航空出行管理系统数据库优化报告

小组成员：郭梦琪 张晗翀 刘勉之 刘书畅

一、业务背景描述

随着我国经济的高速发展，国民生活水平的不断提高，人们的出行方式的要求越来越高，利用飞机出行已经成为了不少人的首选，飞机出行有方便、快捷、 舒适的特点。对于航空公司来说，业务量逐步增加，需要管理上百万的用户，上千万订单记录，上百的航班等。过多的数据会减缓增删改查的速度，为了保持良好的用户体验，提升管理效率，需要对航空出行管理系统进行优化，将查询时间控制在可控范围内，并需要考虑到多用户并发查询的情况。

二、优化目标

- 降低单人查询时间，包括但不限于查询订单、航班、飞行记录、登机牌信息等
- 提高并发查询速度
- 使用查询缓存，再次查询可提高速度 （加分项）

三、性能测试与分析

1. 查询优化

本项目的查询功能较多，我们均进行了优化，重点分析较为典型的查询订单函数queryOrder。

· 查询订单queryOrder

① 函数优化

查询订单函数需要返回航班信息，登机牌信息等多项信息，而这些信息分别储存在boardingpass、flight、route、order多个表里，需要进行join操作。并且用户通常是通过自己的用户id而非order表里，并非主键查询。

优化前的函数：

```
select `order`.orderId,time,boardId,flightId,number,seat,type,price
from `order` natural join boardingpass
where `order`.CustomerId=CustomerId
```

针对该查询的特点我们做了以下优化：

1. 增加索引：

   boardingpass：flightId CustomerId
   order：CustomerId
   route：departureTime departureAirport arrivalAirport

2. 将fight表的status改为enum类型 ， 因为固定内容范围的查询会比varchar快

3. 若语句使用join加上`order`.orderId=boardingpass.orderId和`order`.CustomerId=CustomerId条件，是order这个数据量较小的表作为驱动表，boardingpass这个数据量很大的表作为被驱动表，检索的行数较少时间较少；若语句使用join加上`order`.orderId=boardingpass.orderId和 boarding.CustomerId=CustomerId条件或者使用natural join ，是在数据量大的表里进行检索，检索的行数很多，时间较慢
4. 替换select* 按需取数据

优化后的函数：

```
 (select `order`.orderId,time,boardId,flightId,number,seat,type,price
from `order`  join boardingpass
where `order`.orderId=boardingpass.orderId and `order`.CustomerId=1)
```

② 利用mysqlslap进行性能测试

*注：此测试设置了两组数据：大数据组有500000条boardingpass数据，小数据组有5000条boardingpass数据。部分测试极高并发下情况使用小数据组进行测试。

此数据针对测试函数设计，用户1的登机牌数和订单数非常多，采用用户1进行测试，所以用户1的查询时间比正常情况下要高。

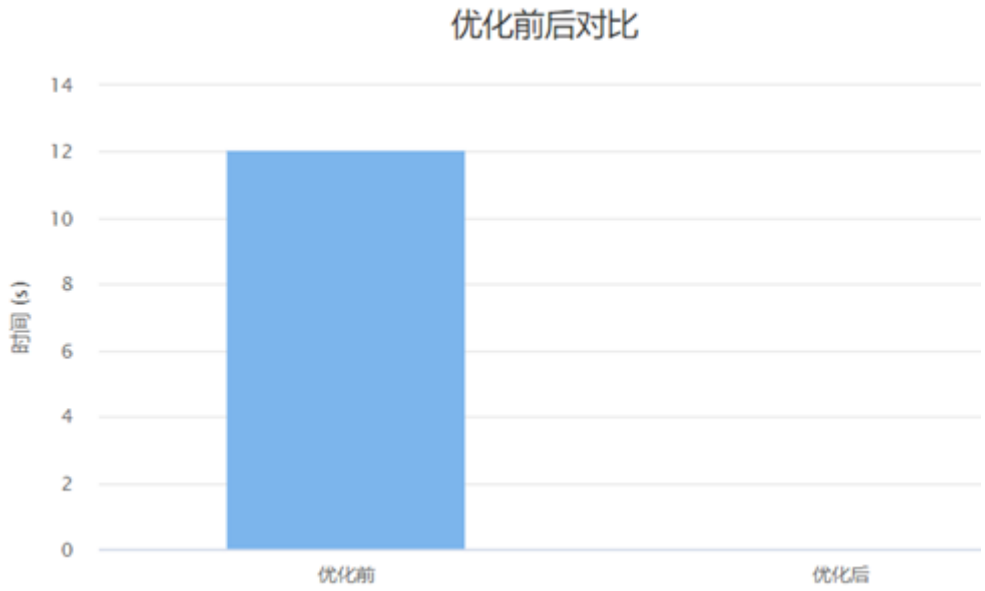
10用户并发 迭代10次 （大数据）

优化前：

```
C:\Users\97537>mysqlslap --no-defaults -u root --create-schema=old_large_data --query="call queryOrder(1)" -c 100 -i 5 -
p586970
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
^C
C:\Users\97537>mysqlslap --no-defaults -u root --create-schema=old_large_data --query="call queryOrder(1)" -c 10 -i 5 -p
586970
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
Benchmark
        Average number of seconds to run all queries: 12.087 seconds
        Minimum number of seconds to run all queries: 9.640 seconds
        Maximum number of seconds to run all queries: 13.735 seconds
        Number of clients running queries: 10
        Average number of queries per client: 1
```

优化后：

```
C:\Users\97537>mysqlslap --no-defaults -u root --create-schema=new_large_data --query="call queryOrder(1)" -c 10 -i 5 -p
586970
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
Benchmark
        Average number of seconds to run all queries: 0.012 seconds
        Minimum number of seconds to run all queries: 0.000 seconds
        Maximum number of seconds to run all queries: 0.032 seconds
        Number of clients running queries: 10
        Average number of queries per client: 1
```



③ 利用explain分析性能

优化前：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	boarding	(Null)	ref	FK_belongOrder,F	FK_take	4	const	248956	100	Using ind
1	SIMPLE	order	(Null)	eq_ref	PRIMARY,FK_buy	PRIMAR	4	old_larg	1	50	Using whe

优化后：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	ref	PRIMARY,FK_buy,o	FK_buy	4	const	25305	100	(Null)
1	SIMPLE	boarding	(Null)	ref	FK_belongOrder	FK_belongOrder	4	new_lar	14	100	(Null)

对比possible_keys，可以看到优化后的函数使用了新建的索引。

对比rows可以发现优化后的函数检索行数大大减少。

· queryFlight 查询航班

①函数优化

1. 由于查询条件是根据日期和起飞降落地的，所以给departureAirport, arrivalAirport建立了索引
2. 由于如果将某个信息设为允许null会增加存储查询成本，所以在还不知道actualDeparture actualArrival时间时，填为scheduleDepartual scheduleArrival
3. 由于exists的查询效率比in高， 将关键字in 改为了exists

   优化前函数：

```
create
    definer = root@localhost procedure queryFlight(IN date varchar(20), IN
departureAirport varchar(30),
                                                   IN arrivalAirprot
varchar(30))
select *
from flight
where flight.routeId in (select routeId from route where
DATE(route.departureTime)=date and route.departureAirport=departureAirport
and route.arrivalAirport=arrivalAirport)
;
```

优化后函数：

```
create
    definer = root@localhost procedure queryFlight(IN date varchar(20), IN
departureAirport varchar(30),
                                                   IN arrivalAirprot
varchar(30))
begin
    select *
    from flight
    where exists(select routeId from route where
DATE(route.departureTime)=date and route.departureAirport=departureAirport
and route.arrivalAirport=arrivalAirport and route.routeId=flight.routeId);
end;
```

② 利用mysqlslap进行性能测试

优化前：

```
C:\Users\97537>mysqlslap --no-defaults -u root --create-schema=old_large_data --query="call queryFlight('2018
-09-10','cjkfghjklbnm','fghjklvbnmrtjkff')" -c 500 -i 10 -p586970
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
Benchmark
        Average number of seconds to run all queries: 0.953 seconds
        Minimum number of seconds to run all queries: 0.141 seconds
        Maximum number of seconds to run all queries: 2.156 seconds
        Number of clients running queries: 500
        Average number of queries per client: 1
```

优化后：

```
C:\Users\97537>mysqlslap --no-defaults -u root --create-schema=new_large_data --query="call queryFlight('2
-09-10','cjkfghjklbnm','fghjklvbnmrtjkff')" -c 500 -i 10 -p586970
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
Benchmark
        Average number of seconds to run all queries: 0.709 seconds
        Minimum number of seconds to run all queries: 0.187 seconds
        Maximum number of seconds to run all queries: 1.765 seconds
        Number of clients running queries: 500
        Average number of queries per client: 1
```

③ 利用explain分析性能

优化前：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	route	(Null)	ALL	PRIMARY	(Null)	(Null)	(Null)	50	2	Using whe
1	SIMPLE	flight	(Null)	ref	FK_belongRoute	FK_belor	4	airplane	10	100	(Null)

优化后：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	flight	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	500	100	Using whe
2	DEPENDENT SU	route	(Null)	eq_ref	PRIMARY,departureAirport_index	PRIMAR	4	airplane	1	5	Using whe

对比possible_keys，可以看到优化后的函数使用了新建的索引，自建的索引发挥了作用。

对比type,可以看到优化后提升为eq_ref

*注：其他查询函数queryUnfinishedTake 、queryFinishTake和queryFlight大同小异，就不再赘述。

2. 查询缓存

① 什么是查询缓存：

MySQL查询缓存保存查询返回的完整结果。当查询命中该缓存，MySQL会like返回结果，跳过了解析、优化和执行截断。

查询缓存系统会跟踪查询中涉及的每个表，如果这些表发生变化，那么和这个表相关的所有的缓存数据都将失效。这种机制效率看起来比较低，因为数据表变化时很有可能对应的查询结果没有变更，但是这种简单实现代价很小，而这点对于一个非常繁忙的系统来说非常重要。

随着现在的通用服务器越来越强大，查询缓存被发现是一个影响服务器扩展性的因素。它可能成为整个服务器的资源竞争单点，在多核服务器上还可能导致服务器僵死。所以大部分时候应该默认关闭查询缓存，如果查询缓存作用很大的话，可以配置个几十兆的小缓存空间。

② 使用方法

1. 在mysql安装目录下的my.ini(windows)中进行修改，在其中加入“query_cache_type=1”后重启mysql可启用缓存
2. SHOW VARIABLES LIKE '%query_cache%';可检查当前查询缓存相关配置参数
3. 参数解释：

   query_cache_type: 查询缓存类型,是否打开缓存

   可选项

   a、0(OFF)：关闭 Query Cache 功能，任何情况下都不会使用 Query Cache；
   b、1(ON)：开启 Query Cache 功能，但是当SELECT语句中使用SQL_NO_CACHE提示后，将不使用Query Cache；
   c、2(DEMAND)：开启Query Cache 功能，但是只有当SELECT语句中使用了SQL_CACHE 提示后，才使用Query Cache。

   备注1：

   如果query_cache_type为on而又不想利用查询缓存中的数据，可以用下面的SQL：

   SELECT SQL_NO_CACHE * FROM my_table WHERE condition;

   如果值为2，要使用缓存的话，需要使用SQL_CACHE开关参数：

   SELECT SQL_CACHE * FROM my_table WHERE condition;

   query_cache_size: 缓存使用的总内存空间大小,单位是字节,这个值必须是1024的整数倍,否则MySQL 会自动调整降低最小量以达到1024的倍数；(感觉这个应该跟文件系统的blcok大小有关)

   query_cache_min_res_unit: 分配内存块时的最小单位大小，设置查询缓存Query Cache每次分配内存的最小空间大小，即每个查询的缓存最小占用的内存空间大小；

query_cache_limit: 允许缓存的单条查询结果集的最大容量，默认是1MB，超过此参数设置的查询结果集将不会被缓存；

query_cache_wlock_invalidate: 如果某个数据表被锁住,是否仍然从缓存中返回数据,默认是OFF,表示仍然可以返回

控制当有写锁定发生在表上的时刻是否先失效该表相关的Query Cache，如果设置为 1(TRUE)，则在写锁定的同时将失效该表相关的所有Query Cache，如果设置为0(FALSE)则在锁定时刻仍然允许读取该表相关的Query Cache。

③ 测试配置：

Variable_name	Value
have_query_cache	YES
query_cache_limit	26843545(
query_cache_min_re	4096
query_cache_size	26843545(
query_cache_type	ON
query_cache_wlock_	OFF

④ 测试结果

开启缓存后多次测试时间对比

```
C:\Users\liumi>mysqlslap --no-defaults -u root --create-schema=gmqairplane --query="select * from (select * from boardin
gpass where boardingpass.CustomerId=1) as B,(select * from flight natural join route) as FR where (select F.actualArriva
l from flight as F where B.flightId=F.flightId)<FR.actualDeparture;" -c 5 i 10
Benchmark
        Average number of seconds to run all queries: 22.313 seconds
        Minimum number of seconds to run all queries: 22.313 seconds
        Maximum number of seconds to run all queries: 22.313 seconds
        Number of clients running queries: 5
        Average number of queries per client: 1


C:\Users\liumi>mysqlslap --no-defaults -u root --create-schema=gmqairplane --query="select * from (select * from boardin
gpass where boardingpass.CustomerId=2) as B,(select * from flight natural join route) as FR where (select F.actualArriva
l from flight as F where B.flightId=F.flightId)<FR.actualDeparture;" -c 5 i 10
Benchmark
        Average number of seconds to run all queries: 0.015 seconds
        Minimum number of seconds to run all queries: 0.015 seconds
        Maximum number of seconds to run all queries: 0.015 seconds
        Number of clients running queries: 5
        Average number of queries per client: 1
```

对比结果可以看到，打开缓存后第一次操作时间需要22.313秒，而第二次测试时速度明显提升，只需要0.015秒，缓存大大提升了查询的速度。

值得注意的是，call function无法触发查询缓存机制，其不产生cache_inserts，原因未知。在理想的实际使用中，应是包装好的软件直接使用sql语句而非调用函数，故仍然可用。

3.数据库存储引擎

① MYSQL5.7中的引擎

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, ro	YES	YES	YES
MRG_MYISAM	YES	Collection of identical My	NO	NO	NO
MEMORY	YES	Hash based, stored in me	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO

InnoDB存储引擎介绍

Mysql版本>=5.5 默认的存储引擎，MySQL推荐使用的存储引擎。支持事务，行级锁定，外键约束。事务安全型存储引擎。更加注重数据的完整性和安全性。

MyISAM存储引擎介绍

MySQL<= 5.5 MySQL默认的存储引擎。

ISAM：Indexed Sequential Access Method（索引顺序）的缩写，是一种文件系统。

擅长与处理，高速读与写。

myisam支持全文索引，innodb在5.6之后支持

myisam可以用myisamPack压缩数据，节省磁盘空间，缺点是压缩后需要重新修复索引且变为只读，修改需要重新解压。非常适用于route和过往记录等极少修改的数据的储存

关于Innodb 和myisam的取舍：

Innodb ： 数据完整性，并发性处理，擅长更新，删除。

myisam：高速查询及插入。擅长插入和查询，**不支持事务。**

② 测试

```
C:\Users\liumi>mysqlslap --no-defaults -u root -a -c 1 i 100 --engine=myisam,innodb
Benchmark
        Running for engine myisam
        Average number of seconds to run all queries: 0.219 seconds
        Minimum number of seconds to run all queries: 0.219 seconds
        Maximum number of seconds to run all queries: 0.219 seconds
        Number of clients running queries: 1
        Average number of queries per client: 0

Benchmark
        Running for engine innodb
        Average number of seconds to run all queries: 0.390 seconds
        Minimum number of seconds to run all queries: 0.390 seconds
        Maximum number of seconds to run all queries: 0.390 seconds
        Number of clients running queries: 1
        Average number of queries per client: 0
```

```
C:\Users\liumi>mysqlslap --no-defaults -u root --create-schema=gmqairplane --query="call queryOrder(1)" -c 10 i 5 --engi
ne=myisam,innodb
Benchmark
        Running for engine myisam
        Average number of seconds to run all queries: 33.828 seconds
        Minimum number of seconds to run all queries: 33.828 seconds
        Maximum number of seconds to run all queries: 33.828 seconds
        Number of clients running queries: 10
        Average number of queries per client: 1

Benchmark
        Running for engine innodb
        Average number of seconds to run all queries: 35.766 seconds
        Minimum number of seconds to run all queries: 35.766 seconds
        Maximum number of seconds to run all queries: 35.766 seconds
        Number of clients running queries: 10
        Average number of queries per client: 1
```

由上图测试结果可以发现，在低并发性的时候两者的效率相近，且myisam始终略优。

但是在高并发的情况下innodb对myisam体现出显著优势：

```
C:\Users\liumi>mysqlslap --no-defaults -u root -a -c 100 i 100 --engine=myisam,innodb
Benchmark
        Running for engine myisam
        Average number of seconds to run all queries: 51.453 seconds
        Minimum number of seconds to run all queries: 51.453 seconds
        Maximum number of seconds to run all queries: 51.453 seconds
        Number of clients running queries: 100
        Average number of queries per client: 0

Benchmark
        Running for engine innodb
        Average number of seconds to run all queries: 5.500 seconds
        Minimum number of seconds to run all queries: 5.500 seconds
        Maximum number of seconds to run all queries: 5.500 seconds
        Number of clients running queries: 100
        Average number of queries per client: 0
```

③对于我们的航空出行管理系统，如何选择

1. 由于航空出行管理系统需要经常进行数据的修改和删除，InnoDB较为适合
2. 在进行变更时，如果失败需要回滚必须用到事务，InnoDB较为适合
3. 每个用户账户数据的完整性和同步性非常重要，需要外键支持，否则会导致混乱，InnoDB较为适合。

四、总结

查询优化：

1. 建立索引
2. 将部分范围固定的数据类型从varchar改为enum
3. 用exists代替in
4. 减少join操作，并先进行select再join，减少join操作需要操作的函数
5. 把所有字段都改为not null

查询缓存：

多次相同或者相近的查询则会由于cache_hits，速度大大加快

存储引擎：

低并发性的时候两者的效率相近，且myisam始终略优

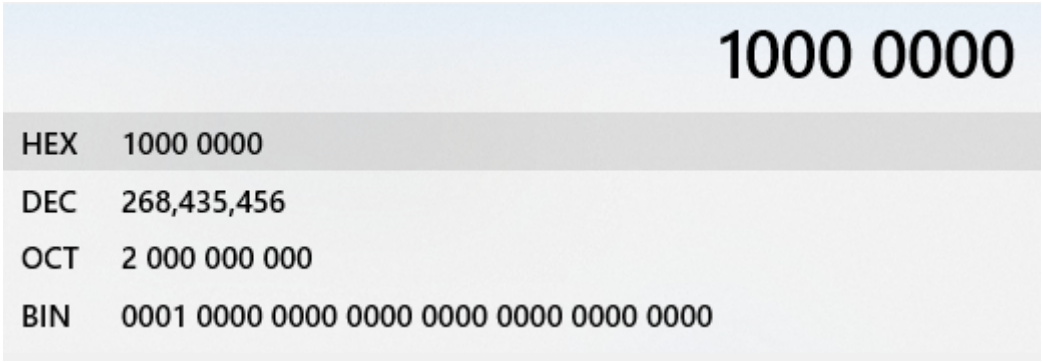
但是在高并发的情况下innodb对myisam体现出显著优势

且由于需要用到事务，支持高并发，经常修改，此系统更适合InnoDB

测试：

最后进行汇总对比。

在缓存属性为如下图的情况下：



优化前无缓存+两种引擎对比：

```
C:\Users\liumi>mysqlslap --no-defaults -u root --create-schema=airplane_old_large --query="select `order`.orderId,time,b
oardId,flightId,number,seat,type,price from `order` join boardingpass where `order`.orderId=boardingpass.orderId and `or
der`.CustomerId=1;" -c 50 i 5 --engine=myisam,innodb
Benchmark
        Running for engine myisam
        Average number of seconds to run all queries: 31.454 seconds
        Minimum number of seconds to run all queries: 31.454 seconds
        Maximum number of seconds to run all queries: 31.454 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1

Benchmark
        Running for engine innodb
        Average number of seconds to run all queries: 20.484 seconds
        Minimum number of seconds to run all queries: 20.484 seconds
        Maximum number of seconds to run all queries: 20.484 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1
```

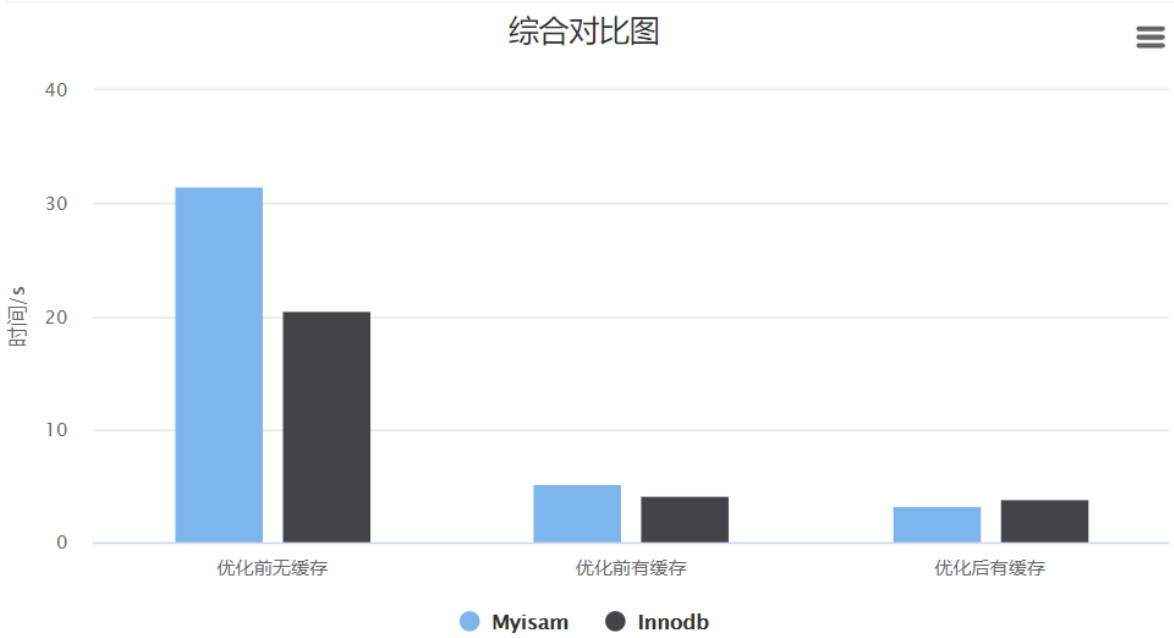
优化前有缓存+两种引擎对比：

```
C:\Users\liumi>mysqlslap --no-defaults -u root --create-schema=airplane_old_large --query="select `order`.orderId,time,b
oardId,flightId,number,seat,type,price from `order` join boardingpass where `order`.orderId=boardingpass.orderId and `or
der`.CustomerId=1;" -c 50 i 5 --engine=myisam,innodb
Benchmark
        Running for engine myisam
        Average number of seconds to run all queries: 5.156 seconds
        Minimum number of seconds to run all queries: 5.156 seconds
        Maximum number of seconds to run all queries: 5.156 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1

Benchmark
        Running for engine innodb
        Average number of seconds to run all queries: 4.203 seconds
        Minimum number of seconds to run all queries: 4.203 seconds
        Maximum number of seconds to run all queries: 4.203 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1
```

优化后有缓存+两种引擎对比：

```
C:\Users\liumi>mysqlslap --no-defaults -u root --create-schema=airplane_new_large --query="select t1.orderId,time,boardI
d,flightId,number,seat,type,price from (select orderId,time from `order` where ( `order`.CustomerId=1)) t1, (select boar
dId,orderId,flightId,number,seat,type,price from boardingpass where boardingpass.CustomerId=1)  t2 where t1.orderId=t2.o
rderId;" -c 50 i 5 --engine=myisam,innodb
Benchmark
        Running for engine myisam
        Average number of seconds to run all queries: 3.813 seconds
        Minimum number of seconds to run all queries: 3.813 seconds
        Maximum number of seconds to run all queries: 3.813 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1

Benchmark
        Running for engine innodb
        Average number of seconds to run all queries: 3.875 seconds
        Minimum number of seconds to run all queries: 3.875 seconds
        Maximum number of seconds to run all queries: 3.875 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1
```

可以看到针对我们的航空出行管理系统，进行优化和开启缓存，并使用innodb引擎，可以即达提升性能。


综合对比图
40
30
20
10
0
时间/s
优化前无缓存
优化前有缓存
优化后有缓存
Myisam
Innodb