# 第六次实验

22375080 杨佳宇轩

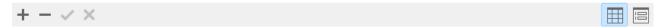
# Task 1

## Q1

• 语句 2 输出

开启事务之后将 id = 1 的 money + 1000, 查询得到 2000

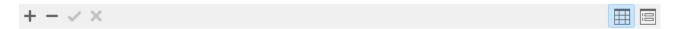




• 语句 4 输出

开启新事务,查询得到 id = 1 的结果仍然为 2000





• 语句 5 输出

session 1 中进行了回滚,得到的结果较最初不会发生变化

1	言息	摘要	结果1	剖	折	状态
	id	ı	name		mon	iey
Þ		1 t	om			1000



# Q2

#### read committed

● 语句 2 输出

事务开启之后,更新后读取





• 语句 4 输出

由于是 read committed ,事务结果未提交,读原来的

1	息	摘要	结果1	剖析	状态
	id		name	mo	ney
Þ		1	tom		1000



● 语句 5 输出

回滚后结果不变

ſ	息	摘要	结果 1	剖	折	状态
	id	ı	name		mon	iey
Þ		1 t	om			1000



# repeatable read

● 语句 2 输出

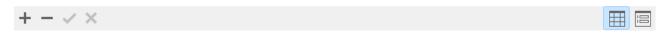




● 语句 4 输出

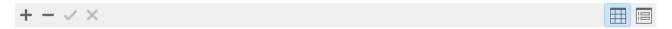
事务结果未提交,读原来的





• 语句 5 输出

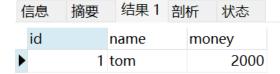
1	息	摘要	结果1	剖	折	状态
	id	ı	name		mon	iey
Þ		1 t	om			1000

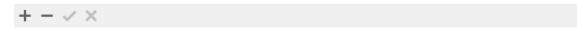


#### serializable

• 语句 2 输出

同理



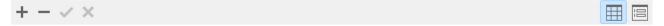


• 语句 4 输出

没有输出,忙等待

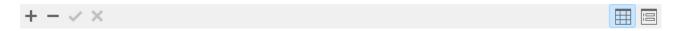
session 1 提交之后读取

1	息	摘要	结果 1	剖	析	状态	
	id		name		mon	iey	
Þ		1	tom			1000	



• 语句 5 输出

1	言息	摘要	结果1	剖	析	状态
	id	r	name		mon	iey
١		1 t	om			1000

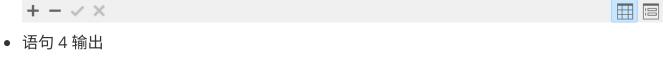


# Q3

● 语句 2 输出

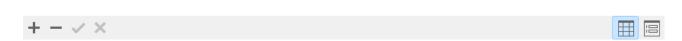
开启事务,正常读出





session 2 提交之后 session 1 读入更新结果





# Q4

# repeatable read

● 语句 2 输出

信息	摘要	结果1	剖析	状态
id	ı	name	moi	ney
•	2	bob		0





• 语句 4 输出

由于 session 1 开始之前标记了隔离级别为 repeatable read ,因此无法看到 session 2 提交的更新,读取原来的 0

信息	摘要	结果1	剖	析	状态	
id	r	name		mor	ney	
•	2 k	oob				0

#### + - $\checkmark$ $\times$



#### serializable

• 语句 2 输出



### + - $\checkmark$ $\times$



• 语句 4 输出

由于隔离级别是 serializable, session 2 在 session 1 未提交之前无法开始事务,因此 session 1 读取结果不变

1	言息	摘要	结果 1	剖	斤	状态	
	id	ı	name		mon	iey	
Þ		2	oob				0

# + - - - ×

# Q5

• T2 时刻

由于给 account 表加上了 S 锁,因此 session 2 无法进行修改后

需要进行忙等

#### 信息 摘要

- > 查询时间: 0s
- -- set session transaction isolation level serializable; start transaction
- OK
- > 查询时间: 0s

update account set money = money+1000 where id=2

运行时间: 17.877s

• T4 时刻

session 1 提交之后可以进行修改,但由于还没有提交,数据库不变

信息	摘要	剖析	状态			
查询					信息	查询时间
set session transaction isolation level repeatable read					OK	0s
set session transaction isolation level serializable; start transaction				tion level	OK	0s
update account set money = money+1000 where id=2			noney+1000	Affected rows: 1	28.623s	

提交之后数据库改变

id	name	money
1	tom	1000
<b>)</b>	bob	1000

# Q6

• 语句 2 输出

正常读



+ -  $\checkmark$   $\times$ 



• 语句 4 输出

由于可重复读,因此不会读到 session 2 提交的更改



	id	name	money
Þ	1	tom	1000
	2	bob	0

+ -  $\checkmark$   $\times$ 



• 语句 6 输出

session 2 中插入数据,读取三个行

信息		摘要	结果1	剖析		状态
	id		name		mon	iey
١		1 1	om			1000
		2	oob			0
		3 8	alen			0

+ -  $\checkmark$   $\times$ 



• 语句 7 输出



# Q7

• 语句 2 输出



● 语句 4 输出

由于串行执行,值不会改变



# Q8

session 2 能够成功执行 insert 操作和读取新插入的行,而加入了 look in share mode 或改为 serializable 后, 在 session 1 进行提交操作前, session 2 的所有操作都会被阻塞, 无法进行

这是由于在 repeatable read 权限下, session 2 只是被限制了修改表的权限, 但是可以增加内容, 而由于快照读的存在, 即使 session 2 进行了提交操作, session 1 中读取的数据仍然是复制出来的数据库镜像数据, 即实际上数据库中已经新插入了数据但是 session 1 不会读出来

而加入了 lock in share mode 或改为 serializable 后,实际上真实对表进行了加锁操作,使得 session 2 只能读表的内容,不能修改表的内容,因此才有了上述情况产生

#### Q9

• 语句 2 输出

没有输出, 因为为序列隔离

> OK

> 查询时间: 0s

start transaction

> OK

> 查询时间: 0s

delete from account where id=1

• 语句 3 输出

由于对于 session 2 已经删除了这一行, 因此语句 3 不会影响

信息	摘要	剖析	状态		
查询				信息	查询时间
delete from account where id=1			vhere id=	Affected rows: 0	0s

# Task 2

# Q10

由于 session 1 对A表获取X锁, session 2 对于B表获取X锁 此时 session 1 向 B 表申请锁, session 2 又向 A 表申请锁 满足了死锁的必要条件, 双方都在请求对方持有的锁资源, 引发死锁 信息 摘要 状态 信息 查询时间 update tableA set columnA=2 where id=1 Deadlock found when trying to