第七次实验

22371437

张智威

Task1

Q1

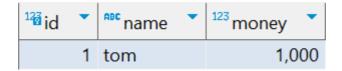
语句2

¹² d id	•	name *	¹²³ money
	1	tom	2,000

语句4



语句5

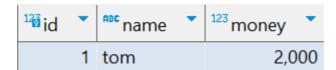


解析

将两个事务都设置成read uncommoditted,在session1进行数据更新后,session1自己先读到自己更新后的数据即 2000,随后由于session2设置可读未提交的数据,因此也读到session1更新后的数据 2000,随后session1回退后,session2再查询得到的是未处理前的数据 1000。

Q2-read committed

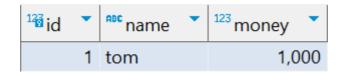
语句2



语句4



语句5

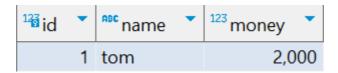


解析

将两个事务都设置成read commoditted,在session1进行数据更新后,session1自己先读到自己更新后的数据即 2000,随后由于session2设置不可读未提交的数据,因此也读到session1更新前的数据 1000,随后session1回退后,session2再查询得到的是未处理前的数据 1000。

Q2-repeatable read

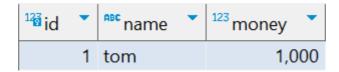
语句2



语句4



语句5

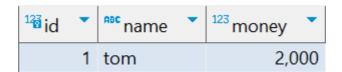


解析

原理与read committed类似。

Q2-serializable

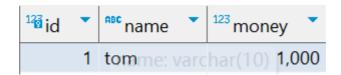
语句2



语句4

无筛选结果。

语句5

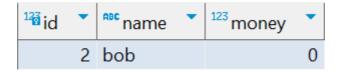


解析

将两个事务都设置成serializable,在session1进行数据更新后,session1自己先读到自己更新后的数据即 2000 ,随后由于session2&session1设置为serializable,因此session2无法读到session1正在处理的数据,随后session1回退后,session2再查询得到的是未处理前的数据 1000。

Q3

语句2



语句4

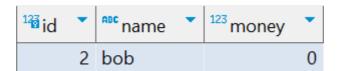


解析

由于将两个事务都设置成read uncommoditted,因此session1前面先读取到初始状态,后面session2提交了数据更改后session1再次读取就读取到了数据变化。

Q4-repeatable read

语句2



语句4



Name	Value
Updated Rows	0
Query	commit
Start time	Mon May 13 17:02:33 CST 2024
Finish time	Mon May 13 17:02:33 CST 2024

解析

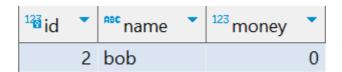
由于将两个事务都设置成repeatable read,因此session1前面先读取到初始状态,后面session2提交了数据更改时无法进行更新,session1再次读取时数据无变化。

Q4-serializable

语句2



语句4



解析

由于将两个事务都设置成serializable,因此session1前面先读取到初始状态,后面session2提交了数据更改时无法进行更新,session1再次读取时数据无变化(在此处由于只设计更新操作,serializable与repeatable read类似)。

Q5

T2时刻

在session1进行提交操作前,session2的语句2一直处于停滞状态,没有响应,在session1提交之后,session2的语句2成功操作并返回。

Name	Value
Updated Rows	1
Query	update account set money = money+1000 where id=2
Start time	Mon May 13 18:12:03 CST 2024
Finish time	Mon May 13 18:12:03 CST 2024

T4时刻

session2即刻返回成功的消息,返回的内容与上图相同。

解析

由于session1和session2的状态都设置成repeatable read,导致session1在读取数据后,session2希望对数据进行修改无法完成,等待session1提交后释放锁,session2才能实现数据更新,而session1提交后session2再次尝试更新,由于数据没有上锁,更新的步骤立刻得以完成。

Q6

语句2

¹² 6 id ▼	name T	¹²³ money
1	tom	1,000
2	bob	0

语句4

¹² 6 id ▼	name T	¹²³ money
1	tom	1,000
2	bob	0

语句6

¹ã id	•	name T	¹²³ money
	1	tom	1,000
	2	bob	0
	3	alen	0

语句7

¹² 2 id ▼	name T	¹²³ money
1	tom	1,000
2	bob	0
3	alen	1,000

Q7

语句2

¹² 6 id ▼	name T	¹²³ money
1	tom	1,000
2	bob	0

语句4

¹ã id ▼	name *	123 money
1	tom	1,000
2	bob	0

Q8

在repeatable read的权限下, session2是能够成功的执行insert操作和读取到新插入的行的, 而加了 lock in share mode或改为serializable后, 在session1进行提交操作前, session2的所有操作都被阻塞, 无法进行。

这是由于在repeatable read的权限下,session2只是被限制了修改表的权限,但是能够新增表中的内容,而由于快照读的存在,即使session2进行了提交操作,session1中读取到的数据仍是复制出的数据库镜像中的数据,即实际上数据库中已经新插入了数据但是session1不会读出来。

而加了lock in share mode或改为serializable后,实际上真实对表进行了加锁的操作,使得session2只能读表中的内容,不能修改表中的内容,因此才有上述情况产生。

Q9

语句2

Name	Value
Updated Rows	1
Query	delete from account where id=1
Start time	Mon May 13 18:41:31 CST 2024
Finish time	Mon May 13 18:41:45 CST 2024

语句3

Name	Value
Updated Rows	0
Query	delete from account where id=1
Start time	Mon May 13 18:42:23 CST 2024
Finish time	Mon May 13 18:42:23 CST 2024

解析

在session1进行了update语句后加上排他锁,导致session2的delete语句被阻塞,直到session1进行提交后,才进行删除操作,而第二次删除操作由于先前已经进行了删除于是没有行被删除。

Task2

Q10

由于session1先对A表进行更新操作对A表加上排他锁,session2再对B表进行更新操作对B表加上排他锁,随后session1申请对B更新,需要等待session2释放B的锁,session2申请对A更新,需要等待session1释放A的锁,二者互相需要对方释放锁,因此陷入死锁。