基于MySQL等关系数据库的12306售票设计

总述

12306是铁路总公司的官方售票窗口，而中国每年的春运即为世界上最大的人口迁移，因此，个人认为这个售票系统超越了双11的电商，应该是世界上并发最大的系统。

设计时充分考虑并发等详情，提炼出以下要点

- 空间换时间
- 数据库、代码一同优化
- 人力、器力之能力有大限，有些问题是没法解决的

关于第三点，个人理解是这样的，不是一句丧气的话，而是现实情况。春运时每一个座位都可以抽象为有1000多人去抢，再多的技术支持、锁结构，面对1000个访问请求也不可能瞬间完成，因此要尽最大努力！

思路

将按照这样的思路来进行描述，首先是数据库层次的设计，之后是数据库层级提供的扩展（因为时间、能力关系没有实现但是感觉会更好的举措）;然后是代码级的优化，最后是压力测试和结果

数据库设计

表结构设计

一共设计了8+车辆数张表，将分别介绍这9张表。

account

```
CREATE TABLE IF NOT EXISTS account (
  aid BIGINT NOT NULL AUTO_INCREMENT,
  email VARCHAR(30) NOT NULL ,
  password VARCHAR(20) NOT NULL ,
  PRIMARY KEY (aid,email)
);
```

主要是负责存储账户，这个表的安全性大于它的并发性，因此作业中设计的比较简单，这样也方便。

customer

```
CREATE TABLE IF NOT EXISTS customer (
  aid BIGINT NOT NULL ,
  cid BIGINT NOT NULL ,
  cname VARCHAR(255) NOT NULL ,
  ctype TINYINT DEFAULT 0 NOT NULL ,
```

```
  phone VARCHAR(20) DEFAULT NULL ,
  identity VARCHAR(20) NOT NULL ,
  PRIMARY KEY (aid,cid),
  FOREIGN KEY aidf(aid) REFERENCES account(aid) ON DELETE CASCADE ON UPDATE CASCADE
);
```

customer对应12306中的常用乘车人，每一个账号下面可能会有多个，因此采用复合主键，其重要程度也一般。这里的ctype是保留扩展，现在有成人票和学生票之分，将来或许还会有老人票等等

carriage_stype_seats

```
CREATE TABLE IF NOT EXISTS carriage_stype_seats (
  stype TINYINT NOT NULL DEFAULT 2,
  row INTEGER NOT NULL ,
  location TINYINT DEFAULT 0 NOT NULL ,
  price DOUBLE DEFAULT 0,
  student_price DOUBLE DEFAULT 0,
  PRIMARY KEY (stype,row,location)
);
```

这个表是功能表，代码中还未实现，存储了车厢和其对应的座位图，比如当前情况下stype只有商务座、一等座、二等座，然后存储的是其内部的座位排号、编号以及价格。**这是为了后面建表的可扩展性设计的。比如现在动车有了卧铺，那么这个表的stype就应该多一种，然后存储其内部的座位编号。** row代表第几排，location代表座位号，现在是0~4，即ABCDE。

carriage

```
CREATE TABLE IF NOT EXISTS carriage (
  carriage_id BIGINT NOT NULL AUTO_INCREMENT,
  stype TINYINT DEFAULT 2 NOT NULL ,
  producedAt DATETIME  DEFAULT '1000-01-01 00:00:00',
  fixedAt DATETIME  DEFAULT '1000-01-01 00:00:00',
  dumpedAt DATETIME   DEFAULT '1000-01-01 00:00:00',
  PRIMARY KEY (carriage_id),
  FOREIGN KEY stypef(stype) REFERENCES carriage_stype_seats(stype) ON DELETE
CASCADE ON UPDATE CASCADE
);
```

这个表用来存储车厢信息，比如其唯一的id，然后stype代表类型，之后的是车厢自己的信息，比如生产于producedAt的日期，最近一次维修是fiedAt，并且将于dumpedAt时报废。对应作业要求中的车厢管理。比如50年后，铁总肯定会从南车、北车订购，那么信息应该加入这里

train_carriage

```
CREATE TABLE IF NOT EXISTS train_carriage (
  tid VARCHAR(20) NOT NULL ,
  carriage_id BIGINT ,
  stype TINYINT NOT NULL DEFAULT 2,
  t_c_id TINYINT NOT NULL DEFAULT 0,
  t_time DATETIME   DEFAULT '1000-01-01 00:00:00',
  PRIMARY KEY (tid,carriage_id)
);
```

这个表用来存储车厢编组信息，虽然CMS上有同学指出动车组车厢不会改变，但是个人感觉不妥，即使是同一批次的车厢也会寿命不同，因此肯定会有早坏的，这时候必然是更换一节车厢而不是整个动车。tid代表车号，比如G5，然后carriage_id代表车厢的id，特别的是t_c_id代表车厢在动车组的车号，也就是平常所说的第几车。最后一个是车厢编组的开始时间，这个是凭空的个人猜测，不一定稳妥，只是感觉会有作用。

station_train

```
CREATE TABLE IF NOT EXISTS station_train (
  station VARCHAR(255) NOT NULL ,
  tid VARCHAR(20) NOT NULL ,
  orderedNum INT DEFAULT 0 NOT NULL ,
  arrive_at TIME NOT NULL ,
  leave_at TIME NOT NULL ,
  after_day TINYINT DEFAULT 0 NOT NULL ,
  length DOUBLE DEFAULT 0 NOT NULL,
  PRIMARY KEY (station,tid)
);
```

两个重要表之一！存储了车站和列车的关系，比如北京到上海都有哪些车次，是依靠这个实现的。orderedNum代表该站在列车时刻表的序号。比如列车是北京-南京-上海，对应为1-2-3。另一个需要说明的是arrive_at,leave_at采用time的格式，是简化了模型，默认每次车每天都会发车（其实一般的确是这样），而after_day代表过了几天（距离发车），每当time过12点就会加一。最后的里程模仿高速公路，记录的是距离出发点的总里程，用来与车厢的price一起计算票价。
额外说明是，目前没有实现时间显示功能，因为时间原因，因为mongo第一次用学习成本太高。但是想说明的是实现这个功能很简单，多选数据就ok

order

```
CREATE TABLE IF NOT EXISTS `order` (
  `oid` VARCHAR(255) NOT NULL ,
  aid BIGINT NOT NULL ,
  orderAt DATETIME  DEFAULT '1000-01-01 00:00:00',
  tid VARCHAR(20) NOT NULL ,
  `start` VARCHAR(255) NOT NULL ,
  `end` VARCHAR(255) NOT NULL ,
```

```
  startAt DATETIME NOT NULL  DEFAULT '1000-01-01 00:00:00',
  PRIMARY KEY (oid),
  FOREIGN KEY aidf(aid) REFERENCES account(aid) ON UPDATE CASCADE ON DELETE CASCADE
,
  FOREIGN KEY tidf(tid) REFERENCES train_carriage(tid) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

这张表用于存放订单，记录了订单号、账户号、下单时间、车次、出发站、到达站和发车时间。简言之就是订单基本信息。没有什么特殊的地方。

order_ticket

```
CREATE TABLE IF NOT EXISTS order_ticket (
  `oid` VARCHAR(255) NOT NULL ,
  cid BIGINT NOT NULL ,
  aid BIGINT NOT NULL ,
  stype TINYINT DEFAULT 2 NOT NULL ,
  ctype TINYINT DEFAULT 0 NOT NULL ,
  t_c_id TINYINT NOT NULL ,
  row INT NOT NULL ,
  location TINYINT NOT NULL ,
  PRIMARY KEY (oid,cid),
  FOREIGN KEY (aid) REFERENCES account(aid) ON DELETE CASCADE ON UPDATE CASCADE
);
```

这个表存储订单与顾客的映射，由于每个订单可能会有多个乘车人。存储的东西之前都出现过，这里不再说明。

G?

```
 CREATE TABLE IF NOT EXISTS g_time (
   tid VARCHAR(20) NOT NULL ,
   t_c_id TINYINT NOT NULL ,
   stype TINYINT NOT NULL DEFAULT 2,
   row INTEGER NOT NULL ,
   location TINYINT DEFAULT 0 NOT NULL ,
   ticket BIT(?) DEFAULT b'?',
   PRIMARY KEY (tid,t_c_id,row,location)
 );
```

空间换时间，因此每个车是一个表。这里有一个疑问，这样的设计和聚合大表之后采用partion究竟哪个更有效？希望助教解答一下啦 这里比较精心的设计是采用了bitvalue，存进数据库中的是一堆二进制的1，位数由线路上的车站数决定。然后1代表有座位，并且是以右边作为第一站点。 例如南京-济南-北京-天津，111代表全有票，010代表只有济南到北京的票。这样做的好处在于，结合之前车站在线路上的编号，可以很快的生成查询的掩

码，比如查询济南到天津，掩码为110，就是

- 可以由之前查到济南编号2，天津为4，相差为2，代表有两位1->生成2^i次方-1，4-1=3=11
- 之后由于出发站济南为2，代表右移1位，就有了110

同时，mysql支持按位的与、或、异或操作，并且个人感觉这些操作速度很快，因此选择了这样的设计。

下一步展望

分表

对于账户、订单这些相关的表，应该采取一定的分表策略。订单可以采用日期（不加时间），而账户可以考虑个人身份证前两位，代表省份。

行级别锁

这次作业，对于买票，由于每个座位单独独立，因此可以考虑采取行锁，因为每次update操作只改变一行，这样会更有利于并发执行。

分布式？

这个是疑问，MySQL应该会有支持分布式的，这样可以减少机器的压力。

代码级的优化

为什么

个人认为对于12306这样的系统，想要最大程度的支持并发，绝不仅仅是数据库设计这一部分，完美的数据库也会被逻辑层、展示层调用，这些代码很大程度上决定了系统的质量。因此来描述一下实现上的想法。

synchronized代码块

在搜寻资料的时候，有人放出来过12年的时候12306的报错图，那张图上他们用的是hibernate框架。无论现在是用的什么语言，Java中的多线程要使用synchronized来保证线程间的互斥。也就是多人买票的问题。这里我采用了guava包中的方法并设置synchronized的锁是String类型的tid，即列车编号。这样确保了在代码中对于同一辆车的操作是互斥的。虽然还有不足的地方，比如最好是仅仅互斥到排或者座位号，但是对于这次大作业的测验已经足够

transaction事务

借助于synchronized代码块，我们可以手动实现事务的绑定。虽然我个人感觉SQL级别的事务应该更好。不过令人惊奇的是，oracle官方给出的jdbc实现即为这样，采用try-catch模块，在有异常时rollback。

与SQL语言交互

这里个人的中心思想是逻辑判断放在Java，值的判断放在数据库。比如连座问题用代码解决。

连座策略

比如二等座、每排5人，有一组顾客买票假设3人。程序会先搜寻一下有没有符合条件的排，即空座大于等于3的，如果没有，再会随机查找。

测试

在这里吹了这么久，需要测试来说话。但是有一点需要注意：

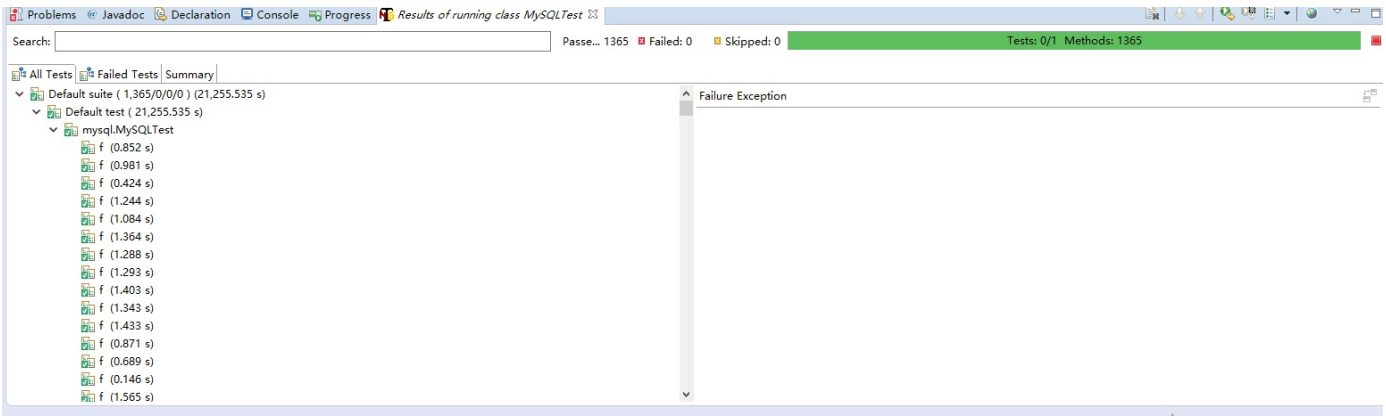
测试是用的网络数据库，因为我心疼自己的小电脑，因此有时延迟很高

分别测试查票和买票操作，为了测试高并发，测试这样设计。

- 查票操作查找的是G41列车所经过的所有站点来回随机查
- 买票操作是所有人、买同一天的G41车次，随机的地点、随机的人数（1~15），随机的座位类型
- 测试采用testng框架，支持多线程测试

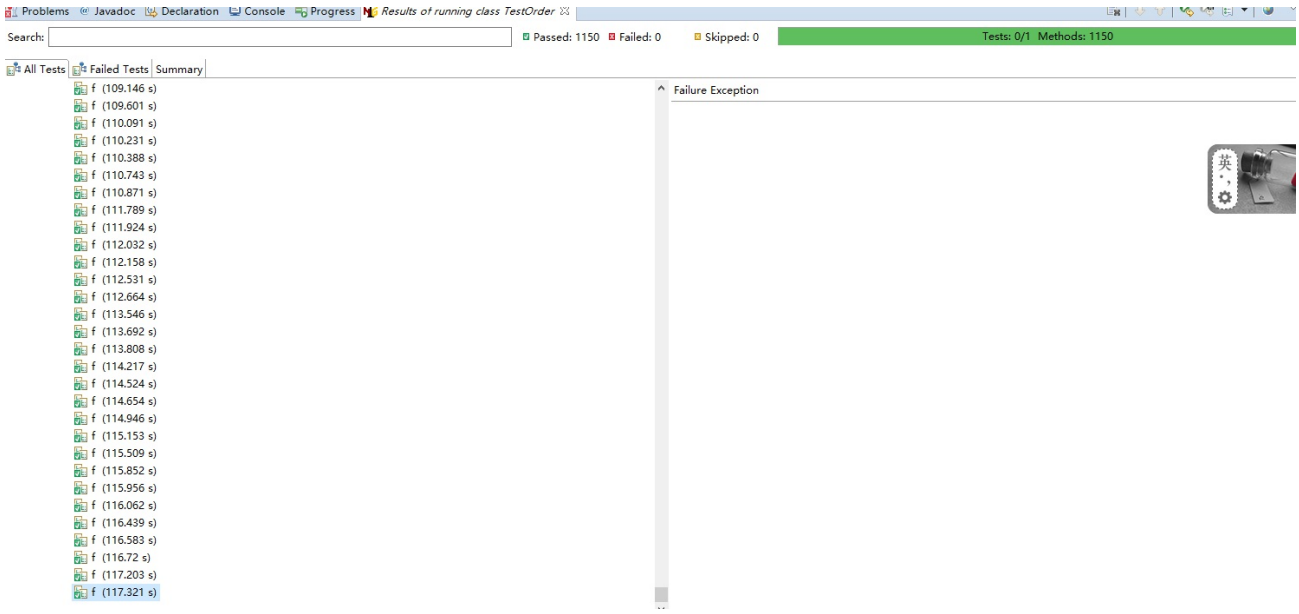
查票测试

```
@Test(invocationCount = 1500, threadPoolSize = 500)
public void f() {
    int start=0;
      while(start==0||start==destination.length){
          Double a=new Double(Math.random()*destination.length);
          start=a.intValue();
      }
      int end=0;
      while(end<=start||end==destination.length){
          end=new Double(Math.random()*destination.length).intValue();
      }
      System.out.println(ticketService.queryTrain(destination[start], destination[end], calendar));
}
```



买票测试

```
  @Test(invocationCount = 1300, threadPoolSize = 500)
public void f() {
    int start=0;
      while(start==0||start==destination.length){
          Double a=new Double(Math.random()*destination.length);
          start=a.intValue();
      }
      int end=0;
      while(end<=start||end==destination.length){
          end=new Double(Math.random()*destination.length).intValue();
      }
      int people = new Double(Math.random()*16).intValue();
      if (people==0) {
          people=1;
      }
      int[] a=new int[people];
      for(int i=0;i<people;i++){
          a[i]=i+1;
      }
      int stype=new Double(Math.random()*4).intValue();
      System.out.println(ticketService.orderTicket(destination[start], destination[en
      System.out.println(i++);
}
```



尾声

通过这次作业，对于MySQL和程序有了更深的理解，同时深感12306的不易。但是就我个人而言，首先，我没有骂过他的逻辑，只吐槽过验证码。其次，我想说，给我3个亿，我也会很有动力。。。