Nov 9, 2021

版本: 1.0

同济大学 TONGJI UNIVERSITY

内存数据库实验方案

归宿——一个民宿预定平台

指导教授：**袁时金**

1851231 王立友

1852612 周涵嵩

1952560 安江涛

1953068 高天宸

1954098 香宁雨

In-memory Database

Tongji University

School of Software Engineering

目录

实验一 数据库方案对比实验 1

在不同数据量时，对于不同数据库的选择可能影响数据库查询的速度以及数据库的性能。因此可以对不同数据库进行其性能对比实验。

1.1 实验目的

对于TimesTen而言，所有的数据都会被加载到内存中，而对于Oracle而言，数据的处理可能是先从辅存读入内存再进行处理。在小数据量和大数据量不同时，TimesTen和Oracle的表现可能不同。所以我们对TimesTen和Oracle分别进行查询操作，分析二者在相同数据量时性能的差异。在归宿中存在大量对数据的查询业务，而首要任务是对数据库进行选择。我们将对比Oracle和TimesTen在千万级表和万级表之间的查询速度来进行对数据库的选择。

1.2 实验方案

1.2.1 订单信息查询

数据表：Order

Order表为本项目中千万级数据表，我们通过查询所有订单的数量来进行Oracle和TimesTen在千万级数据表上性能的比较。

查询SQL语句如下：

```
select count(*)
from order;
```

实现结果：

数据库	查询时间
ORACLE	
TIMESTEN	

Tab. 1.1: 数据库对比查询实验实验记录表格

1.2.2 用户信息查询

数据表：Customer

Customer表为本项目中万级数据表，我们通过查询所有订单的数量来进行Oracle和

TimesTen在万级数据表上性能的比较。

查询SQL语句如下：

```
select count(*)
from customer;
```

实现结果如下：

数据库	查询时间
ORACLE	
TIMESTEN	

Tab. 1.2: 数据库对比查询实验记录表格

实验二 查询优化实验

2

通常对于同一个目标，我们会有多种SQL查询语句的写法，但是各种写法的实际执行效率并不相同。为此，我们针对本系统的主要的查询业务，来进行SQL语句级别的优化实验。

2.1 实验目的

订单管理系统连接的是卖家用户与买家用户，对于这两方用户的使用来说，都存在大量的查询业务，这也是整个系统并发量最大的业务。所以我们需要对查询过程进行详细的优化设计，来提升用户的使用体验。

2.2 实验方案

2.2.1 房东订单数据查询

数据表：order, order_room, room, stay

优化策略：房东订单信息查询涉及到多表查询，很多时候我们都需要用join来拼接多个表。在join操作中的优化规则是：数据行较少的表在左边，数据行较多的表在右边。在join当中，是使用左边表的每一个数据行去扫描右边的整个表的所有数据行，所以虽然总的匹配次数是相同的，但是如果左边表数据行很多，则需要加载右边的整个表很多次，使用小表驱动大表主要是减少这个次数，来提高性能，在本实验中，我们将在Oracle数据库中分别对于 Hash Join 和 Nested Loop 两种表连接方式，分别按照不同的顺序进行表的连接并进行查询。

在Hash join中，优化器使用两个表中较小的表（通常是小一点的那个表或数据源）利用连接键在内存中建立散列表，将列数据存储到hash列表中，然后扫描较大的表，同样对JOIN KEY进行HASH后探测散列表，找出与散列表匹配的行。

在使用Hash Join的情况下，优化前查询SQL语句如下：

```
select count(distinct(order_id))
from order_room natural join order natural join room  natural join stay
where host_id = id
```

在使用Hash Join的情况下，优化后查询SQL语句如下：

```
select count(distinct(order_id))
from stay natural join room natural join order  natural join order_room
where host_id = id
```

Nested loop的工作方式是循环从一张表中读取数据(驱动表outer table)，然后访问另一张表（被查找表 inner table ,通常有索引）。驱动表中的每一行与inner表中的相应记录JOIN。类似一个嵌套的循环。

在使用Nested Loop的情况下，优化前查询SQL语句如下：

```
select count(distinct(order_id)
from  order_room, order, room, stay
where order_room.order_id = order.order_id and
      order_room.room_id = room.room_id and
      room.stay_id = stay.stay_id and
      stay.host_id = id
```

在使用Nested Loop的情况下，优化后查询SQL语句如下：

```
select count(distinct(order_id)
from  stay, room, order, order_room
where order_room.order_id = order.order_id and
      order_room.room_id = room.room_id and
      room.stay_id = stay.stay_id and
      stay.host_id = id
```

实验结果如下：

数据库	优化前	优化后
ORACLE		
TIMESTEN		

Tab. 2.1: 房东订单数据查询优化实验(Hash Join)部分

数据库	优化前	优化后
ORACLE		
TIMESTEN		

Tab. 2.2: 房东订单数据查询优化实验(Nested Loop)部分

2.2.2 房源订单数据查询

数据表：order, order_room, room, stay

优化策略：查询时需要指定房源表的 stay_id 。Oracle 采用自下而上或自右向左的顺序解析 WHERE 子句。根据这个原理,表之间的连接必须写在其他 WHERE 条件之前，那些可以过滤掉最大数量记录的条件必须写在WHERE 子句的末尾。

优化前查询SQL语句如下：

```
select count(distinct(order_id)
from  stay, room, order, order_room
where stay.host_id = id and
      order_room.room_id = room.room_id and
      room.stay_id = stay.stay_id and
      order_room.order_id = order.order_id
```

优化后查询SQL语句如下：

```
select count(distinct(order_id)
from  stay, room, order, order_room
where order_room.order_id = order.order_id and
      order_room.room_id = room.room_id and
      room.stay_id = stay.stay_id and
      stay.host_id = id
```

实验结果如下：

数据库	优化前	优化后
ORACLE		
TIMESTEN		

Tab. 2.3: 房源订单数据查询优化实验

2.2.3 房东订单信息查询

数据表： order, order_room, room, stay

优化策略：查询房东订单的信息，查询结果需要对订单ID字段去重。去重查询可以有两种查询方案：通过 select distinct 进行查询；先进行 group by，再进行 select。通过比较两种查询方案的实验结果，分析性能上的差异。

使用 distinct 的SQL语句如下：

```
select distinct order_id
from stay natural join room natural join order  natural join order_room
where host_id = id
```

使用 group by 的查询SQL语句如下：

```
select order_id
from stay natural join room natural join order  natural join order_room
where host_id = id
group by order_id
```

数据库	优化前	优化后
ORACLE		
TIMESTEN		

Tab. 2.4: 房东订单信息查询优化实验

实验三 索引优化实验

3

3.1 实验目的

根据实际的应用场景，通过对不同字段进行组合，在 Oracle 和 TimesTen 数据库中分别建立不同类型的索引，并进行查询和插入两种操作检验性能的变化，最终找到适用于不同数据库的最合理的索引方案，达到提高系统性能的目的。

3.2 实验方案

3.2.1 针对评论等级建立索引

查询SQL语句：

```
select count(*)
from comment
where comment_grade = 1
```

插入SQL语句：

```
insert into comment
values (COMMENT_INC.nextval, null, 1, '2021/11/8 15:56:52')
```

TimesTen部分：

无索引

无需额外操作

范围索引

```
drop index comment_index;
create index comment_index
on order(comment_grade)
```

位图索引

```
drop index comment_index;
create bitmap index comment_index
on order(comment_grade)
```

哈希索引

```
drop index comment_index;
create hash index comment_index
on order(comment_grade)
```

在创建索引之后分别进行查询和插入操作。

实验结果如下：

索引类型	查询操作	插入操作
无索引		
范围索引		
位图索引		
哈希索引		

Tab. 3.1: 针对评论等级建立索引的实验记录（TimesTen）

Oracle部分：

在创建索引之后分别进行查询和插入操作。

实验结果如下：

索引类型	查询操作	插入操作
无索引		
范围索引		
位图索引		

Tab. 3.2: 针对评论等级建立索引的实验记录（Oracle）

实验四 批量写入实验

4

本系统会有集中性的高并发场景。因此会需要进行大规模批量写入实验，便于开展对多线程效果的测试，其实验结果也是有效评估数据库性能的重要指标。

4.1 实验目的

针对用户收藏房源这一业务情景，进行大规模单表写入实验，测试系统在单线程和多线程环境下的操作用时，从而评估系统的单表批量写入能力。

针对用户下单房源这一业务情景，进行大规模多表插入实时更新实验，测试系统分别在单线程和多线程环境下的操作用时，从而评估系统的多表批量插入更新能力。

利用多线程实现数据库查询时，不同的线程数会带来不同的查询性能。通过进行线程数的对比实验，探究不同业务情景下线程数对数据库性能的影响。

4.2 实验方案

4.2.1 单表批量写入实验

业务情景：用户收藏房源时，需要向收藏夹单表导入大量数据。

实验说明：设计不同的线程数进行实验，对比不同线程方案的实现。

数据表：Favorites

实验代码如下：

实验结果：

线程数	1,000条	10,000条	100,000条
1（单线程）			
2（多线程）			
3（多线程）			
4（多线程）			
5（多线程）			

Tab. 4.1: 单表批量写入实验记录表格

4.2.2 多表批量写入实验

业务情景：用户下单房源时，需要向订单表、房间表、房源表级连写入更新大量数据。

实验说明：设计不同的线程数进行实验，对比不同线程方案的实现。

数据表：Order、Room、Stay

实验代码如下：

实验结果：

线程数	1,000条	10,000条	100,000条
1（单线程）			
2（多线程）			
3（多线程）			
4（多线程）			
5（多线程）			

Tab. 4.2: 多表批量写入实验记录表格

实验五 数据报表实验

5

数据报表实验涉及多表联合查询，并且需要配合使用聚合函数对表字段进行统计分析。因此大数据量下数据报表查询操作可以很好地反映数据库查询与优化性能。

5.1 实验目的

一方面民宿预定平台具有用户查看个人行程住宿统计信息的功能，提供用户订单交易统计信息、与出行住宿等统计信息，另一方面民宿预定平台提供给民宿房主查看统计个人民宿出租收益统计数据的信息，使老板根据民宿的出租情况与利益对民宿的基础设施与价格进行调整。这两方面功能都是以数据报表为基础实现，因此通过对两类功能实验进行性能对比分析。

5.2 实验方案

5.2.1 用户查看民宿与订单统计数据报表

业务情景：根据用户的标识信息，筛选出给定时间范围(以月份为基本单位)内该用户的总订单数量、总订单金额、最大交易金额订单信息、最大交易金额相关民宿信息、旅居民宿次数最多的省份等信息，根据以上筛选与统计结果生成数据报表返回用户。

返回数据格式如下：

```
{
    "total_order_num":100,
    "total_order_amount":10045.8,
    "maximum_amount_order":{
        "order_id":100055,
        "order_time":"2021-10-11",
        "order_total_amount":2000
    },
    "maximum_amount_stay":[{
        "stay_id":100054,
        "room_id":201,
        "room_price":199,
        "room_photo":"xxx.png",
        "start_time":"2021-11-11",
        "end_time":"2021-11-14"
    },{
        "stay_id":100054,
        "room_id":202,
        "room_price":399,
        "room_photo":"xxx.png",
        "start_time":"2021-11-11",
```

```
        "end_time":"2021-11-14"
    }],
    "maximum_time_province":"上海"
}
```

实现结果如下：

数据库	查询时间
ORACLE	
TIMESTEN	

Tab. 5.1: 用户查看民宿与订单统计数据报表实验记录表格

5.2.2 房主查看房源统计出租收益报表

业务情景：根据房主的标识信息，筛选出一定时间范围内房主名下房源的出租情况，如统计房源订单总数量、房源订单总金额、最大交易金额订单信息、最大交易金额房源信息、最受欢迎房源类型、最受欢迎价格区间等信息，根据以上筛选与统计结果生成数据报表返回房主。

返回数据格式如下：

```
{
    "total_order_num":233,
    "total_order_amount":50045.8,
    "maximum_amount_order":{
        "order_id":100055,
        "order_time":"2021-10-11",
        "order_total_amount":2000
    },
    "maximum_amount_stay":[{
        "stay_id":100054,
        "room_id":201,
        "room_price":199,
        "room_photo":"xxx.png",
        "start_time":"2021-11-11",
        "end_time":"2021-11-14"
    },{
        "stay_id":100054,
        "room_id":202,
        "room_price":399,
        "room_photo":"xxx.png",
        "start_time":"2021-11-11",
        "end_time":"2021-11-14"
    }],
    "popular_stay_description":"安静舒适",
    "popular_price_section":"300-350"
}
```

实现结果如下：

数据库	查询时间
ORACLE	
TIMESTEN	

Tab. 5.2: 房主查看房源统计出租收益报表

5.3 实验设计思考

数据报表实验设计目的在于处理复杂的业务逻辑需求，其中需要多方面对数据存储、数据读取的优化与设计。另一方面，数据报表实验中可能遇到的问题在于多个统计量的查询与运算的过程会对效率有很大的损耗，需要利用存储过程、函数以及对底层表优化实现查询目标。

实验六 SQL脚本与过程对比实验 6

SQL脚本是点对点运行的，且未被预编译；存储过程是预编译并存储在服务器上以供重用的一组SQL命令集，相比之下，存储过程的运行性能会优于SQL脚本。

6.1 实验目的

通过分别比较Oracle和TimesTen数据库下SQL脚本和存储过程带来的处理时间差异，分析两种方法的性能表现。

6.2 实验方案

6.2.1 生成订单统计报表

业务情景:统计平台订单的平均成交金额、热门地区

数据表：order, room, stay

优化策略：使用存储过程代替单纯的SQL脚本

优化前：

```
select avg(total_price)
from order;

select stay_address, count(*) as order_num
from `order` natural join room natural join stay
group by stay_address
having count(*) > 10000;
```

优化后：

```
create procedure get_order_report()
begin
  select avg(total_price) from `order`;
  select stay_address, count(*) as order_num
  from `order` natural join room natural join stay
  group by stay_address
  having count(*) > 10000;
end;
call get_order_report();
```

预期实验结果如下：

数据库	SQL语句	存储过程
ORACLE		
TIMESTEN		

Tab. 6.3: 生成统计报表实验记录

6.3 实验设计思考

SQL脚本是即开即用，用完即删的一次性过程，而存储过程在第一次使用时被载入内存，下次使用会大大加快执行速度。SQL脚本只能执行单一操作，而存储过程可以互相调用，引用其他存储过程，更加灵活，使用更加方便，但修改起来更加困难。

实验七 添加冗余表

7

通过增加冗余表，可以降低外码和索引的数目，减少计算量，提高数据库查询性能，但同时也会破坏数据库范式，带来数据冗余。

7.1 实验目的

本实验探究添加冗余表相较于普通方法所带来的性能提升率，并分析添加冗余的代价，可能产生的问题。

7.2 实验方案

数据表： order, order_room, room, stay

优化策略：房东经常需要查看自己的订单数据，但是查询房东订单时需要order, order_room, room, stay四张表一起join，开销非常大。我们可以增加一个房东订单数据表，这样在查询房东订单数据时，只需要在本表里查询即可，避免了join的开销。

添加冗余表前查询SQL语句如下：

```
select count(distinct(order_id))
from order_room natural join order natural join room  natural join stay
where host_id = id
```

添加冗余表后查询SQL语句如下：

```
select count(distinct(order_id))
from host_order
where host_id = id
```

实验结果：

数据库	优化前	优化后
ORACLE		
TIMESTEN		

Tab. 7.1: 增加冗余表实验

实验八 增加冗余操作

8

当进行联表查询时，如果频繁通过外键查询某一字段，且该字段在两表中表示的意义完全相同，此时需要进行冗余设计，将部分字段冗余到关联表中，避免大表之间的关联查询，提高响应速度。

8.1 实验目的

探究冗余操作相较于未使用时所带来的性能变化，以及分析该方法的优劣。

8.2 实验方案

8.2.1 增加冗余列

业务场景：顾客搜索房源时，结果列表展示房东信息（包括姓名），需要以房源表host_id为外键查询host表中的host_name字段。

数据表：stay, host

优化策略：在房源表中加入房东姓名作为冗余列，这样可以避免在查询时对stay和host表进行join操作，从而优化性能。

预期实验结果如下：

查询策略	花费时间
不增加冗余列	
增加冗余列	

Tab. 8.1: 增加冗余列实验

8.3 实验设计思考

适当增加冗余列会使得部分查询操作时速度加快，但其不符合第三范式，过多添加会使得数据库冗余数据增多，反而降低某些查询操作效率。因此，在实际数据库设计中，需要谨慎考虑，适当采取该措施，优化查询性能。

实验九 冷热对比实验

9

星级越高的房源越容易被频繁查询，星级低的房源只会被偶尔查询。通过冷热数据对比实验可以选择最佳的冷热数据比，评估优化前和优化后的查询性能对比。

9.1 实验目的

对于不同评价星级的房源来说，其数据访问频次的差距是很明显的，而对于内存数据库而言， 所有数据都存储在内存中，无法承受那么大的数据量。星级高的房源会被经常访问，而星级低的房源访问频率会变得很低，经常访问的热数据因为访问频次需求大，效率要求高，所以可以就近计算和部署，而访问频率低的冷数据对效率要求并不如热数据，所以冷热数据可以进行分开存储，来提高系统的整体性能。

9.2 实验方案

9.2.1 根据星级进行冷热数据区分

数据表：Stay

将星级高的房源数据放入TimesTen中，将其余星级较低的房源数据放入Oracle中，实现冷热数据在星级方面的区分，理论而言会因为数据表大小的减小而对系统的性能有所提升，但其数据分配的比例点需要通过实验进行测试。

9.2.2 对冷热数据比例进行划分

基于给定的查询次数并且对高星级房源可能占全部查询的比例对冷热数据进行比例划分，并且根据不同比例将冷热数据存储在Oracle和TimesTen中，对不同比例数据的查询速度进行测试，寻找最佳的冷热数据比例。

实验结果：

查询高星级占比	冷热数据7：3所用时间	冷热数据6：4所用时间	冷热数据5：5所用时间
50%			
60%			
70%			

Tab. 9.1: 冷热数据查询实验记录表格

9.2.3 对查询次数与房源星级进行模拟

查询次数和房源星级的模拟图如下图所示，高星级房源的查询次数较高，而低星级的房源查询次数较低，实验会根据不同的查询次数比例对查询速度进行测试。

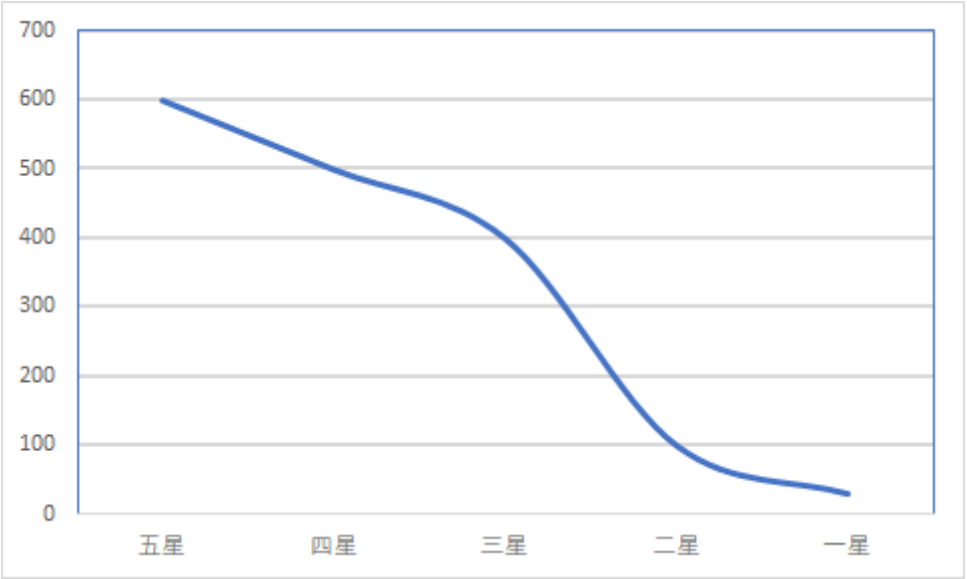

Fig. 9.1: 查询次数与星级模拟分布图

9.3 实验设计思考

本实验的难点在于如何对冷热数据的比例进行划分，当热数据表过大时，热数据表查询的时间会因为表的增大而增加，可能会减缓对于整个房源表的查询速度，而当冷数据表过大时，冷数据表中可能会包含很多本应存放在热数据表中的内容，在查询时可能会多次调用冷数据表，反而降低查询的效率。所以本次实验需要通过多次对比试验对冷热数据的划分进行对比，来确保划分冷热数据块确实能增快数据的查询速度。

实验十 事务对插入的影响实验

10

事务可以将多条插入语句并成一次进行commit，可以对插入的速度进行优化，在引入事务后可以对数据库的插入效率进行比较。

10.1 实验目的

对数据库而言，默认每次插入数据都会commit一次，也就是写入一次磁盘，开启事务相当于多次插入后进行一次commit，减少了对磁盘的读写，会加快数据库插入的速度和效率。为此，我们针对数据库的插入效率，对事务对其速率的影响进行实验，来判断怎样才能达到提高系统性能的目的。

10.2 实验方案

数据表：Order

通过对千万级数据表插入时所进行的时间测算来推定事务对插入时间的影响，将插入数据按照量级分为1w、10w、100w，分别进行使用事务和不使用事务时间的判断。

实验结果：

数据量级	使用事务插入时间	不使用事务插入时间
1w		
10w		
100w		

Tab. 10.1: 针对评论等级建立索引的实验记录（Oracle）

实验十一 数据分区实验

11

数据库中所有数据对象都放在指定的表空间中，当表中数据量不断增大时，查询速度相应会变慢。通过考虑对表进行分区操作，将表中数据在物理上存放到多个表空间中，提高查询指定数据时的性能，一定程度上减少全表扫描开销。

11.1 实验目的

民宿预定平台中存在如对某一时间范围内订单数据查询的功能，即根据下单时间选择订单信息。另一方面，也存在对民宿房源评论信息进行筛选，筛选出有实际评论信息的数据返回展示。可以采用索引对上述需求进行优化，但是另一方面可以采用数据分区的方式减少扫描次数，提高数据库的查询性能。

11.2 实验方案

11.2.1 范围分区：查询时间范围内的订单信息

业务情景：民宿预定平台中部分复杂业务逻辑实现是基于基础功能的优化，例如在生成数据报表时需要根据时间范围统计订单信息，在此处可以采用根据订单表中下单时间列字段的取值范围对订单表进行分区，通过测试相关SQL脚本完成性能实验对比。

查询下单时间在2021-11-10与2021-12-10之间，由标识码为10002的用户所下的订单信息，SQL执行语句如下：

```
select *
from order
where customer_id = "10002" and order_time >= "2021-11-10" and order_time <=
"2021-12-10"
```

预期实验结果如下：

数据库	不使用数据分区	使用数据分区
ORACLE		
TIMESTEN		

Tab. 11.1: 范围分区实验结果表格

11.2.2 列表分区：目的性对评论数据进行筛选查询

业务情景：民宿预定平台在实际生活中会遇到评价中仅有评价得分但没有评价内容的情况，在对房源进行展示时需要筛选出有实际内涵的评价信息，如果对全表进行扫描会比较产生比较大的开销，因此通过对列表分区的变种，根据评论内容的有无进行区分，将评论信息分区为两部分表，分别为有实际评论内容的表与无实际评论内容的表，进行性能实验对比测试。

通过查询房源标识码为100034的房源评价进行对分区实验的测试，其中SQL执行语句如下所示：

```
select customer_id,comment_cotent,comment_grade,comment_time
from comment nature join order nature join order_room
where stay_id = "100034"
```

预计实现结果如下：

数据库	不使用数据分区	使用数据分区
ORACLE		
TIMESTEN		

Tab. 11.2: 列表分区实验结果表格

11.3 实验设计思考

数据分区实验会对查找性能上有所优化，但无论选择范围分区优化或者列表分区优化，都是选择具有代表性的特殊的表字段进行筛选。相对于索引而言，没有额外的增删改开销，但是相对来说可能灵活性较差。实验可能遇到的困难在于分区带来的收益是否较大以及如何确定分区的数目以及分区适应的场景。