一. 实验目的

- 理解事务并发中不一致的问题，以及通过设置隔离级别解决不一致问题。

二. 实验内容

- **事务并发不一致问题:**
  - **读"脏"数据**：一个事务读取另一个事务尚未提交的数据引起。
  - **不可重复读**：事务T1读取数据a后，事务T2对数据a进行更新，事务T1再次读取，无法读取前一次的结果。
  - **幻象读**：事务T1两次查询过程中，事务T2对数据进行插入或删除，导致事务T1两次查询的记录数不一致。
- **事务隔离级别:**
  - READ UNCOMMITTED(未提交读，读脏)
  - READ COMMITTED(已提交读，不读脏，但允许不重复读，SQL默认级别)
  - REPEATABLE READ(可重复读，禁止读脏和不重复读，但允许幻象读)
  - SERIALIZABLE(可串行化，最高级别，事务不能并发，只能串行)

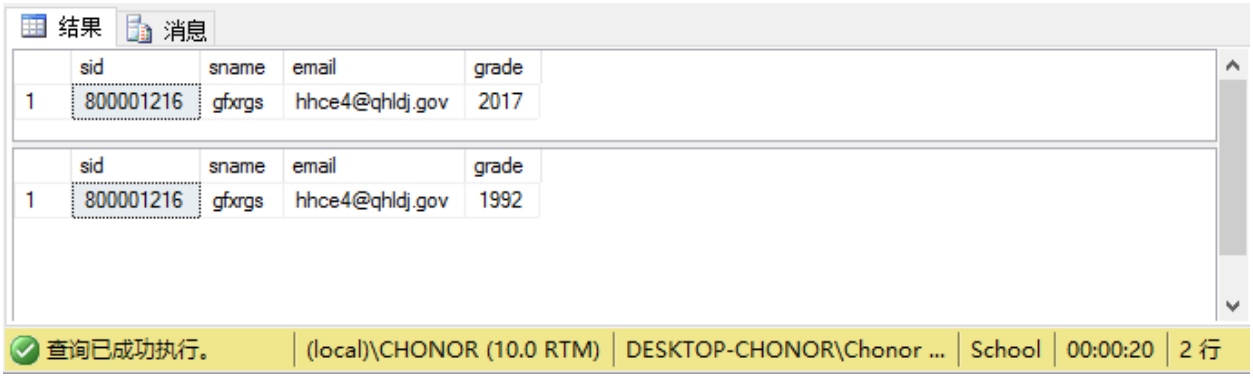
三. 实验结果

1. **设置"未提交读"隔离级别（READ UNCOMMITTED），在students表上演示读"脏"数据**

   事务1为执行更新，延迟20秒后回滚事务1.

```
BEGIN TRAN
    UPDATE STUDENTS
    SET grade = '2017'
    WHERE sid = '800001216'
    WAITFOR DELAY '00:00:20'
    SELECT * FROM STUDENTS WHERE sid = '800001216'
ROLLBACK TRAN
SELECT * FROM STUDENTS WHERE sid = '800001216'
```

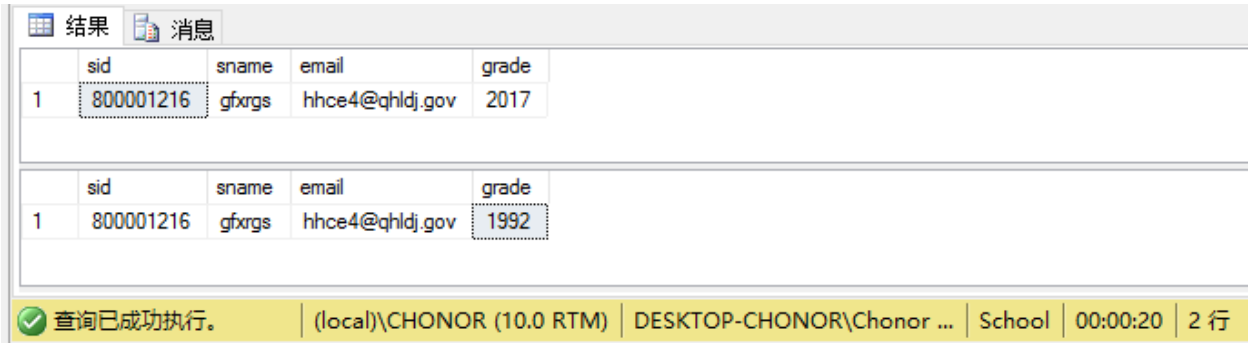
   单独执行事务1效果



   查询2，先查询一次，延迟20秒后再次查询

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM STUDENTS WHERE sid = '800001216'
IF @@ROWCOUNT <> 0
BEGIN
    WAITFOR DELAY '00:00:20'
    SELECT * FROM STUDENTS WHERE sid = '800001216'
END
```

此时先执行事务1，然后立即执行查询2，查询2结果如下



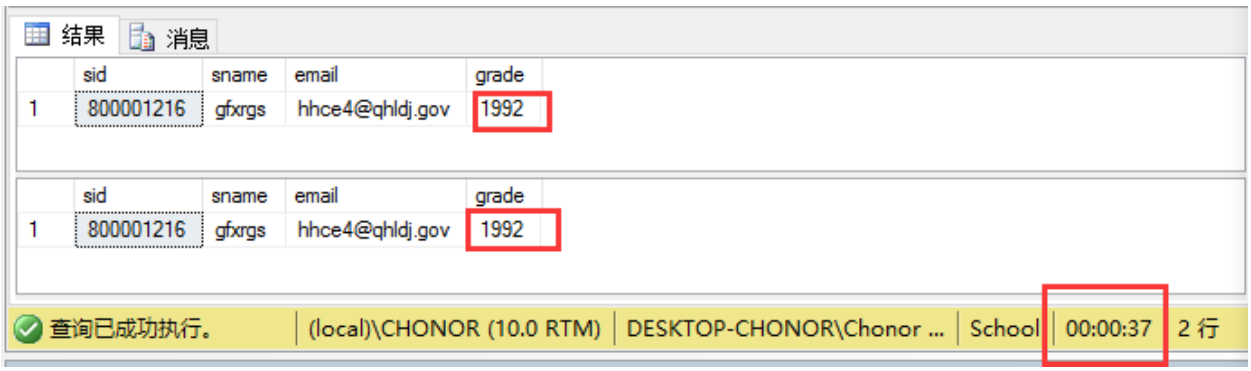
此时我们看到第一次查询结果为读“脏”数据，查询的是事务1没有提交前的数据，延迟20s后因为因为事务1已经进行回滚，此时出现了“不可重复读”，无法读取到和第一次查询一致的结果。

2. **设置“提交读”隔离级别(READ COMMITTED)，在students表上演示避免读“脏”数据。**

   我们将查询2的READ UNCOMMITTED改为READ COMMITTED

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
SELECT * FROM STUDENTS WHERE sid = '800001216'
IF @@ROWCOUNT <> 0
BEGIN
    WAITFOR DELAY '00:00:20'
    SELECT * FROM STUDENTS WHERE sid = '800001216'
END
```

此时先执行事务1在执行查询2的结果如下：



此时两次查询的结果为事务1提交后的结果，同时我们也可以看到查询2的时间花费大于20s,因为这是要等到事务1执行完毕后才执行，所以这里包含了部分事务1的执行时间。

3. **设置“可重复读”隔离级别(REPEATABLE READ)，在students表上演示避免读“脏”数据、不可重复读，但不能避免幻象读。**

   首先我们先向数据库中插入一条新的学生信息

```
INSERT INTO STUDENTS
VALUES ('300000000','test','test@gmail.com',2017)
```

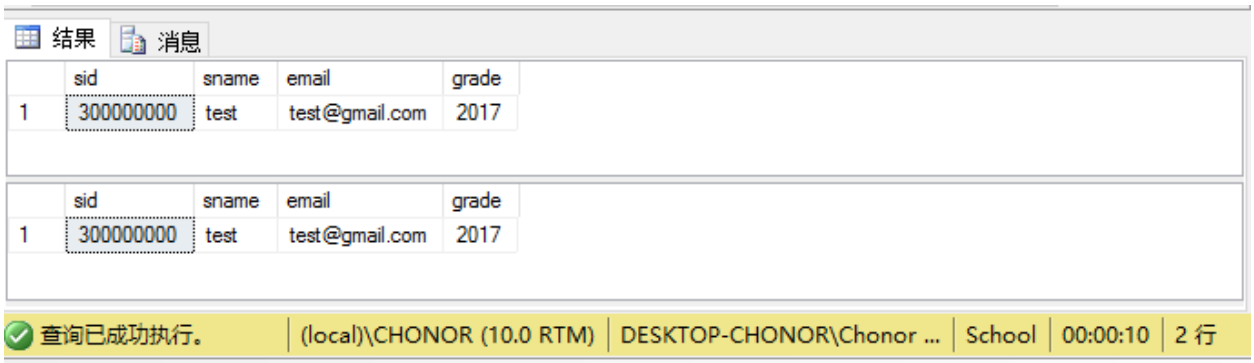
设置事务2为，从数据库中读取sid='300000000'的学生，延迟10s后再次读出。

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
    SELECT * FROM STUDENTS WHERE sid = '300000000'
    IF @@ROWCOUNT<>0
    BEGIN
        WAITFOR DELAY '00:00:10'
        SELECT * FROM STUDENTS WHERE sid = '300000000'
    END
ROLLBACK TRAN
```

设置查询3，为删掉sid='300000000'的学生

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
DELETE FROM STUDENTS WHERE sid = '300000000'
```

此时我们先执行事务2 紧接着执行查询3，结果如下：



此时出现了两次查询结果相同，但是实际上 sid = '300000000'的记录已经被删除，此时避免读“脏”数据、不可重复读，但是发生了幻象读。

4. **设置 “可串行化”隔离级别(SERIALIZABLE)，在students表上演示防止其他用户在事务提交之前更新数据。**

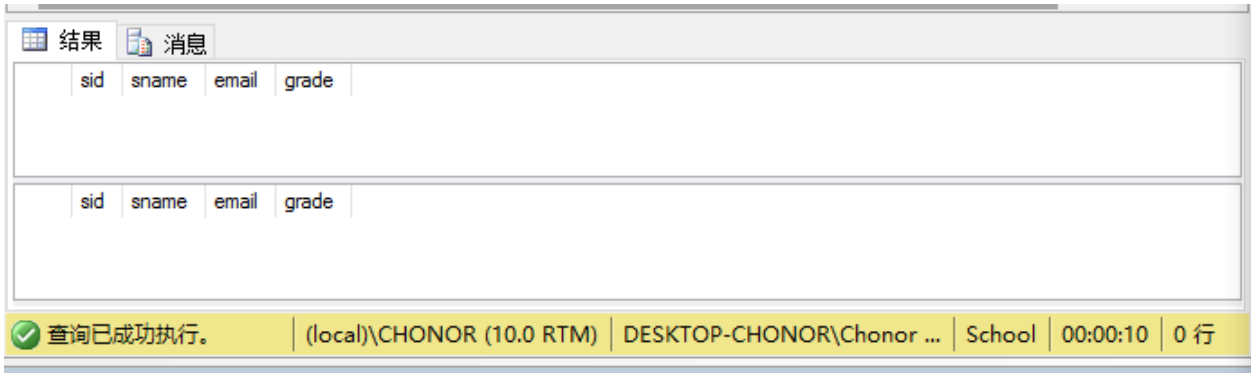
   设置事务3为：从数据库中查询sid = '300000000'的记录，延迟10s后再次查询

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
    SELECT * FROM STUDENTS WHERE sid = '300000000'
    WAITFOR DELAY '00:00:10'
    SELECT * FROM STUDENTS WHERE sid = '300000000'
ROLLBACK TRAN
```

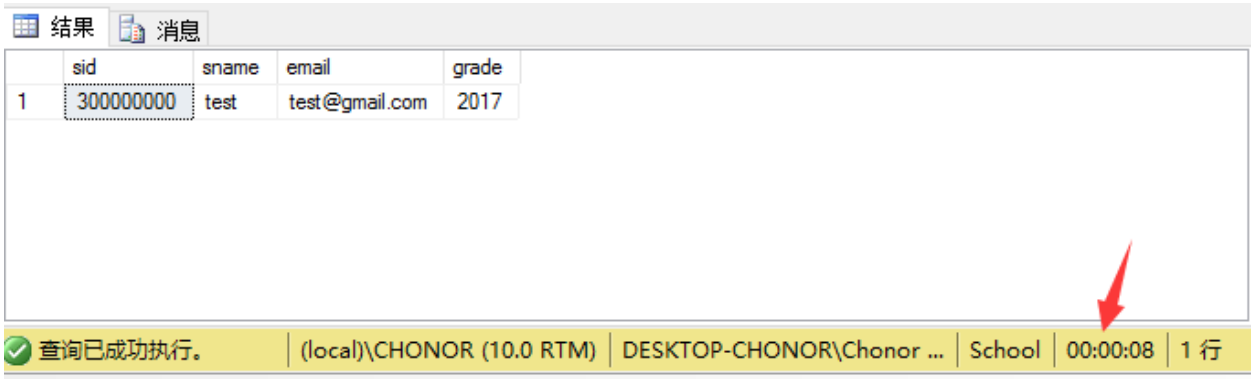
设置查询4为：向student表中插入数据

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
INSERT INTO STUDENTS
VALUES ('300000000','test','test@gmail.com',2017)
SELECT * FROM STUDENTS WHERE sid = '300000000'
```

先执行事务3，然后立即执行查询4，事务3结果如下：



查询4执行结果如下



我们可以看到，事务3查询结果全部为空，同时查询4的执行时间较长没有立即执行，查询4的执行需要等待事务3完成，此时查询4中包含了等待事务3查询完成的时间，说明此时事务的执行时串行的，所以事务3在执行的过程中防止查询4插入数据，所以查询结果为空。

四. 实验感想

这次实验是在实践事务的并发性和隔离级别，主要就是通过SET TRANSACTION ISOLATION LEVEL设置隔离级别，从而实现对于事务并发性的控制，实验中实践这个设置隔离级别的效果也很明显，直接通过查询的结果和所花的时间我们就可直接看出不同的隔离级别的不同影响，同时也能观察到事务并发性不一致的问题