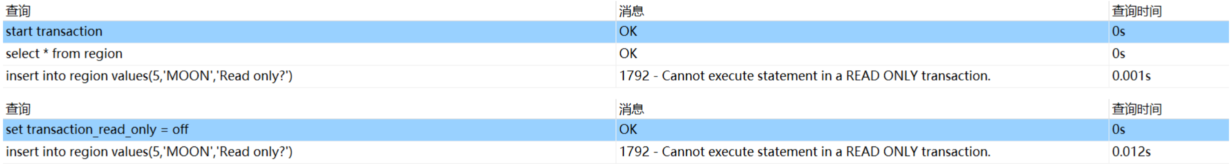
@@global.transaction_is	@@global.transaction_read_only
REPEATABLE-READ	0

@@global.transaction_is	@@global.transaction_re
SERIALIZABLE	1

@@session.transaction_i	@@session.transaction_read_only
REPEATABLE-READ	0

@@session.transaction_i	@@session.transaction_read_only
READ-COMMITTED	1

图 34: T2 执行结果（1）

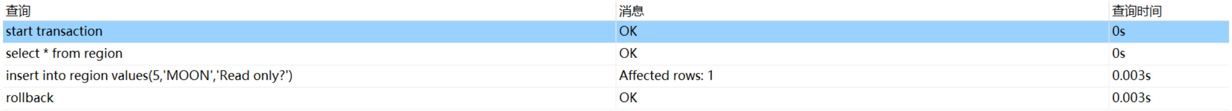


查询	消息	查询时间
start transaction	OK	0s
select * from region	OK	0s
insert into region values(5,'MOON','Read only?')	1792 - Cannot execute statement in a READ ONLY transaction.	0.001s

查询	消息	查询时间
set transaction_read_only = off	OK	0s
insert into region values(5,'MOON','Read only?')	1792 - Cannot execute statement in a READ ONLY transaction.	0.012s

图 35: T2 执行结果（2）

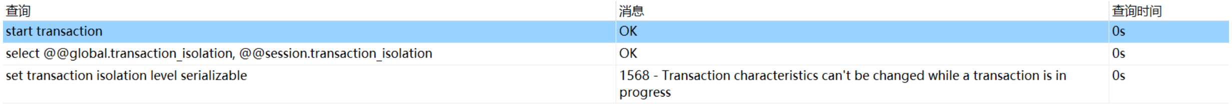
可以发现，在只读事务中，无法插入数据。



查询	消息	查询时间
start transaction	OK	0s
select * from region	OK	0s
insert into region values(5,'MOON','Read only?')	Affected rows: 1	0.003s
rollback	OK	0.003s

图 36: T2 执行结果（3）

可以发现，只有当当前事务被提交后，对只读属性的修改才会生效。



查询	消息	查询时间
start transaction	OK	0s
select @@global.transaction_isolation, @@session.transaction_isolation	OK	0s
set transaction isolation level serializable	1568 - Transaction characteristics can't be changed while a transaction is in progress	0s

图 37: T2 执行结果（4）

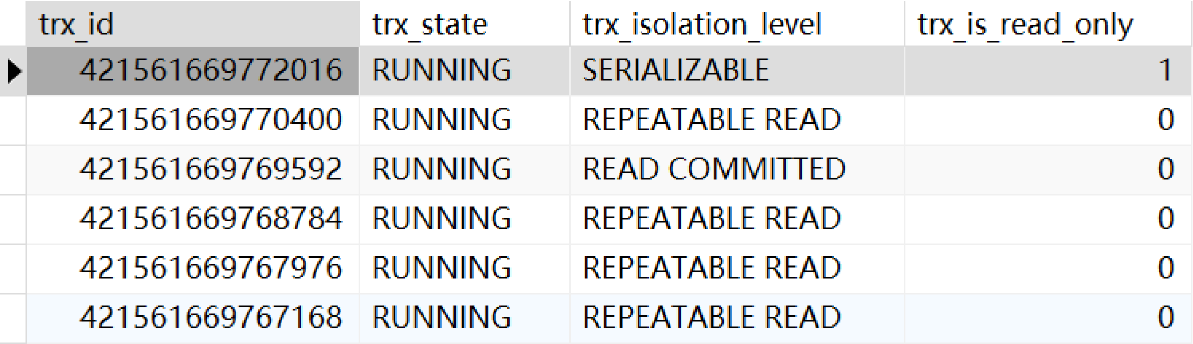
事务正在执行时，不能修改当前事务的隔离级别。

分别在不同客户端中执行下列语句，关注如何查询事务的状态；

```sql
set innodb_lock_wait_timeout = 600; -- T1
set innodb_lock_wait_timeout = 600; -- T2
set innodb_lock_wait_timeout = 600; -- T3
set session transaction isolation level read committed; -- T1
start transaction; -- T1
set session transaction isolation level repeatable read; -- T2
start transaction; -- T2
set session transaction isolation level serializable;
start transaction; -- T3
select trx_id, trx_state, trx_isolation_level, trx_is_read_only from
information_schema.innodb_trx; -- T1
select * from performance_schema.processlist; -- T1
select * from performance_schema.data_locks; -- T1
select * from region; -- T1
select * from nation limit 5; -- T2
select * from customer limit 5; -- T3
select trx_id, trx_state, trx_isolation_level, trx_is_read_only from
information_schema.innodb_trx; -- T1
select * from performance_schema.processlist; -- T1
select engine_transaction_id, thread_id, object_schema, object_name,
lock_type, lock_mode, lock_data from performance_schema.data_locks; -- T1
insert into nation values(8888,'TEST',1,'It is a test.'); -- T1
update nation set n_comment = 'It is a test.' where n_nationkey = 0; -- T1
insert into customer
values(99999,'Nobody','Nowhere',10,'12345678',3.14,'BUILDING','It is a
test.'); -- T2
update customer set c_comment = 'It is a test.' where c_custkey = 1; -- T2
```

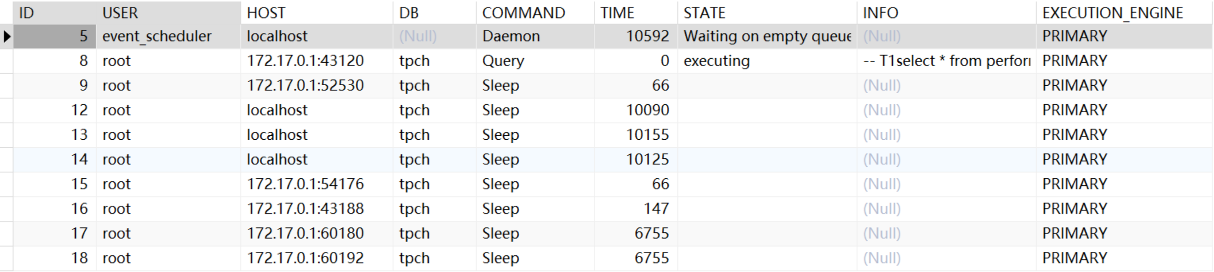
```sql
insert into region values(5,'MOON','It is a test.'); -- T3
update region set r_comment = 'It is a test.' where r_regionkey = 0; -- T3
select trx_id, trx_state, trx_isolation_level, trx_is_read_only from
information_schema.innodb_trx; -- T1
select * from performance_schema.processlist; -- T1
select engine_transaction_id, thread_id, object_schema, object_name,
lock_type, lock_mode, lock_data from performance_schema.data_locks; -- T1
SELECT r.trx_id waiting_trx_id, r.trx_mysql_thread_id waiting_thread,
r.trx_query waiting_query, b.trx_id blocking_trx_id, b.trx_mysql_thread_id
blocking_thread, b.trx_query blocking_query FROM
performance_schema.data_lock_waits w INNER JOIN
information_schema.innodb_trx b ON b.trx_id =
w.blocking_engine_transaction_id INNER JOIN information_schema.innodb_trx r
ON r.trx_id = w.requesting_engine_transaction_id; -- T1
SELECT waiting_trx_id, waiting_pid, waiting_query, blocking_trx_id,
blocking_pid, blocking_query FROM sys.innodb_lock_waits; -- T1
rollback; -- T3
rollback; -- T2
rollback; -- T1
```

结果如下：



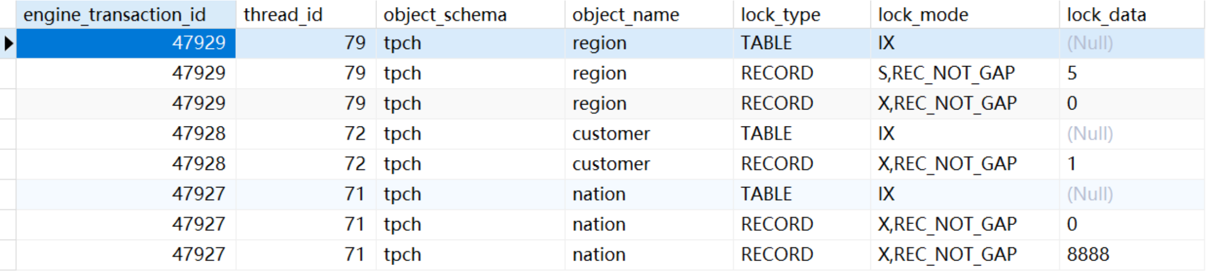
trx_id	trx_state	trx_isolation_level	trx_is_read_only
421561669772016	RUNNING	SERIALIZABLE	1
421561669770400	RUNNING	REPEATABLE READ	0
421561669769592	RUNNING	READ COMMITTED	0
421561669768784	RUNNING	REPEATABLE READ	0
421561669767976	RUNNING	REPEATABLE READ	0
421561669767168	RUNNING	REPEATABLE READ	0

图 38: 执行结果（1）



ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO	EXECUTION_ENGINE
5	event_scheduler	localhost	(Null)	Daemon	10592	Waiting on empty queue	(Null)	PRIMARY
8	root	172.17.0.1:43120	tpch	Query	0	executing	-- T1select * from perfor	PRIMARY
9	root	172.17.0.1:52530	tpch	Sleep	66		(Null)	PRIMARY
12	root	localhost	tpch	Sleep	10090		(Null)	PRIMARY
13	root	localhost	tpch	Sleep	10155		(Null)	PRIMARY
14	root	localhost	tpch	Sleep	10125		(Null)	PRIMARY
15	root	172.17.0.1:54176	tpch	Sleep	66		(Null)	PRIMARY
16	root	172.17.0.1:43188	tpch	Sleep	147		(Null)	PRIMARY
17	root	172.17.0.1:60180	tpch	Sleep	6755		(Null)	PRIMARY
18	root	172.17.0.1:60192	tpch	Sleep	6755		(Null)	PRIMARY

图 39: 执行结果（2）



engine_transaction_id	thread_id	object_schema	object_name	lock_type	lock_mode	lock_data
47929	79	tpch	region	TABLE	IX	(Null)
47929	79	tpch	region	RECORD	S,REC_NOT_GAP	5
47929	79	tpch	region	RECORD	X,REC_NOT_GAP	0
47928	72	tpch	customer	TABLE	IX	(Null)
47928	72	tpch	customer	RECORD	X,REC_NOT_GAP	1
47927	71	tpch	nation	TABLE	IX	(Null)
47927	71	tpch	nation	RECORD	X,REC_NOT_GAP	0
47927	71	tpch	nation	RECORD	X,REC_NOT_GAP	8888

图 40: 执行结果（3）

2.2 异常与隔离级别

2.2.1 Dirty Write

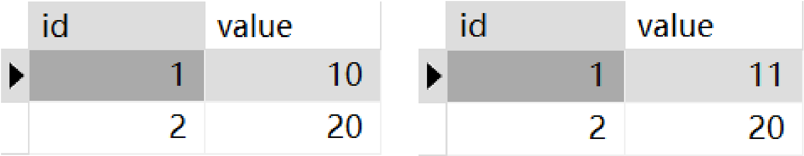
```sql
drop table if exists test_dirty_write;
create table test_dirty_write(id int primary key, value int) engine=innodb;
insert into test_dirty_write(id, value) values(1, 10), (2, 20);

set session transaction isolation level read uncommitted; begin; -- T1
set session transaction isolation level read uncommitted; begin; -- T2

update test_dirty_write set value = 11 where id = 1; -- T1
update test_dirty_write set value = 12 where id = 1; -- T2 (blocked here)

commit; -- T2

commit; -- T1

select * from test_dirty_write; -- Shows 1 => 12, 2 => 20
```

在四种隔离级别下，均被阻塞在 T2 试图更改时，没有异常出现。

2.2.2 Dirty Read

```sql
drop table if exists test_dirty_read;
create table test_dirty_read(id int primary key, value int) engine=innodb;
insert into test_dirty_read(id, value) values(1, 10), (2, 20);

set session transaction isolation level read uncommitted; begin; -- T1
set session transaction isolation level read uncommitted; begin; -- T2

-- T1更新数据，但不提交
update test_dirty_read set value = 11 where id = 1; -- T1

-- T2读取未提交的数据
select * from test_dirty_read; -- T2, Shows 1 => 11, 2 => 20

-- T1回滚更改
rollback; -- T1

-- T2再次读取数据
select * from test_dirty_read; -- T2, Shows 1 => 10, 2 => 20

commit; -- T2
```

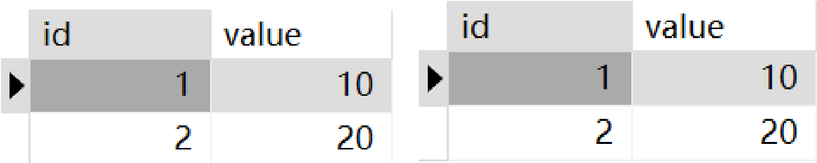
只有在 READ UNCOMMITTED 隔离级别下，出现了脏读。



id	value
1	10
2	20

id	value
1	11
2	20

图 41: READ UNCOMMITTED 隔离级别下的脏读



id	value
1	10
2	20

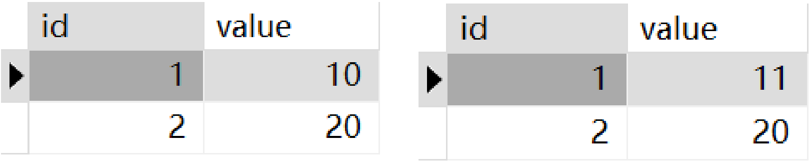
id	value
1	10
2	20

图 42: 其他隔离级别下无异常

2.2.3 Non-Repeatable Read

```sql
drop table if exists test_non_repeatable_read;
create table test_non_repeatable_read(id int primary key, value int) engine=innodb;
insert into test_non_repeatable_read(id, value) values(1, 10), (2, 20);

set session transaction isolation level read uncommitted; begin; -- T1
set session transaction isolation level read uncommitted; begin; -- T2

-- T1读取数据
select * from test_non_repeatable_read; -- T1, Shows 1 => 10, 2 => 20

-- T2更新数据并提交
update test_non_repeatable_read set value = 11 where id = 1; -- T2
commit; -- T2

-- T1再次读取同一行的数据
select * from test_non_repeatable_read where id = 1; -- T1, Shows 1 => 11

commit; -- T1
```

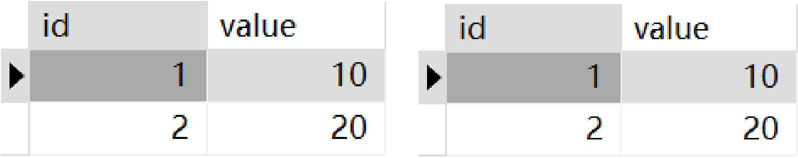
READ UNCOMMITTED 隔离级别和 READ COMMITTED 隔离级别下，出现了不可重复读。



id	value
1	10
2	20

id	value
1	11
2	20

图 43: READ UNCOMMITTED 和 READ COMMITTED 隔离级别下的不可重复读



id	value
1	10
2	20

id	value
1	10
2	20

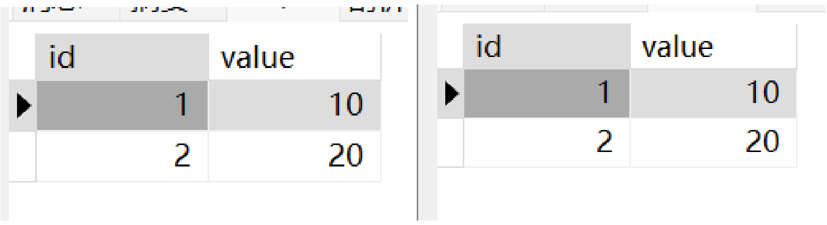
图 44: 其他隔离级别下无异常

2.2.4 Phantom Read

```sql
drop table if exists test_phantom_read;
create table test_phantom_read(id int primary key, value int) engine=innodb;
insert into test_phantom_read(id, value) values(1, 10), (2, 20);
```

```sql

set session transaction isolation level read committed; begin; -- T1
set session transaction isolation level read committed; begin; -- T2

-- T1读取数据
select * from test_phantom_read; -- T1, Shows 1 => 10, 2 => 20

-- T2插入新数据并提交
insert into test_phantom_read(id, value) values(3, 30); -- T2
commit; -- T2

-- T1再次读取数据，发现多了一行新的数据
select * from test_phantom_read; -- T1, Shows 1 => 10, 2 => 20, 3 => 30

commit; -- T1
```

除了 SERIALIZABLE 隔离级别外，其他隔离级别下，出现了幻读。



id	value
1	10
2	20

id	value
1	10
2	20

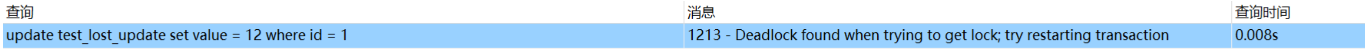
图 45: READ UNCOMMITTED, READ COMMITTED 和 REPEATABLE READ 隔离级别下的幻读

2.2.5 Lost Update

```sql
drop table if exists test_lost_update;
create table test_lost_update(id int primary key, value int) engine=innodb;
insert into test_lost_update(id, value) values(1, 10), (2, 20);

set session transaction isolation level REPEATABLE READ; begin; -- T1
set session transaction isolation level REPEATABLE READ; begin; -- T2

-- T1读取数据
select value from test_lost_update where id = 1; -- T1, Shows 10

-- T2读取数据
select value from test_lost_update where id = 1; -- T2, Shows 10

-- T1更新数据
update test_lost_update set value = 11 where id = 1; -- T1
```

```

-- T2更新数据
update test_lost_update set value = 12 where id = 1; -- T2

-- 提交T1
commit; -- T1

-- 提交T2
commit; -- T2

-- 检查表中的数据
select * from test_lost_update; -- Shows 1 => 12, 2 => 20
```

除了 SERIALIZABLE 隔离级别外，其他隔离级别下，均在 T2 更新数据时，被阻塞。而 SERIALIZABLE 隔离级别下，T1 更新数据时就发生阻塞，T2 更新数据时，报出了死锁异常。



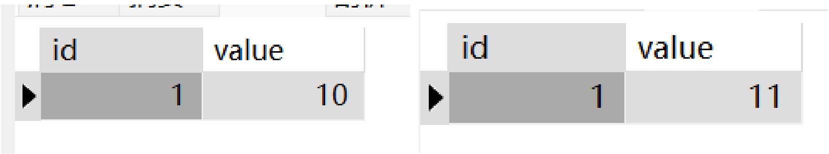
查询	消息	查询时间
update test_lost_update set value = 12 where id = 1	1213 - Deadlock found when trying to get lock; try restarting transaction	0.008s

图 46: 死锁异常

2.2.6 Read Skew

```
drop table if exists test_read_skew;
create table test_read_skew(id int primary key, value int) engine=innodb;
insert into test_read_skew(id, value) values(1, 10), (2, 20);

set session transaction isolation level REPEATABLE READ; begin; -- T1
set session transaction isolation level REPEATABLE READ; begin; -- T2

-- T1读取数据
select * from test_read_skew where id = 1; -- T1, Shows 1 => 10

-- T2更新数据并提交
update test_read_skew set value = 11 where id = 1; -- T2
commit; -- T2

-- T1再次读取同一行的数据
select * from test_read_skew where id = 1; -- T1, Shows 1 => 11

commit; -- T1
```

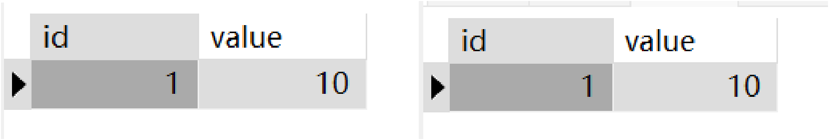
READ UNCOMMITTED 隔离级别和 READ COMMITTED 隔离级别下，出现了读偏斜。



id	value
1	10

id	value
1	11

图 47: READ UNCOMMITTED 和 READ COMMITTED 隔离级别下的读偏斜



id	value
1	10

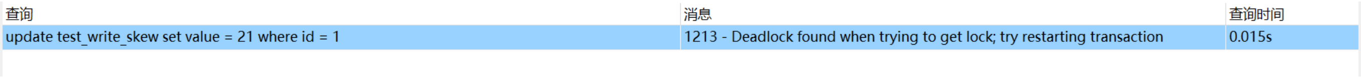
id	value
1	10

图 48: 其他隔离级别下无异常

2.2.7 Write Skew

```sql
drop table if exists test_write_skew;
create table test_write_skew(id int primary key, value int) engine=innodb;
insert into test_write_skew(id, value) values(1, 10), (2, 20);

set session transaction isolation level REPEATABLE READ; begin; -- T1
set session transaction isolation level REPEATABLE READ; begin; -- T2

select * from test_write_skew where id in (1, 2); -- T1, Shows 1 => 10, 2 => 20
select * from test_write_skew where id in (1, 2); -- T2, Shows 1 => 10, 2 => 20

update test_write_skew set value = 11 where id = 1; -- T1
update test_write_skew set value = 21 where id = 1; -- T2

commit; -- T1
commit; -- T2
```

除 SERIALIZABLE 隔离级别外，其他隔离级别下，均在 T2 更新数据时，被阻塞。而 SERIALIZABLE 隔离级别下，T1 更新数据时就发生阻塞，T2 更新数据时，报出了死锁异常。



查询	消息	查询时间
update test_write_skew set value = 21 where id = 1	1213 - Deadlock found when trying to get lock; try restarting transaction	0.015s

图 49: 死锁异常

Anomalies/Isolation Level	READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE
Dirty Write	✓	✓	✓	✓
Dirty Read	×	✓	✓	✓
Non-repeatable Read	×	×	✓	✓
Phantom Read	×	×	×	✓
Lost Update	×	×	×	✓
Read Skew	×	×	✓	✓
Write Skew	×	×	×	✓

表 1: Isolation Levels and Anomalies

3 存在的问题及解决方案

1. 显示当前为只读事务时，无法插入数据；

   解决方案：在事务开始前设置事务为读写事务。

4 实验小结

通过本次实验，我学习了 MySQL 数据库管理系统中事务处理相关的设置和基本操作，以及事务处理过程中不同事务隔离级别所对应的异常，能够根据应用场景选择合适的事务隔离级别。