

数据库技术第二次作业

1. Sqlite

- (1) 隔离级别: sqlite 的隔离级别只能为 TRANSACTION_SERIALIZABLE

下图为尝试了设置其他隔离级别后抛出的异常, 也验证了上述情况。



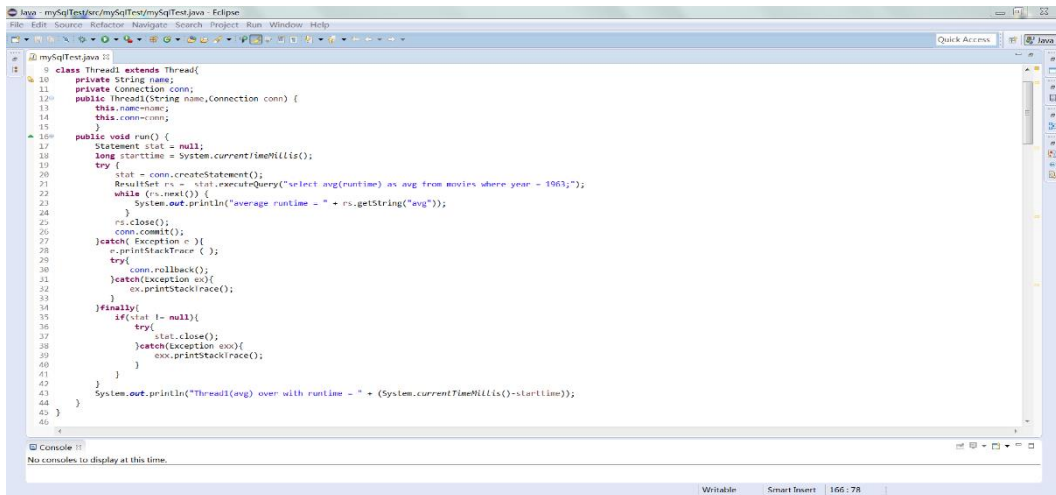
- (2) 封锁粒度: 由于 sqlite 的隔离级别只能是 TRANSACTION_SERIALIZABLE

故粒度只能为表锁

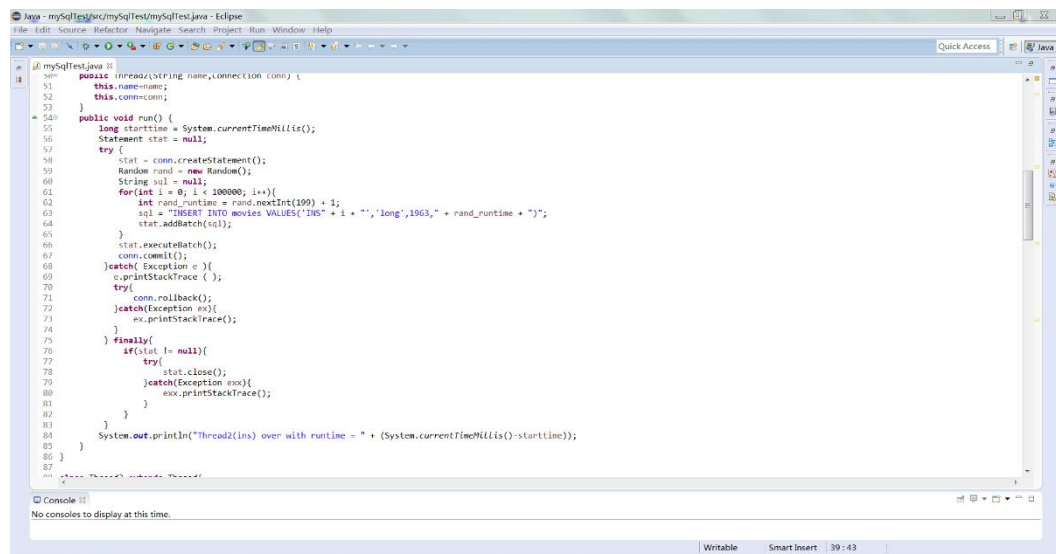
- (3) 查询代码:

- a) 问题 1:

查询平均数的线程代码:



插入数据的线程代码:



- b) 问题 2:

查询平均数的线程代码:

```

132     }
133     }
134     }
135     }
136     }
137     }
138     }
139     }
140     }
141     }
142     }
143     }
144     }
145     }
146     }
147     }
148     }
149     }
150     }
151     }
152     }
153     }
154     }
155     }
156     }
157     }
158     }
159     }
160     }
161     }
162     }
163     }
164     }
165     }
166     }
167     }
168     }
169     }
170     }
171     }
172     }
173     }
174     }
175     }
176     }
177     }
178     }
179     }
180     }
181     }
182     }
183     }
184     }
185     }
186     }
187     }
188     }
189     }
190     }
191     }
192     }
193     }
194     }
195     }
196     }
197     }
198     }
199     }
200     }
201     }
202     }
203     }
204     }
205     }
206     }
207     }
208     }
209     }
210     }
211     }
212     }
213     }
214     }
215     }
216     }
217     }
218     }
219     }
220     }
221     }
222     }
223     }
224     }
225     }
226     }
227     }
228     }
229     }
230     }
231     }
232     }
233     }
234     }
235     }
236     }
237     }
238     }
239     }
240     }
241     }
242     }
243     }
244     }
245     }
246     }
247     }
248     }
249     }
250     }
251     }
252     }
253     }
254     }
255     }
256     }
257     }
258     }
259     }
260     }
261     }
262     }
263     }
264     }
265     }
266     }
267     }
268     }
269     }
270     }
271     }
272     }
273     }
274     }
275     }
276     }
277     }
278     }
279     }
280     }
281     }
282     }
283     }
284     }
285     }
286     }
287     }
288     }
289     }
290     }
291     }
292     }
293     }
294     }
295     }
296     }
297     }
298     }
299     }
300     }
301     }
302     }
303     }
304     }
305     }
306     }
307     }
308     }
309     }
310     }
311     }
312     }
313     }
314     }
315     }
316     }
317     }
318     }
319     }
320     }
321     }
322     }
323     }
324     }
325     }
326     }
327     }
328     }
329     }
330     }
331     }
332     }
333     }
334     }
335     }
336     }
337     }
338     }
339     }
340     }
341     }
342     }
343     }
344     }
345     }
346     }
347     }
348     }
349     }
350     }
351     }
352     }
353     }
354     }
355     }
356     }
357     }
358     }
359     }
360     }
361     }
362     }
363     }
364     }
365     }
366     }
367     }
368     }
369     }
370     }
371     }
372     }
373     }
374     }
375     }
376     }
377     }
378     }
379     }
380     }
381     }
382     }
383     }
384     }
385     }
386     }
387     }
388     }
389     }
390     }
391     }
392     }
393     }
394     }
395     }
396     }
397     }
398     }
399     }
400     }
401     }
402     }
403     }
404     }
405     }
406     }
407     }
408     }
409     }
410     }
411     }
412     }
413     }
414     }
415     }
416     }
417     }
418     }
419     }
420     }
421     }
422     }
423     }
424     }
425     }
426     }
427     }
428     }
429     }
430     }
431     }
432     }
433     }
434     }
435     }
436     }
437     }
438     }
439     }
440     }
441     }
442     }
443     }
444     }
445     }
446     }
447     }
448     }
449     }
450     }
451     }
452     }
453     }
454     }
455     }
456     }
457     }
458     }
459     }
460     }
461     }
462     }
463     }
464     }
465     }
466     }
467     }
468     }
469     }
470     }
471     }
472     }
473     }
474     }
475     }
476     }
477     }
478     }
479     }
480     }
481     }
482     }
483     }
484     }
485     }
486     }
487     }
488     }
489     }
490     }
491     }
492     }
493     }
494     }
495     }
496     }
497     }
498     }
499     }
500     }
501     }
502     }
503     }
504     }
505     }
506     }
507     }
508     }
509     }
510     }
511     }
512     }
513     }
514     }
515     }
516     }
517     }
518     }
519     }
520     }
521     }
522     }
523     }
524     }
525     }
526     }
527     }
528     }
529     }
530     }
531     }
532     }
533     }
534     }
535     }
536     }
537     }
538     }
539     }
540     }
541     }
542     }
543     }
544     }
545     }
546     }
547     }
548     }
549     }
550     }
551     }
552     }
553     }
554     }
555     }
556     }
557     }
558     }
559     }
560     }
561     }
562     }
563     }
564     }
565     }
566     }
567     }
568     }
569     }
570     }
571     }
572     }
573     }
574     }
575     }
576     }
577     }
578     }
579     }
580     }
581     }
582     }
583     }
584     }
585     }
586     }
587     }
588     }
589     }
590     }
591     }
592     }
593     }
594     }
595     }
596     }
597     }
598     }
599     }
600     }
601     }
602     }
603     }
604     }
605     }
606     }
607     }
608     }
609     }
610     }
611     }
612     }
613     }
614     }
615     }
616     }
617     }
618     }
619     }
620     }
621     }
622     }
623     }
624     }
625     }
626     }
627     }
628     }
629     }
630     }
631     }
632     }
633     }
634     }
635     }
636     }
637     }
638     }
639     }
640     }
641     }
642     }
643     }
644     }
645     }
646     }
647     }
648     }
649     }
650     }
651     }
652     }
653     }
654     }
655     }
656     }
657     }
658     }
659     }
660     }
661     }
662     }
663     }
664     }
665     }
666     }
667     }
668     }
669     }
670     }
671     }
672     }
673     }
674     }
675     }
676     }
677     }
678     }
679     }
680     }
681     }
682     }
683     }
684     }
685     }
686     }
687     }
688     }
689     }
690     }
691     }
692     }
693     }
694     }
695     }
696     }
697     }
698     }
699     }
700     }
701     }
702     }
703     }
704     }
705     }
706     }
707     }
708     }
709     }
710     }
711     }
712     }
713     }
714     }
715     }
716     }
717     }
718     }
719     }
720     }
721     }
722     }
723     }
724     }
725     }
726     }
727     }
728     }
729     }
730     }
731     }
732     }
733     }
734     }
735     }
736     }
737     }
738     }
739     }
740     }
741     }
742     }
743     }
744     }
745     }
746     }
747     }
748     }
749     }
750     }
751     }
752     }
753     }
754     }
755     }
756     }
757     }
758     }
759     }
760     }
761     }
762     }
763     }
764     }
765     }
766     }
767     }
768     }
769     }
770     }
771     }
772     }
773     }
774     }
775     }
776     }
777     }
778     }
779     }
780     }
781     }
782     }
783     }
784     }
785     }
786     }
787     }
788     }
789     }
790     }
791     }
792     }
793     }
794     }
795     }
796     }
797     }
798     }
799     }
800     }
801     }
802     }
803     }
804     }
805     }
806     }
807     }
808     }
809     }
810     }
811     }
812     }
813     }
814     }
815     }
816     }
817     }
818     }
819     }
820     }
821     }
822     }
823     }
824     }
825     }
826     }
827     }
828     }
829     }
830     }
831     }
832     }
833     }
834     }
835     }
836     }
837     }
838     }
839     }
840     }
841     }
842     }
843     }
844     }
845     }
846     }
847     }
848     }
849     }
850     }
851     }
852     }
853     }
854     }
855     }
856     }
857     }
858     }
859     }
860     }
861     }
862     }
863     }
864     }
865     }
866     }
867     }
868     }
869     }
870     }
871     }
872     }
873     }
874     }
875     }
876     }
877     }
878     }
879     }
880     }
881     }
882     }
883     }
884     }
885     }
886     }
887     }
888     }
889     }
890     }
891     }
892     }
893     }
894     }
895     }
896     }
897     }
898     }
899     }
900     }
901     }
902     }
903     }
904     }
905     }
906     }
907     }
908     }
909     }
910     }
911     }
912     }
913     }
914     }
915     }
916     }
917     }
918     }
919     }
920     }
921     }
922     }
923     }
924     }
925     }
926     }
927     }
928     }
929     }
930     }
931     }
932     }
933     }
934     }
935     }
936     }
937     }
938     }
939     }
940     }
941     }
942     }
943     }
944     }
945     }
946     }
947     }
948     }
949     }
950     }
951     }
952     }
953     }
954     }
955     }
956     }
957     }
958     }
959     }
960     }
961     }
962     }
963     }
964     }
965     }
966     }
967     }
968     }
969     }
970     }
971     }
972     }
973     }
974     }
975     }
976     }
977     }
978     }
979     }
980     }
981     }
982     }
983     }
984     }
985     }
986     }
987     }
988     }
989     }
990     }
991     }
992     }
993     }
994     }
995     }
996     }
997     }
998     }
999     }
1000    }

```

插入数据的线程代码:

```

127 }
128 }
129 class Thread4 extends Thread{
130     private String name;
131     private Connection conn;
132     public Thread4(String name,Connection conn) {
133         this.name=name;
134         this.conn=conn;
135     }
136     public void run() {
137         long starttime = System.currentTimeMillis();
138         int line = -1;
139         boolean flag = true;
140         while(flag){
141             try{
142                 Statement stat = conn.createStatement();
143                 String sql = "delete from movies where year in (2016,2017) and id in (select movie_id from ratings where rating > 6);";
144                 line = stat.executeUpdate(sql);
145                 conn.commit();
146             }catch( Exception e ){
147                 e.printStackTrace();
148                 try{
149                     conn.rollback();
150                 }catch(Exception ex){
151                     ex.printStackTrace();
152                 }
153             }finally{
154                 if(line != -1) flag = false;
155             }
156             System.out.println("Thread4(del) over with runtime = " + (System.currentTimeMillis()-starttime));
157         }
158     }
159 }
160
161 public class mySqlTest {
162     public static void main(String[] args) throws ClassNotFoundException {
163         try {
164             Class.forName("com.mysql.jdbc.Driver");

```

(4) 查询结果:

Test in sqlite...

Problem1:

Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

average runtime = 90.1671807526218
Thread1(avg) over with runtime = 2324
Thread2(ins) over with runtime = 33733

Problem2:

Return the average runtime of movies whose rating is over 6.0

And delete movies from the table whose rating is over 6.0 and release year is 2016 and 2017.

```
java.sql.SQLException: database is locked
    at org.sqlite.DB.throwex(DB.java:252)
    at org.sqlite.NestedDB.prepare(NestedDB.java:84)
    at org.sqlite.DB.prepare(DB.java:62)
    at org.sqlite.Stmt.executeQuery(Stmt.java:69)
    at sqliteTest.Thread3.run(sqliteTest.java:102)
java.sql.SQLException: database is locked
    at org.sqlite.DB.throwex(DB.java:252)
    at org.sqlite.NestedDB.prepare(NestedDB.java:84)
    at org.sqlite.DB.prepare(DB.java:62)
    at org.sqlite.Stmt.executeQuery(Stmt.java:69)
    at sqliteTest.Thread3.run(sqliteTest.java:102)
java.sql.SQLException: database is locked
    at org.sqlite.DB.throwex(DB.java:252)
    at org.sqlite.NestedDB.prepare(NestedDB.java:84)
    at org.sqlite.DB.prepare(DB.java:62)
    at org.sqlite.Stmt.executeQuery(Stmt.java:69)
    at sqliteTest.Thread3.run(sqliteTest.java:102)
java.sql.SQLException: database is locked
    at org.sqlite.DB.throwex(DB.java:252)
    at org.sqlite.NestedDB.prepare(NestedDB.java:84)
    at org.sqlite.DB.prepare(DB.java:62)
    at org.sqlite.Stmt.executeQuery(Stmt.java:69)
    at sqliteTest.Thread3.run(sqliteTest.java:102)
java.sql.SQLException: database is locked
    at org.sqlite.DB.throwex(DB.java:252)
    at org.sqlite.NestedDB.prepare(NestedDB.java:84)
    at org.sqlite.DB.prepare(DB.java:62)
    at org.sqlite.Stmt.executeQuery(Stmt.java:69)
    at sqliteTest.Thread3.run(sqliteTest.java:102)
Thread4(del) over with runtime = 17036
average runtime = 94.6837400640132
Thread3(avg) over with runtime = 32472
```

The test program for sqlite is over.....

结果分析（性能、正确性）:

在问题 1 中，由于查询时间很快，所以在执行插入操作之前，就已经提交了这个事务，导致再插入时数据库的状态不是被锁的，因为没有抛出因为对加锁时数据库操作而造成的异常。

在问题 2 中，由于查询和删除的操作的时间都较长，所以在一个进程争取到对数据库的控制之后，另一个线程就无法继续操作数据库了，从结果的截图上来看，就是抛出了异常。针对这一情况，我将两个线程都写为不断地试探访问，直到最后访问成功了才退出 **while** 循环。所以等待的线程不断的尝试访问，直到获得了数据库的锁，才开始执行，并不再抛出异常。

可以看到 **sqlite** 中，并发粒度是很粗燥的，它无法精确到元组（行），所以导致执行力的相对低下。

2. MySql

- (1) 隔离级别: SERIALIZABLE
- (2) 封锁粒度: table
- (3) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: lock table
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0191
Thread Pre (avg) over with runtime = 13341

Thread A starts...
Thread B starts...
average runtime = 100.0191
Thread A (avg) over with runtime = 12648
Thread B (ins) over with runtime = 24018

The current average number:
Thread Con starts...
average runtime = 100.0200
Thread Con (avg) over with runtime = 15767

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: table lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 87
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 947

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 942
Thread D (del) over with runtime = 1417

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 912

The test program for mysql is over.....
```


- (4) 隔离级别: SERIALIZABLE
- (5) 封锁粒度: IS
- (6) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: share lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0332
Thread Pre (avg) over with runtime = 18705

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 19273
average runtime = 100.0345
Thread A (avg) over with runtime = 30344

The current average number:
Thread Con starts...
average runtime = 100.0345
Thread Con (avg) over with runtime = 16346

The test program for mysql is over.....

Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: share lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 85
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 943

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 946
Thread D (del) over with runtime = 1361

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 925

The test program for mysql is over.....
```

(7) 隔离级别: SERIALIZABLE

(8) 封锁粒度: IX

(9) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: x lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 84
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 1018

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 931
Thread D (del) over with runtime = 1348

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 912

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: x lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0345
Thread Pre (avg) over with runtime = 25949

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 20079
average runtime = 100.0338
Thread A (avg) over with runtime = 33199

The current average number:
Thread Con starts...
average runtime = 100.0338
Thread Con (avg) over with runtime = 17052

The test program for mysql is over.....
```

(10) 隔离级别: REPEATABLE READ

(11) 封锁粒度: table

(12) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: lock table
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0137
Thread Pre (avg) over with runtime = 12873

Thread A starts...
Thread B starts...
average runtime = 100.0137
Thread A (avg) over with runtime = 12956
Thread B (ins) over with runtime = 25690

The current average number:
Thread Con starts...
average runtime = 100.0191
Thread Con (avg) over with runtime = 13871

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: table lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and release year is 1963.

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 84
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 942

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 948
Thread D (del) over with runtime = 1364

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 921

The test program for mysql is over.....
```


(13) 隔离级别: REPEATABLE READ

(14) 封锁粒度: IS

(15) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: share lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0311
Thread Pre (avg) over with runtime = 15931

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 16214
average runtime = 100.0332
Thread A (avg) over with runtime = 26234

The current average number:
Thread Con starts...
average runtime = 100.0332
Thread Con (avg) over with runtime = 15205

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: share lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 91
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 944

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 935
Thread D (del) over with runtime = 1351

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 910

The test program for mysql is over.....
```


(16) 隔离级别: REPEATABLE READ

(17) 封锁粒度: IX

(18) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: x lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0338
Thread Pre (avg) over with runtime = 17554

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 15450
average runtime = 100.0349
Thread A (avg) over with runtime = 27509

The current average number:
Thread Con starts...
average runtime = 100.0349
Thread Con (avg) over with runtime = 16046

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: x lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 89
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 943

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 934
Thread D (del) over with runtime = 1349

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 912

The test program for mysql is over.....
```

(19) 隔离级别: READ UNCOMMITTED

(20) 封锁粒度: table

(21) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_READ_UNCOMMITTED
Granularity: lock table
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0200
Thread Pre (avg) over with runtime = 13214

Thread A starts...
Thread B starts...
average runtime = 100.0200
Thread A (avg) over with runtime = 12510
Thread B (ins) over with runtime = 23082

The current average number:
Thread Con starts...
average runtime = 100.0165
Thread Con (avg) over with runtime = 15423

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_READ_UNCOMMITTED
Granularity: table lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 87
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 949

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 939
Thread D (del) over with runtime = 1361

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 913

The test program for mysql is over.....
```

(22) 隔离级别: READ UNCOMMITTED

(23) 封锁粒度: IS

(24) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_READ_UNCOMMITTED
Granularity: share lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0245
Thread Pre (avg) over with runtime = 15370

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 15222
average runtime = 100.0311
Thread A (avg) over with runtime = 25905

The current average number:
Thread Con starts...
average runtime = 100.0311
Thread Con (avg) over with runtime = 15215

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: share lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 88
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 940

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 946
Thread D (del) over with runtime = 1380

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 917

The test program for mysql is over.....
```


(25) 隔离级别: READ UNCOMMITTED

(26) 封锁粒度: IX

(27) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_READ_UNCOMMITTED
Granularity: x lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0292
Thread Pre (avg) over with runtime = 18102

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 17818
average runtime = 100.0290
Thread A (avg) over with runtime = 31583

The current average number:
Thread Con starts...
average runtime = 100.0290
Thread Con (avg) over with runtime = 18669

The test program for mysql is over.....

Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_READ_UNCOMMITTED
Granularity: x lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 85
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 939

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 953
Thread D (del) over with runtime = 1363

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 916

The test program for mysql is over.....
```

(28) 隔离级别: READ COMMITTED

(29) 封锁粒度: table

(30) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: lock table
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0165
Thread Pre (avg) over with runtime = 13757

Thread A starts...
Thread B starts...
average runtime = 100.0165
Thread A (avg) over with runtime = 13611
Thread B (ins) over with runtime = 25819

The current average number:
Thread Con starts...
average runtime = 100.0209
Thread Con (avg) over with runtime = 14945

The test program for mysql is over.....

Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日)
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: table lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 88
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 943

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 942
Thread D (del) over with runtime = 1359

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 913

The test program for mysql is over.....
```

(31) 隔离级别: READ COMMITTED

(32) 封锁粒度: IS

(33) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: share lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0209
Thread Pre (avg) over with runtime = 14144

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 15833
average runtime = 100.0245
Thread A (avg) over with runtime = 26129

The current average number:
Thread Con starts...
average runtime = 100.0245
Thread Con (avg) over with runtime = 14098

The test program for mysql is over.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: share lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 88
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 940

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 946
Thread D (del) over with runtime = 1380

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 917

The test program for mysql is over.....
```


(34) 隔离级别: READ COMMITTED

(35) 封锁粒度: IX

(36) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem1:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: x lock
Return the average runtime of movies whose release year is 1963.
And insert movies infos into the movies table whose year is 1963.

The original average number:
Thread Pre starts...
average runtime = 100.0256
Thread Pre (avg) over with runtime = 17118

Thread A starts...
Thread B starts...
Thread B (ins) over with runtime = 16262
average runtime = 100.0292
Thread A (avg) over with runtime = 29753

The current average number:
Thread Con starts...
average runtime = 100.0292
Thread Con (avg) over with runtime = 17640

The test program for mysql is over.....

Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月23日
Test in mysql...

*****
Problem2:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: x lock
Return the average runtime of movies whose rating is over 6.0
And delete movies from the table whose rating is over 6.0 and releas

Thread Rcv starts to recover...
Thread Rcv (del) over with runtime = 85
The original average number:
Thread Pre starts to calculate for the average...
average runtime = 94.7575
Thread Pre (avg) over with runtime = 943

Thread C starts to calculate for the average...
Thread D starts to delete
average runtime = 94.7575
Thread C (avg) over with runtime = 941
Thread D (del) over with runtime = 1358

The current average number:
Thread Con starts to calculate for the average...
average runtime = 94.6837
Thread Con (avg) over with runtime = 907

The test program for mysql is over.....
```

结果分析（性能、正确性）：

* 对于问题二，导入的数据库不同于问题一所使用的数据库，原因是问题 2 中需要的嵌套查询/删除在问题 1 中的数据库中无法进行，所以两个问题使用的数据库不同，因此以下只做横向（在同一数据库中）的分析。

* 关于行锁的使用：文档中提到对于 `update` 的操作，只有索引存在时才会执行行锁，否则严格为表锁。通过对于 `movies` 的 `schema` 的检查发现，`movies` 中的 `id` 属性是主码。又表会自动建立主码的索引，所以我们无需另外建立索引以执行 `update` 的行锁。

正确性：

在问题 1 中，我分别在场景开始前后计算了平均值（通过单独运行计算均值的线程以及 `join` 函数对于线程顺序的控制），可以从上述的截图中看到，在场景中计算得到的均值要么和场景开始前一致，或者与场景结束后一致，说明在场景中均值的计算要么是在插入之前，要么是在插入之后，证明了运行的正确性。

在问题 2 中，我所使用的数据库中已经对于要删除的内容建立了新表 `recovery`，所以每次开始场景前，都先要将 `recovery` 中的内容插入到 `movies` 表中。也是基于这一点，问题二中答案比较固定（因为问题 1 中有随机插入，答案因此不确定）。我延续了问题 1 中的方法，同样可以看到均值的结果与场景执行前/后一致，因此可以证明正确性。

性能分析：

在问题 1 中，可以看到排他锁的时间性能与表锁的性能相当，原因在于在我的程序执行 `x` 锁时，都是插入线程先抢到控制权，那么由于我插入的手段是批处理，所以只有等所有数据项都插入之后，才继续进行，所以这些行都在 `commit` 之前被锁住，所以之后进行的 `average` 的查询只能等待这些行被释放之后，才能继续访问这些行，所以说 `x` 锁和表锁的运行时间相当。而在执行 `s` 锁时，求平均数的线程似乎比另外两种情况来的快，可能是读取时使用读锁时自带的优化？这一点尚不确定。另外值得注意的是，当执行表锁的时候，先唤起的均值线程先执行，而另外两种情况则后执行，这是因为表锁时均值线程先获得了整个表的锁因此插入线程只能等待均值线程锁的释放，而后二者则因为插入线程的控制元组是均值线程访问的子集，因此插入线程先获得了所有插入行的行锁，而均值线程即使已经访问了其他原有的线程，还是只能等待插入线程先释放锁，因此会出现均值线程先被唤醒却后结束的现象。

在问题 2 中，输出结果的差别在 `20ms` 之内，几乎看不出性能上的差别。

我认为作业的两个问题在所有粒度以及隔离程度的组合下都差别不大的原因在于场景中的读操作都需要更新操作结束后才能得到正确的结果，或者在更新操作结束前就读完所有的数据。这一点导致了本质上正确性依赖于串行化的执行顺序，因此各个粒度的隔离程度的区别不大。

3. Bonus

- (1) 场景描述: 多用户读取同一个表中的数据
- (2) 隔离级别: SERIALIZABLE VS. READ_COMMITTED
- (3) 封锁粒度: table lock VS. share lock
- (4) 查询结果:

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月24日)
Test in mysql...

*****
Problem3:
Isolation level: TRANSACTION_READ_COMMITTED
Granularity: share lock
Return the sum runtime of movies of every period

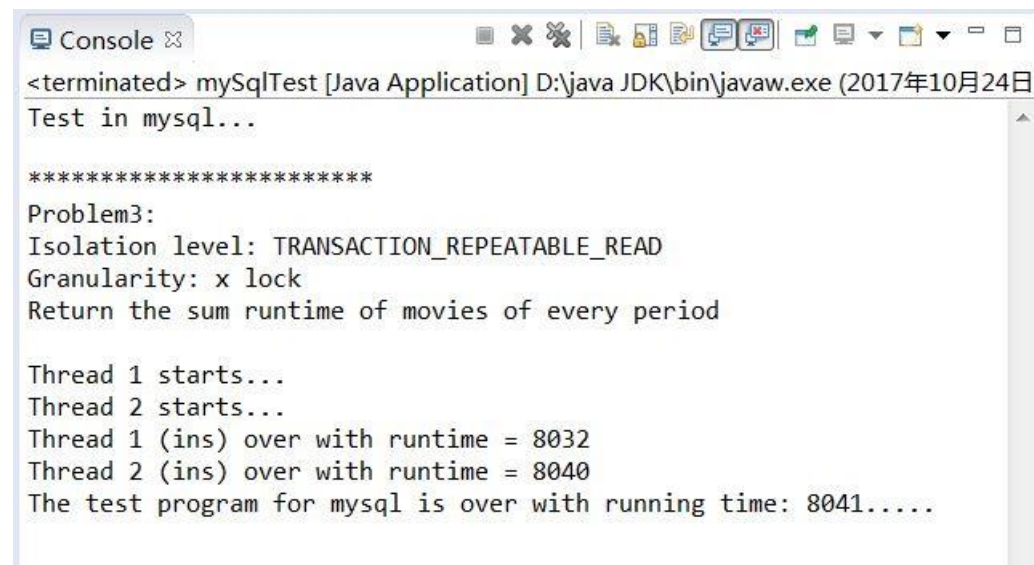
Thread 1 starts...
Thread 1 has key...
Thread 2 starts...
Thread 2 has key...
sum runtime (1900<= year <2100) = 74787032
Thread 2 (avg) over with runtime = 1295
sum runtime (1900<= year <2100) = 74787032
Thread 1 (avg) over with runtime = 1297
The test program for mysql is over with running time: 1298.....
```

```
Console
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月24日)
Test in mysql...

*****
Problem3:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: table lock
Return the sum runtime of movies of every period

Thread 2 starts...
Thread 1 starts...
Thread 2 has key...
sum runtime (1900<= year <2100) = 74787032
Thread 1 has key...
Thread 2 (avg) over with runtime = 1308
sum runtime (1900<= year <2100) = 74787032
Thread 1 (avg) over with runtime = 2609
The test program for mysql is over with running time: 2610.....
```


- (1) 场景描述:
- (2) 隔离级别:
- (3) 封锁粒度:
- (4) 查询结果:

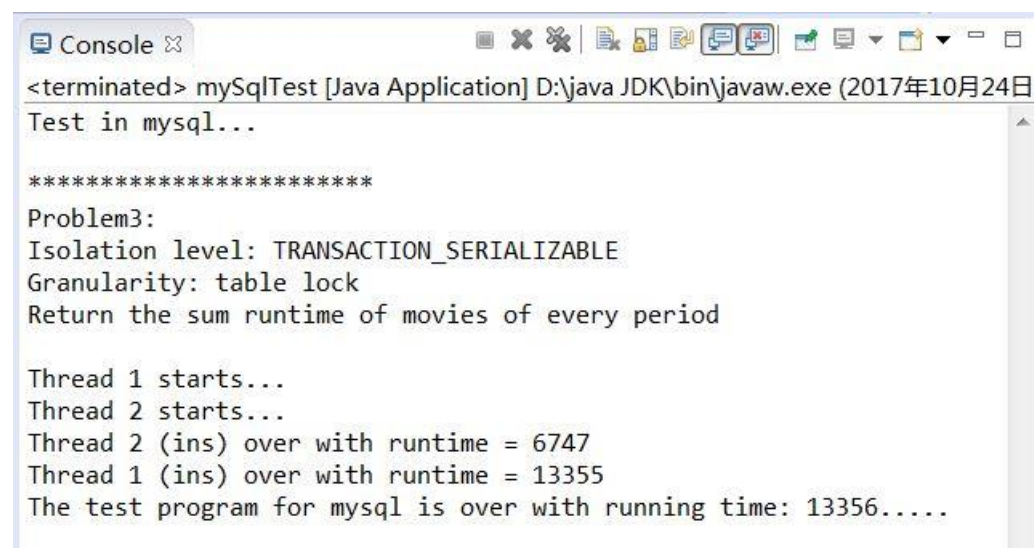


```
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月24日
Test in mysql...

*****

Problem3:
Isolation level: TRANSACTION_REPEATABLE_READ
Granularity: x lock
Return the sum runtime of movies of every period

Thread 1 starts...
Thread 2 starts...
Thread 1 (ins) over with runtime = 8032
Thread 2 (ins) over with runtime = 8040
The test program for mysql is over with running time: 8041.....
```



```
<terminated> mySqlTest [Java Application] D:\java JDK\bin\javaw.exe (2017年10月24日
Test in mysql...

*****

Problem3:
Isolation level: TRANSACTION_SERIALIZABLE
Granularity: table lock
Return the sum runtime of movies of every period

Thread 1 starts...
Thread 2 starts...
Thread 2 (ins) over with runtime = 6747
Thread 1 (ins) over with runtime = 13355
The test program for mysql is over with running time: 13356.....
```

结果分析（性能、正确性）:

可以看到，对于有多个线程同时读或者同时写的时候，设置表锁的效率要明显低于设置行锁的效率，这是因为表锁的设置让线程的执行串行化了（后执行的线程的执行时间是先执行的线程的两倍，因为先要等前一个线程执行完才能继续，而自己的执行时间也与另一个线程相同（因为两个线程的任务是一致的）），而行锁的设置则类似于 pipeline 的形式，即两个线程的执行几乎是同时的（两个线程的结束时间几乎是一致的，就好像同时运行一样）。