测试说明文档

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, 'asdfhjkl');
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, 'asdfhjkl');
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 = 'asdfhjkl';
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 | asdfhjkl |
failure
failure
| w_id | name |
| 10 | qqqqoooo |
| 10 | qweruiop |
| 500 | lastdanc |
| 507 | asdfhjkl |
```

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time_b / time_a * 100% <= 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time_b / time_a * 100% <= 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：选择运算下推

测试示例包含basic_query_test1和basic_query_test2:

```
create table students (stu_id int, stu_name char(20), class_id int, score
int);
create table classes (class_id int, class_name char(30), teacher
char(20));

insert into students values (1, 'anna',  100, 85);
insert into students values (2, 'ben',   200, 72);
insert into students values (3, 'carol', 100, 90);
insert into students values (4, 'david', 300, 95);

insert into classes values (100, 'math',    'smith');
insert into classes values (200, 'history', 'lee');
insert into classes values (300, 'physics', 'smith');

explain select * from students s join classes c on s.class_id = c.class_id
where s.score > 80 and c.teacher = 'smith';

drop table students;
drop table classes;
```

期待输出：

```
Project(columns=[*])
    Join(tables=[classes,students],condition=[s.class_id=c.class_id])
        Filter(condition=[c.teacher='smith'])
            Scan(table=classes)
        Filter(condition=[s.score>80])
            Scan(table=students)
```

测试点2：投影下推

测试示例包含basic_query_test3和basic_query_test4:

```
create table teams (team_id int, team_name char(20), city char(20));
create table players (player_id int, team_id int, player_name char(20),
points int);

insert into teams values (1, 'Rockets',  'Houston');
insert into teams values (2, 'Lakers',   'LA');

insert into players values (101, 1, 'john',   2300);
insert into players values (102, 1, 'mike',   1800);
insert into players values (103, 2, 'tony',   2100);

explain select t.team_name, p.player_name, p.points from teams t join
players p on t.team_id = p.team_id;
```

```
drop table teams;
drop table players;
```

期待输出：

```
Project(columns=[p.player_name,p.points,t.team_name])
    Join(tables=[players,teams],condition=[t.team_id=p.team_id])
        Project(columns=[t.team_id,t.team_name])
            Scan(table=teams)
        Project(columns=[p.player_name,p.points,p.team_id])
            Scan(table=players)
```

测试点3：连接顺序优化

测试示例包含basic_query_test5和basic_query_test6:

```
create table classes (class_id int, class_name char(30));
create table students (student_id int, class_id int, student_name
char(30));
create table grades (grade_id int, student_id int, subject char(30), score
int);

insert into classes values (1, 'Mathematics');
insert into classes values (2, 'History');
...
insert into classes values (9, 'Music');
insert into classes values (10, 'Computer Science');

insert into students values (1001, 1, 'Student_1');
insert into students values (1002, 1, 'Student_2');
...
insert into students values (1049, 10, 'Student_49');
insert into students values (1050, 10, 'Student_50');

insert into grades values (5001, 1001, 'Subject_A', 85);
insert into grades values (5002, 1001, 'Subject_B', 90);
...
insert into grades values (5149, 1050, 'Subject_B', 90);
insert into grades values (5150, 1050, 'Subject_C', 78);

explain select g.grade_id, s.student_name, c.class_name, g.subject,
g.score from grades g join students s on g.student_id = s.student_id join
classes c on s.class_id = c.class_id;

drop table classes;
drop table students;
drop table grades;
```

期待输出：

```
Project(columns=
[c.class_name,g.grade_id,g.score,g.subject,s.student_name])
    Join(tables=[classes,grades,students],condition=
[g.student_id=s.student_id])
        Join(tables=[classes,students],condition=[s.class_id=c.class_id])
            Project(columns=[c.class_id,c.class_name])
                Scan(table=classes)
            Project(columns=[s.class_id,s.student_id,s.student_name])
                Scan(table=students)
        Project(columns=[g.grade_id,g.score,g.student_id,g.subject])
            Scan(table=grades)
```

测试点4：稳健性测试

测试示例包含basic_query_test7和basic_query_test8:

```
create table authors (author_id int, author_name char(50), country
char(30));
create table books (book_id int, author_id int, title char(100), price
float);

insert into authors values (1, 'Leo Tolstoy', 'Russia');
insert into authors values (2, 'Ernest Hemingway', 'USA');
insert into authors values (3, 'Gabriel Garcia Marquez', 'Colombia');

insert into books values (101, 1, 'War and Peace', 14.99);
insert into books values (102, 1, 'Anna Karenina', 11.50);
insert into books values (201, 2, 'The Old Man and the Sea', 13.25);
insert into books values (202, 2, 'A Farewell to Arms', 9.75);
insert into books values (301, 3, 'One Hundred Years of Solitude', 15.00);
insert into books values (302, 3, 'Love in the Time of Cholera', 10.25);

explain select a.author_name, b.title from authors a join books b on
a.author_id = b.author_id where a.country = 'USA' and b.price > 10.000000;
drop table authors;
drop table books;
```

期待输出：

```
Project(columns=[a.author_name,b.title])
    Join(tables=[authors,books],condition=[a.author_id=b.author_id])
        Project(columns=[a.author_id,a.author_name])
            Filter(condition=[a.country='USA'])
                Scan(table=authors)
        Project(columns=[b.author_id,b.title])
```

```
            Filter(condition=[b.price>10.000000])
                Scan(table=books)
```

题目五 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
```

测试点4：order by 语句测试

测试示例:

```
create table records (vendor char(5), invoice_number int, amount float);

insert into records values('alpha', 1001, 98.0);
insert into records values('bravo', 2002, 76.5);
insert into records values('charl', 3003, 99.0);
insert into records values('delta', 1001, 98.5);
insert into records values('echoo', 4004, 88.25);
insert into records values('foxxx', 4004, 77.0);
insert into records values('golfy', 5005, 97.75);
insert into records values('hotel', 5005, 86.75);
insert into records values('indio', 6006, 76.25);
insert into records values('julie', 3003, 88.0);
insert into records values('karen', 5005, 89.25);
insert into records values('lenny', 2002, 91.125);
insert into records values('mango', 6006, 98.5);
insert into records values('nancy', 1001, 89.75);
insert into records values('oscar', 2002, 90.0);
insert into records values('peter', 3003, 95.0);
insert into records values('quack', 6006, 88.625);
insert into records values('romeo', 4004, 92.0);
insert into records values('sunny', 1001, 95.25);
insert into records values('tonny', 7007, 98.125);
insert into records values('ultra', 4004, 91.5);
insert into records values('vivid', 7007, 98.3125);

select * from records order by invoice_number, amount asc limit 2;
```

期待输出:

```
| vendor | invoice_number | amount |
| nancy | 1001 | 89.750000 |
| sunny | 1001 | 95.250000 |
```

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。

题目六 半连接 Semi Join

测试数据准备:

```
create table departments (dept_id int, dept_name char(20));

create table employees (emp_id int, emp_name char(20), dept_id int, salary
int);

insert into departments values(1, 'HR');

insert into departments values(2, 'Engineering');

insert into departments values(3, 'Sales');

insert into departments values(4, 'Marketing');

insert into employees values(101, 'Alice', 1, 70000);

insert into employees values(102, 'Bob', 2, 80000);

insert into employees values(103, 'Charlie', 2, 90000);

insert into employees values(104, 'David', 1, 75000);
```

测试点1：基本的 Semi Join (查询有员工的部门)

测试示例：

```
select dept_id, dept_name from departments SEMI JOIN employees ON
departments.dept_id = employees.dept_id;
```

期待输出：

```
| dept_id | dept_name |
| 1 | HR |
| 2 | Engineering |
```

(说明：HR和Engineering部门因为在employees表中有对应的员工记录而被选中。即使一个部门有多个员工，该部门也只在结果中出现一次。Sales和Marketing部门因为没有员工，所以不出现在结果中。)

测试点2：Semi Join 结果不受右表重复匹配影响 (仍然是查询有员工的部门)

测试示例：(此测试点与测试点1使用相同的查询，旨在强调即使右表（employees）中一个部门有多个员工，左表（departments）的对应行也只输出一次。)

```
select dept_id, dept_name from departments SEMI JOIN employees ON
departments.dept_id = employees.dept_id;
```

期待输出：

```
| dept_id | dept_name |
| 1 | HR |
| 2 | Engineering |
```

测试点3：Semi Join 右表为空或无匹配

测试示例：

```
create table projects (proj_id int, dept_id_assigned int);

select dept_name from departments SEMI JOIN projects ON
departments.dept_id = projects.dept_id_assigned;

insert into projects values(1001, 99); -- 插入一个与任何部门都无法匹配的项目

select dept_name from departments SEMI JOIN projects ON
departments.dept_id = projects.dept_id_assigned;
```

期待输出：

```
| dept_name |
| dept_name |
```

(说明：两次查询结果均为空，只显示表头，因为projects表初始为空，或其后的数据无法与departments表匹配。)

测试点4：健壮性测试 - 选择右表列

测试示例：

```
select dept_name, emp_name from departments SEMI JOIN employees ON
departments.dept_id = employees.dept_id;
```

期待输出：

```
failure
```

(说明：尝试选择右表 `employees` 的 `emp_name` 列，这在 `SEMI JOIN` 中是不允许的。)

测试点5：健壮性测试 - 左表为空

测试示例：

```
create table empty_departments (dept_id int, dept_name char(20));

select dept_name from empty_departments SEMI JOIN employees ON
empty_departments.dept_id = employees.dept_id;
```

期待输出：

```
| dept_name |
```

(说明：左表 `empty_departments` 为空，因此 Semi Join 结果也为空。)

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 多版本并发控制(MVCC)

测试点：MVCC的基础概念以及一些常规的异常处理

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

AbortTest:主动中断测试

Deadlock：死锁检测

DirtyReadTest：脏读测试

InsertDeleteTest：插入删除冲突处理

InsertTest：插入基础功能测试

Non_Repeatable_Read_Lost_Update：不可重复读情况下丢失更新

ReadWriteConflictDeleteTest：包含删除操作的读写冲突处理

ScanTest：Scan基础功能测试

TimestampTracking：时间戳基础功能测试

TupleReconstructTest：元组重构基础功能测试

UpdateTest：更新基础功能测试

WriteWriteConflictDeleteInsertTest：包含删除插入操作的写写冲突处理

WriteWriteConflictUpdateTest：包含更新操作的写写冲突处理

题目九 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%