

# 测试说明文档

## 测试说明文档

### 题目二 查询执行

- 测试点1:尝试建表
- 测试点2:单表插入与条件查询
- 测试点3: 单表更新与条件查询

### 题目三 唯一索引

- 测试点1: 创建、删除、展示索引
- 测试点2: 索引查询
- 测试点3: 索引维护
- 测试点4: 是否真正使用单列索引来进行查询
- 测试点5: 是否真正使用多列索引来进行查询
- 测试文件说明

### 题目四 聚合函数与分组统计

- 测试点1: 单独使用聚合函数
- 测试点2: 聚合函数加分组统计
- 测试点3: 健壮性测试

### 题目五 不相关子查询

- 测试点1: 与子查询结果进行比较
- 测试点2: 带有IN谓词的子查询
- 测试点3: 健壮性测试

### 题目六 归并连接算法

### 题目七 事务控制语句

- 测试点
- 测试文件说明

### 题目八 冲突可串行化

- 测试点: 判断系统是否会出现五种数据异常
- 测试文件说明

### 题目九 细粒度锁: 间隙锁

- 测试点: 判断系统是否实现细粒度锁以及是否出现幻读数据异常
- 测试文件说明

### 题目十 基于静态检查点的故障恢复-测试方案

- 测试点说明
- 测试过程说明

## 题目二 查询执行

### 测试点1:尝试建表

测试示例:

```
create table t1(id int,name char(4));

show tables;

create table t2(id int);

show tables;
```

```
drop table t1;

show tables;

drop table t2;

show tables;
```

期待输出:

```
| Tables |
| t1 |
| Tables |
| t1 |
| t2 |
| Tables |
| t2 |
| Tables |
```

## 测试点2:单表插入与条件查询

测试示例:

```
create table grade (name char(20),id int,score float);

insert into grade values ('Data Structure', 1, 90.5);

insert into grade values ('Data Structure', 2, 95.0);

insert into grade values ('Calculus', 2, 92.0);

insert into grade values ('Calculus', 1, 88.5);

select * from grade;

select score,name,id from grade where score > 90;

select id from grade where name = 'Data Structure';

select name from grade where id = 2 and score > 90;
```

期待输出:

```
| name | id | score |
```

```

| Data Structure | 1 | 90.500000 |
| Data Structure | 2 | 95.000000 |
| Calculus | 2 | 92.000000 |
| Calculus | 1 | 88.500000 |

| score | name | id |
| 90.500000 | Data Structure | 1 |
| 95.000000 | Data Structure | 2 |
| 92.000000 | Calculus | 2 |

| id |
| 1 |
| 2 |

| name |
| Data Structure |
| Calculus |

```

### 测试点3：单表更新与条件查询

测试示例：

```

create table grade (name char(20),id int,score float);

insert into grade values ('Data Structure', 1, 90.5);

insert into grade values ('Data Structure', 2, 95.0);

insert into grade values ('Calculus', 2, 92.0);

insert into grade values ('Calculus', 1, 88.5);

select * from grade;

update grade set score = 90 where name = 'Calculus' ;

select * from grade;

update grade set name = 'Error name' where name > 'A';

select * from grade;

```

```
update grade set name = 'Error' ,id = -1,score = 0 where name = 'Error name' and score >= 90;
```

```
select * from grade;
```

期待输出:

```
| name | id | score |
| Data Structure | 1 | 90.500000 |
| Data Structure | 2 | 95.000000 |
| Calculus | 2 | 92.000000 |
| Calculus | 1 | 88.500000 |
| name | id | score |
| Data Structure | 1 | 90.500000 |
| Data Structure | 2 | 95.000000 |
| Calculus | 2 | 90.000000 |
| Calculus | 1 | 90.000000 |
| name | id | score |
| Error name | 1 | 90.500000 |
| Error name | 2 | 95.000000 |
| Error name | 2 | 90.000000 |
| Error name | 1 | 90.000000 |
| name | id | score |
| Error | -1 | 0.000000 |
| Error | -1 | 0.000000 |
| Error | -1 | 0.000000 |
| Error | -1 | 0.000000 |
```

## 题目三 唯一索引

### 测试点1：创建、删除、展示索引

测试示例：

```
create table warehouse (id int, name char(8));
create index warehouse (id);
show index from warehouse;
create index warehouse (id,name);
show index from warehouse;
drop index warehouse (id);
drop index warehouse (id,name);
show index from warehouse;
```

期待输出：

```
| warehouse | unique | (id) |
| warehouse | unique | (id) |
| warehouse | unique | (id,name) |
```

### 测试点2：索引查询

测试示例：

```
create table warehouse (w_id int, name char(8));
insert into warehouse values (10 , 'qweruiop');
insert into warehouse values (534, 'asdfhjk1');
insert into warehouse values (100,'qwerghjk');
insert into warehouse values (500,'bgtyhnmj');
create index warehouse(w_id);
select * from warehouse where w_id = 10;
select * from warehouse where w_id < 534 and w_id > 100;
drop index warehouse(w_id);
create index warehouse(name);
select * from warehouse where name = 'qweruiop';
select * from warehouse where name > 'qwerghjk';
select * from warehouse where name > 'aszdefgh' and name < 'qweraaaa';
drop index warehouse(name);
create index warehouse(w_id,name);
select * from warehouse where w_id = 100 and name = 'qwerghjk';
select * from warehouse where w_id < 600 and name > 'bztyhnmj';
```

期待输出：

```
| w_id | name |
| 10 | qweruiop |
| w_id | name |
| 500 | bgtyhnmj |
| w_id | name |
| 10 | qweruiop |
| w_id | name |
```

```
| 10 | qweruiop |
| w_id | name |
| 500 | bgtyhnmj |
| w_id | name |
| 100 | qwerghjk |
| w_id | name |
| 10 | qweruiop |
| 100 | qwerghjk |
```

## 测试点3：索引维护

测试示例：

```
create table warehouse (w_id int, name char(8));
insert into warehouse values (10, 'qweruiop');
insert into warehouse values (534, 'asdfhjk1');
select * from warehouse where w_id = 10;
select * from warehouse where w_id < 534 and w_id > 100;
create index warehouse(w_id);
insert into warehouse values (500, 'lastdanc');
insert into warehouse values (10, 'uiopqwer');
update warehouse set w_id = 507 where w_id = 534;
select * from warehouse where w_id = 10;
select * from warehouse where w_id < 534 and w_id > 100;
drop index warehouse(w_id);
create index warehouse(w_id,name);
insert into warehouse values(10,'qqqqoooo');
insert into warehouse values(500,'lastdanc');
update warehouse set w_id = 10, name = 'qqqqoooo' where w_id = 507 and name =
'asdfhjk1';
select * from warehouse;
```

期待输出：

```
| w_id | name |
| 10 | qweruiop |
| w_id | name |
failure
| w_id | name |
| 10 | qweruiop |
| w_id | name |
| 500 | lastdanc |
| 507 | asdfhjk1 |
failure
failure
| w_id | name |
| 10 | qqqqoooo |
| 10 | qweruiop |
| 500 | lastdanc |
| 507 | asdfhjk1 |
```

## 测试点4：是否真正使用单列索引来进行查询

测试示例：

```
create table warehouse (w_id int,name char(8));
insert into warehouse values(1,'12345678');
insert into warehouse values(2,'12345278');
...
insert into warehouse values(2999,'13345678');
insert into warehouse values(3000,'34245418');

-- 后台计时开始
select * from warehouse where w_id = 1;
select * from warehouse where w_id = 2;
...
select * from warehouse where w_id = 2999;
select * from warehouse where w_id = 3000;
-- 后台计时结束，对比期望输出

create index warehouse(w_id);

-- 后台计时开始
select * from warehouse where w_id = 1;
select * from warehouse where w_id = 2;
...
select * from warehouse where w_id = 2999;
select * from warehouse where w_id = 3000;
-- 后台计时结束，对比期望输出
-- 对比两次查询的耗时
```

期待输出：

```
| w_id | name |
| 1 | 12345678 |
| w_id | name |
| 2 | 12345278 |
...
| w_id | name |
| 2999 | 13345678 |
| w_id | name |
| 3000 | 34245418 |
```

## 测试点5：是否真正使用多列索引来进行查询

测试示例：

```
create table warehouse (w_id int,name char(8),flo float);
insert into warehouse values(1,'12345678',1024.5);
insert into warehouse values(2,'12345278',512.5);
...
insert into warehouse values(2999,'13345678',256.5);
insert into warehouse values(3000,'34245418',128.5);

-- 后台计时开始
```

```

select * from warehouse where w_id = 1 and flo = 1024.500000;
select * from warehouse where w_id = 2 and flo = 512.500000;
...
select * from warehouse where w_id = 2999 and flo = 256.500000;
select * from warehouse where w_id = 3000 and flo = 128.500000;
-- 后台计时结束，对比期望输出

create index warehouse(w_id,flo);

-- 后台计时开始
select * from warehouse where w_id = 1 and flo = 1024.500000;
select * from warehouse where w_id = 2 and flo = 512.500000;
...
select * from warehouse where w_id = 2999 and flo = 256.500000;
select * from warehouse where w_id = 3000 and flo = 128.500000;
-- 后台计时结束，对比期望输出
-- 对比两次查询的耗时

```

期待输出:

```

| w_id | name | flo |
| 1 | 12345678 | 1024.500000 |
| w_id | name | flo |
| 2 | 12345278 | 512.500000 |
...
| w_id | name | flo |
| 2999 | 13345678 | 256.500000 |
| w_id | name | flo |
| 3000 | 34245418 | 128.500000 |

```

## 测试文件说明

- storage\_test3: 创建、删除、展示索引。
- storage\_test4: 索引查询。
- storage\_test5: 索引维护。需要保证建有索引的表中，不存在任意两个元组具有相同的索引属性值，当要插入或者修改的元组违背了唯一索引的要求时向output.txt输入failure。在测试文件中，不存在以下情况：在某张表上建立索引前，表中已经存在两个元组具有相同的索引属性值。
- judge\_whether\_use\_index\_on\_single\_attribute: 创建表并插入数据后，进行大量单列查询，记录耗时time\_a，在某列创建索引，再次进行大量单列查询，记录耗时time\_b。若 $\text{time\_b} / \text{time\_a} * 100\% \leq 70\%$ ，视为在单列查询时使用了索引。若判断为没有使用索引，则测试点2和测试点3零分。
- judge\_whether\_use\_index\_on\_multiple\_attributes: 创建表并插入数据后，进行大量多列查询，记录耗时time\_a，创建多列索引，再次进行大量多列查询，记录耗时time\_b。若 $\text{time\_b} / \text{time\_a} * 100\% \leq 70\%$ ，视为在多列查询时使用了索引。若判断为没有使用索引，则测试点2和测试点3零分。



## 题目四 聚合函数与分组统计

### 测试点1：单独使用聚合函数

测试示例:

```
create table grade (course char(20),id int,score float);

insert into grade values('DataStructure',1,95);

insert into grade values('DataStructure',2,93.5);

insert into grade values('DataStructure',4,87);

insert into grade values('DataStructure',3,85);

insert into grade values('DB',1,94);

insert into grade values('DB',2,74.5);

insert into grade values('DB',4,83);

insert into grade values('DB',3,87);

select MAX(id) as max_id from grade;

select MIN(score) as min_score from grade where course = 'DB';

select COUNT(course) as course_num from grade;

select COUNT(*) as row_num from grade;

select SUM(score) as sum_score from grade where id = 1;

drop table grade;
```

期待输出:

```
| max_id |
| 4 |

| min_score |
| 74.500000 |

| course_num |
| 8 |

| row_num |
| 8 |
```

```
| sum_score |
```

```
| 189.000000 |
```

## 测试点2：聚合函数加分组统计

测试示例:

```
create table grade (course char(20),id int,score float);

insert into grade values('DataStructure',1,95);

insert into grade values('DataStructure',2,93.5);

insert into grade values('DataStructure',3,94.5);

insert into grade values('ComputerNetworks',1,99);

insert into grade values('ComputerNetworks',2,88.5);

insert into grade values('ComputerNetworks',3,92.5);

insert into grade values('C++',1,92);

insert into grade values('C++',2,89);

insert into grade values('C++',3,89.5);

select id,MAX(score) as max_score,MIN(score) as min_score,SUM(score) as sum_score
from grade group by id;

select id,MAX(score) as max_score from grade group by id having COUNT(*) > 3;

insert into grade values ('ParallelCompute',1,100);

select id,MAX(score) as max_score from grade group by id having COUNT(*) > 3;

select id,MAX(score) as max_score,MIN(score) as min_score from grade group by id
having COUNT(*) > 1 and MIN(score) > 88;

select course ,COUNT(*) as row_num , COUNT(id) as student_num , MAX(score) as
top_score, MIN(score) as lowest_score from grade group by course;

drop table grade;
```

期待输出:

```
| id | max_score | min_score | sum_score |
| 1 | 99.000000 | 92.000000 | 286.000000 |
| 2 | 93.500000 | 88.500000 | 271.000000 |
| 3 | 94.500000 | 89.500000 | 276.500000 |
```

```

| id | max_score |
| 1 | 100.000000 |

| id | max_score | min_score |
| 1 | 100.000000 | 92.000000 |

| 2 | 93.500000 | 88.500000 |

| 3 | 94.500000 | 89.500000 |

| course | row_num | student_num | top_score | lowest_score |
| DataStructure | 3 | 3 | 95.000000 | 93.500000 |
| ComputerNetworks | 3 | 3 | 99.000000 | 88.500000 |
| C++ | 3 | 3 | 92.000000 | 89.000000 |
| ParallelCompute | 1 | 1 | 100.000000 | 100.000000 |

```

### 测试点3：健壮性测试

测试示例:

```

create table grade (course char(20),id int,score float);

insert into grade values('DataStructure',1,95);

insert into grade values('DataStructure',2,93.5);

insert into grade values('DataStructure',3,94.5);

insert into grade values('ComputerNetworks',1,99);

insert into grade values('ComputerNetworks',2,88.5);

insert into grade values('ComputerNetworks',3,92.5);

-- SELECT 列表中不能出现没有在 GROUP BY 子句中的非聚集列

select id , score from grade group by course;

-- WHERE 子句中不能用聚集函数作为条件表达式

select id, MAX(score) as max_score where MAX(score) > 90 from grade group by id;

```

期待输出:

failure

failure

### 测试输出要求:

本题目的输出要求写入数据库文件夹下的 `output.txt` 文件中，例如测试数据库名称为 `execution_test_db`，则在测试时使用 `./bin/rmdb execution_test_db` 命令来启动服务端，对应输出应写入 `build/execution_test_db/output.txt` 文件中。

## 题目五 不相关子查询

### 测试点1：与子查询结果进行比较

测试示例：

```
create table grade (name char(20),id int,score float);

insert into grade values('tom',1,92);

insert into grade values('jack',2,89);

insert into grade values('mary',3,89.5);

select id from grade where score = (select MAX(score) from grade);

select id from grade where score < (select MAX(score) from grade);

select id from grade where score > (select MIN(score) from grade);
```

期待输出：

```
| id |
| 1 |

| id |
| 2 |

| 3 |

| id |
| 1 |

| 3 |
```

## 测试点2：带有IN谓词的子查询

测试示例：

```
create table grade (name char(20),id int,score float);

insert into grade values('tom',1,92);

insert into grade values('jack',2,89);

insert into grade values('mary',3,89.5);

select id from grade where name in (select name from grade);
```

期待输出：

```
| id |
| 1  |
| 2  |
| 3  |
```

## 测试点3：健壮性测试

测试示例：

```
create table grade (name char(20),id int,score float);

insert into grade values('tom',1,92);

insert into grade values('jack',2,89);

insert into grade values('mary',3,89.5);

select id from grade where score = (select score from grade);

select id from grade where name = (select MAX(score) from grade);
```

期待输出：

```
failure

failure
```

## 题目六 归并连接算法

本题包含三个测试点：

- sort\_merge\_join\_test\_example: 样例数据, 详细内容见后文
- sort\_merge\_join\_test\_small: 数据量较小, 参赛队伍可以认为数据可以全量放入内存
- sort\_merge\_join\_test\_large: 数据量较大, 参赛队伍可能需要考虑将中间结果写入外存

在每个测试点中, 我们首先会插入一定量的数据, 然后分别做三次连接操作

- 第一次为嵌套循环连接, 连接时间记为t1
- 第二次为未建立索引情况下的排序归并连接, 连接时间记为t2
- 第三次为建立索引下的排序归并连接, 连接时间为t3

参赛队伍在第二次和第三次排序归并连接时, 需要将中间结果(即有序中间表)输出到sorted\_results.txt中, 将连接结果输出到output.txt中

每个测试点通过要求:

- 连接结果正确
- 中间结果sorted\_results正确
- 第三次连接的时间t3不超过t1和t2的70%

以sort\_merge\_join\_test\_example为例

1. 首先, 我们建立需要连接的两张表

```
create table item (i_id int, i_im_id int, i_name char(24), i_price float, i_data char(50));
create table stock (s_i_id int, s_w_id int, s_quantity int, s_dist_01 char(24), s_dist_02 char(24), s_dist_03 char(24), s_dist_04 char(24), s_dist_05 char(24), s_dist_06 char(24), s_dist_07 char(24), s_dist_08 char(24), s_dist_09 char(24), s_dist_10 char(24), s_ytd float, s_order_cnt int, s_remote_cnt int, s_data char(50));
insert into item values (1, 6539, 'EPjQ', 140.125000, 'HIETK');
insert into item values (3, 7904, 'Byit', 507.312500, 'qzwfk');
insert into item values (2, 2088, '6BQf', 294.250000, 'FyuyM');
insert into stock values (3, 1, 35, '0ovK', 'pgGX', 'Z7JN', '6D2o', '77xx', 'kf0z', 'cuwy', 'cvac', 'J5v6', 'jBbI', 0.500000, 0, 0, 'JsfN4');
insert into stock values (1, 1, 37, 'ABk7', 'iUng', 'SnaO', 'LARv', 'l9yg', 'Fhpp', 'x6ha', 'Ulgc', 'wyjd', 'TUHV', 0.500000, 0, 0, 'rqHB0');
insert into stock values (2, 1, 72, '4jH3', 'PViF', 'KgLI', 'GnIU', 'Pfr7', 'GuZY', 'nPO2', 'aMAe', '6QfV', 'toID', 0.500000, 0, 0, '42Cin');
```

2. 然后分别做三次连接, 并记录时间和中间结果

```
# 嵌套循环连接
SET enable_nestloop = true;
SET enable_sortmerge = false;
select * from item, stock where s_i_id = i_id order by i_id;
# 未建立索引时的排序连接
SET enable_nestloop = false;
SET enable_sortmerge = true;
select * from item, stock where s_i_id = i_id order by i_id;
# 建立索引之后的排序连接
create index item(i_id);
create index stock(s_i_id);
select * from item, stock where s_i_id = i_id order by i_id;
```

3. 检查sorted\_results.txt和output.txt, 并判断是否t3是否超过t2和t1的70%

output.txt内容如下:

```
| i_id | i_im_id | i_name | i_price | i_data | s_i_id | s_w_id | s_quantity |  
s_dist_01 | s_dist_02 | s_dist_03 | s_dist_04 | s_dist_05 | s_dist_06 | s_dist_07 |  
s_dist_08 | s_dist_09 | s_dist_10 | s_ytd | s_order_cnt | s_remote_cnt | s_data |  
| 1 | 6539 | EPjQ | 140.125000 | HIETK | 1 | 1 | 37 | ABk7 | iUng | SNaO | LArv |  
19yg | Fhpp | x6ha | Ulgc | wyjd | TUHv | 0.500000 | 0 | 0 | rqHB0 |  
| 2 | 2088 | 6BQf | 294.250000 | FyuyM | 2 | 1 | 72 | 4jH3 | PViF | KgL1 | Gn1U |  
PfR7 | GuZY | nPO2 | aMAe | 6Qfv | tOiD | 0.500000 | 0 | 0 | 42CIn |  
| 3 | 7904 | Byit | 507.312500 | qzwfk | 3 | 1 | 35 | 0oVK | pgGX | Z7JN | 6D2o |  
77xx | kf0z | cuwy | cVac | J5v6 | jBbI | 0.500000 | 0 | 0 | Jsfn4 |
```

sorted\_results.txt内容如下:

```
| i_id | i_im_id | i_name | i_price | i_data |  
| 1 | 6539 | EPjQ | 140.125000 | HIETK |  
| 2 | 2088 | 6BQf | 294.250000 | FyuyM |  
| 3 | 7904 | Byit | 507.312500 | qzwfk |  
| s_i_id | s_w_id | s_quantity | s_dist_01 | s_dist_02 | s_dist_03 | s_dist_04 |  
s_dist_05 | s_dist_06 | s_dist_07 | s_dist_08 | s_dist_09 | s_dist_10 | s_ytd |  
s_order_cnt | s_remote_cnt | s_data |  
| 1 | 1 | 37 | ABk7 | iUng | SNaO | LArv | 19yg | Fhpp | x6ha | Ulgc | wyjd |  
TUHv | 0.500000 | 0 | 0 | rqHB0 |  
| 2 | 1 | 72 | 4jH3 | PViF | KgL1 | Gn1U | PfR7 | GuZY | nPO2 | aMAe | 6Qfv |  
tOiD | 0.500000 | 0 | 0 | 42CIn |  
| 3 | 1 | 35 | 0oVK | pgGX | Z7JN | 6D2o | 77xx | kf0z | cuwy | cVac | J5v6 |  
jBbI | 0.500000 | 0 | 0 | Jsfn4 |
```

## 题目七 事务控制语句

### 测试点

测试示例:

```
create table student (id int, name char(8), score float);  
insert into student values (1, 'xiaohong', 90.0);  
begin;  
insert into student values (2, 'xiaoming', 99.0);  
delete from student where id = 2;  
abort;  
select * from student;
```

期待输出:

```
| id | name | score |  
| 1 | xiaohong | 90.000000 |
```

## 测试文件说明

- commit\_test: 事务提交测试, 不包含索引
- abort\_test: 事务回滚测试, 不包含索引
- commit\_index\_test: 事务提交测试, 包含索引
- abort\_index\_test: 事务回滚测试, 包含索引

## 题目八 冲突可串行化

### 测试点: 判断系统是否会出现五种数据异常

测试示例:

```
-- 对脏读数据异常进行测试:
create table concurrency_test (id int, name char(8), score float);
insert into concurrency_test values (1, 'xiaohong', 90.0);
insert into concurrency_test values (2, 'xiaoming', 95.0);
insert into concurrency_test values (3, 'zhanghua', 88.5);

-- 事务1的测试语句:
t1a begin;
t1b update concurrency_test set score = 100.0 where id = 2;
t1c abort;
t1d select * from concurrency_test where id = 2;

--事务2的测试语句:
t2a begin;
t2b select * from concurrency_test where id = 2;
t2c commit;
```

期待操作序列:

```
t1a t2a t1b t2b t1c t1d
```

## 测试文件说明

- concurrency\_read\_test: 并发读测试
- dirty\_write\_test\_1: 脏写测试
- dirty\_write\_test\_2: 脏写测试
- dirty\_read\_test: 脏读测试
- lost\_update\_test: 丢失更新测试
- unrepeatable\_read\_test\_1: 不可重复读测试
- unrepeatable\_read\_test\_2: 不可重复读测试
- unrepeatable\_read\_test\_hard: 不可重复读测试
- phantom\_read\_test\_1: 幻读测试
- phantom\_read\_test\_2: 幻读测试



- phantom\_read\_test\_2\_wait: 幻读测试
- phantom\_read\_test\_3: 幻读测试
- phantom\_read\_test\_3\_wait: 幻读测试
- phantom\_read\_test\_4: 幻读测试

## 题目九 细粒度锁：间隙锁

### 测试点：判断系统是否实现细粒度锁以及是否出现幻读数据异常

测试示例：

```
-- 对脏读数据异常进行测试：
create table gap_lock_test (id int, name char(8), score float);
create index gap_lock_test (id);
insert into gap_lock_test values (1, 'xiaohong', 90.0);
insert into gap_lock_test values (2, 'xiaoming', 95.0);
insert into gap_lock_test values (4, 'zhanghua', 88.5);
insert into gap_lock_test values (7, 'xiaoyang', 91.0);
insert into gap_lock_test values (10, 'wangming', 92.0);
insert into gap_lock_test values (8, 'wanghong', 93.0);
insert into gap_lock_test values (100, 'zhaoming', 94.0);
insert into gap_lock_test values (201, 'zhaohong', 95.0);

-- 事务1的测试语句：
t1a begin;
t1b select * from gap_lock_test where id > 2 and id < 4;
t1c commit;
t1d select * from gap_lock_test where id > 4 and id < 20;

--事务2的测试语句：
txn2 3
t2a begin;
t2b insert into gap_lock_test values (11, 'zhaoyang', 99.0);
t2c commit;
```

期待操作序列：

```
t1a t2a t1b t2b t1c t2c t1d
```

### 测试文件说明

- 涉及Select单间隙、多间隙、间隙可能在最小的索引键之前，最大的索引键之后
- 不涉及update语句
- 同样需要考虑wait-die死锁预防策略

# 题目十 基于静态检查点的故障恢复-测试方案

## 测试点说明

本题目包含6个测试点，其中，前5个测试点是不包括静态检查点的故障恢复，第6个测试点是包含静态检查点的故障恢复：

| 测试点                               | 详细说明  |
|-----------------------------------|---|
| crash_recovery_single_thread_test | 单线程发送事务，数据量较小，不包括建立检查点  |
| crash_recovery_multi_thread_test  | 多线程发送事务，数据量较小，不包括建立检查点  |
| crash_recovery_index_test         | 单线程发送事务，包含建立索引，数据量较大，不包括建立检查点   |
| crash_recovery_large_data_test    | 多线程发送事务，数据量较大，不包括建立检查点  |
| crash_recovery_without_checkpoint | 单线程发送事务，数据量巨大，不包括建立检查点，会记录系统故障恢复时间t1（注：恢复时间指从server重启开始，到server能够提供服务为止所需的时间）   |
| crash_recovery_with_checkpoint    | 单线程发送事务，数据量同crash_recovery_without_checkpoint测试，在执行过程中，会不定时发送create static_checkpoint建立检查点，在系统crash之后会记录系统恢复时间t2，其中t2不超过t1的70%并且通过本题目的一致性检测，可认为通过本测试点 |

## 测试过程说明

2.1 创建表，并插入一定量的数据

```

create table warehouse (w_id int, w_name char(10), w_street_1 char(20),
w_street_2 char(20), w_city char(20), w_state char(2), w_zip char(9), w_tax
float, w_ytd float);
create table district (d_id int, d_w_id int, d_name char(10), d_street_1
char(20), d_street_2 char(20), d_city char(20), d_state char(2), d_zip char(9),
d_tax float, d_ytd float, d_next_o_id int);
create table customer (c_id int, c_d_id int, c_w_id int, c_first char(16),
c_middle char(2), c_last char(16), c_street_1 char(20), c_street_2 char(20),
c_city char(20), c_state char(2), c_zip char(9), c_phone char(16), c_since
char(30), c_credit char(2), c_credit_lim int, c_discount float, c_balance float,
c_ytd_payment float, c_payment_cnt int, c_delivery_cnt int, c_data char(50));
create table history (h_c_id int, h_c_d_id int, h_c_w_id int, h_d_id int, h_w_id
int, h_date char(19), h_amount float, h_data char(24));
create table new_orders (no_o_id int, no_d_id int, no_w_id int);
create table orders (o_id int, o_d_id int, o_w_id int, o_c_id int, o_entry_d
char(19), o_carrier_id int, o_ol_cnt int, o_all_local int);
create table order_line ( ol_o_id int, ol_d_id int, ol_w_id int, ol_number int,
ol_i_id int, ol_supply_w_id int, ol_delivery_d char(30), ol_quantity int,
ol_amount float, ol_dist_info char(24));
create table item (i_id int, i_im_id int, i_name char(24), i_price float, i_data
char(50));
create table stock (s_i_id int, s_w_id int, s_quantity int, s_dist_01 char(24),
s_dist_02 char(24), s_dist_03 char(24), s_dist_04 char(24), s_dist_05 char(24),
s_dist_06 char(24), s_dist_07 char(24), s_dist_08 char(24), s_dist_09 char(24),
s_dist_10 char(24), s_ytd float, s_order_cnt int, s_remote_cnt int, s_data
char(50));
insert ...
insert ...
insert ...

```

## 2.2 执行事务

```

begin;
select c_discount, c_last, c_credit, w_tax from customer, warehouse where w_id=1
and c_w_id=w_id and c_d_id=1 and c_id=2;
select d_next_o_id, d_tax from district where d_id=1 and d_w_id=1;
update district set d_next_o_id=5 where d_id=1 and d_w_id=1;
insert into orders values (4, 1, 1, 2, '2023-06-03 19:25:47', 26, 5, 1);
insert into new_orders values (4, 1, 1);
select i_price, i_name, i_data from item where i_id=10;
select s_quantity, s_data, s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05,
s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10 from stock where s_i_id=10
and s_w_id=1;
update stock set s_quantity=7 where s_i_id=10 and s_w_id=1;
insert into order_line values (4, 1, 1, 1, 10, 1, '2023-06-03 19:25:47', 7,
286.625000, 'VF2uQHlDhtxa5dkhPwwyCqgY');
select i_price, i_name, i_data from item where i_id=10;

```

测试点6会随机创建checkpoint

```
create static_checkpoint;
```

## 2.3 发生crash

```
crash
```

## 2.4 重启server

```
./bin/rmdb checkpoint_test
```

## 2.5 由另外一个线程一直尝试重连，统计恢复时间（仅最后两个测试点需要）

```
std::string query = "select * from district;";
time_t start_time = now();
while(true) {
    ret = connect_database("rmdb"); // 尝试重连
    if(ret == 0) {
        if(run_query(query)) { // 如果重连成功并且执行事务成功，视为系统已经恢复完成
            break;
        }
    }
    sleep(0.05); // 否则等待0.05s之后再次尝试
}

time_t end_time = now()
time_t recovery_time = end_time - start_time // 统计恢复时间
```

## 2.6 一致性检测（详见一致性检测文档），前五个测试点通过一致性测试即可拿到分数，最后一个测试点除了通过一致性测试之外，还需要故障恢复时间t2小于测试点5的故障恢复时间t1的70%