RR行为分析

PG

PG的可重复读(RR)和提交读(RC)都使用Snapshot Isolation的并发控制机制来实现事务的隔离，Snapshot Isolation提供的抽象是，事务会对数据库中某一时刻的数据建立一个快照，事务的read操作都从这份快照中获取结果。RR在事务第一次read时建立一份快照，之后的read操作访问这同一份快照，RC在每次read之前都会建立一份快照。

RR提供的抽象就是重复读取同一份快照，除非当前事务自己改变了快照的内容，否则该事务总是会读到相同的数据。

```
# Session A 在此时进行了读取，建立了快照，快照中有（1，2）、（2，3）这两行元组
> [Time0, SessionA]
>> Begin;
>> set transaction isolation level repeatable read;
>> Select * from t where a=1;


# Session B的操作提交后也不会影响Session A的快照
> [Time1, SessionB]
>> Begin;
>> set transaction isolation level read committed;
>> Delete from t where a=2;
>> Commit;

# 当Session A成功插入新的行（2，4）之后，快照中的确应该包含（1，2）、（2，3）和
 (2，4)
> [Time2, SessonA]
>> Insert into t values(2,4);
>> Select * from t where a=2;
```

现在可能让人困惑的问题在于，表在column a上有主键约束，但是为什么Session A返回的结果中却存在两行的column a都等于2?

简单来说，因为RR的事务在SQL标准中只要求可重复读，再没有其余要求，因此RR的事务的执行结果完全有可能出现不一致的情况，在这个例子中，也就是返回了违背了主键约束的结果。这种不一致来源于Snapshot Isolation并不能保证并发事务的执行是可序列化的，事务并不具备相互隔离执行的特性。只有当事务的执行满足可序列化，事务的一致性（Consistency in ACID）才能被保证，其余情况都可能出现不一致的现象。在SQL标准中，只要求在Serializable的隔离级别能够保证事务执行的可序列化，因此此处PG返回的结果完全是合法的，符合RR要求的。

从本质上说，RR中的不一致来源于没有被处理的读写冲突（read write conflicts），这一话题过于复杂，后续有机会可以分享。

为了避免事务的不一致，请使用Serializable的隔离级别。

MySQL

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> Select * from t where a=1;
+---+------+
| a | b    |
+---+------+
| 1 |    2 |
+---+------+
1 row in set (0.00 sec)

mysql> Insert into t values(2,4);
Query OK, 1 row affected (0.01 sec)

mysql> Select * from t where a=2;
+---+------+
| a | b    |
+---+------+
| 2 |    4 |
+---+------+
1 row in set (0.00 sec)
```

1.

2. Session B deletes (2,3)

3.

4.

MySQL中的行为符合邮件中的描述，但这并不是RR定义中应该有的行为

Oracle

Oracle中只有RC和Serializable的隔离级别

Oracle的事务隔离级别:

Oracle提供read committed和serializable，并提供了一个非SQL标准的read-only级别。

Read commit:

①这是oracle默认的隔离级别;

②保证了不会出现脏读,但是**允许出现**非重复读和幻读。

Serializable:

①serializable使事务看起来一个接着一个地顺序执行(从效果上可以这样理解)

②只能看见在本事务开始前其他事务提交的更改和本事务中所做的更改.

③保证不会出现脏读、不可重复读和幻读。

④Serializable隔离级别提供了read-only事务所提供的读一致性，同时又允许DML(update/insert/delete)操作。