第八周作业：数据库问答题

题目01- ReadView 案例

1. 数据准备

```sql
CREATE TABLE tab_user(
  `id` int(11)NOT NULL,
  `name` varchar (100) DEFAULT NULL,
  `age` int (11)NOT NULL,
  `address` varchar (255) DEFAULT NULL,
  PRIMARY KEY (id))
ENGINE=InnoDB;

Insert into tab_user(id , name , age , address) values (1,'刘备',18,'蜀国');

select * from tab_user ;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	刘备	18	蜀国

2. 案例01–读已提交RC隔离级别下的可见性分析

目标

在RC隔离级别下，Read View生成的时机对可见性的影响

操作步骤

- 事务1、事务2、事务3将事务隔离级别设置为RC
- 事务1、事务2、事务3分别开启事务
- 事务1更新字段name为关羽
- 事务1更新字段name为张飞
- 事务2更新字段name为赵云
- 事务2更新字段name为诸葛亮
- 事务3查询 id =1 的name数据
- 事务1提交事务
- 事务3查询 id =1 的name数据
- 事务2提交事务
- 事务3查询 id=1 的name数据

实践过程

- 事务1、事务2、事务3将事务隔离级别设置为RC

```sql
# 事务01
-- 查询事务隔离级别:
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

SELECT * FROM tab_user; # 默认是刘备

# Transaction 100
BEGIN;
  UPDATE tab_user SET name = '关羽' WHERE id = 1;
  UPDATE tab_user SET name = '张飞' WHERE id = 1;
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

@@tx_isolation
READ-COMMITTED

```sql
# 事务02
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

# Transaction 200
BEGIN;
  # 更新了一些别的表的记录
  ...
  UPDATE tab_user SET name = '赵云' WHERE id = 1;
  UPDATE tab_user SET name = '诸葛亮' WHERE id = 1;
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

@@tx_isolation
READ-COMMITTED

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

@@tx_isolation
READ-COMMITTED

- 事务1、事务2、事务3分别开启事务
- 事务1更新字段name为关羽
- 事务1更新字段name为张飞

```sql
# 事务01
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

SELECT * FROM tab_user; # 默认是刘备

# Transaction 100
BEGIN;
  UPDATE tab_user SET name = '关羽' WHERE id = 1;
  UPDATE tab_user SET name = '张飞' WHERE id = 1;
COMMIT;
```

信息 剖析 状态

```
UPDATE tab_user SET name = '关羽' WHERE id = 1
> Affected rows: 1
> 时间: 0s


UPDATE tab_user SET name = '张飞' WHERE id = 1
> Affected rows: 1
> 时间: 0s
```

- 事务2更新字段name为赵云（阻塞）

```sql
# 事务02
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

# Transaction 200
BEGIN;
  # 更新了一些别的表的记录
  ...
  UPDATE tab_user SET name = '赵云' WHERE id = 1;
  UPDATE tab_user SET name = '诸葛亮' WHERE id = 1;
COMMIT;
```

信息

```
UPDATE tab_user SET name = '赵云' WHERE id = 1
```

- 事务3查询 id =1 的name数据

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	刘备	18	蜀国

- 事务1提交事务

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	张飞	18	蜀国

- 事务2更新字段name为赵云
- 事务2更新字段name为诸葛亮

```sql
# 事务02
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

# Transaction 200
BEGIN;
  # 更新了一些别的表的记录
  ...
  UPDATE tab_user SET name = '赵云' WHERE id = 1;
  UPDATE tab_user SET name = '诸葛亮' WHERE id = 1;
COMMIT;
```

信息 | 剖析 | 状态

```
UPDATE tab_user SET name = '赵云' WHERE id = 1
> Affected rows: 1
> 时间: 0s


UPDATE tab_user SET name = '诸葛亮' WHERE id = 1
> Affected rows: 1
> 时间: 0s
```

- 事务3查询 id =1 的name数据

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	张飞	18	蜀国

- 事务2提交事务
- 事务3查询 id=1 的name数据

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level read committed;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	诸葛亮	18	蜀国

结论

在RC隔离级别下，事务在每次查询开始时都会生成一个独立的 Read View ，因此看到的都是已经提交的数据。

3. 案例02–可重复读RR隔离级别下的可见性分析

目标

在RR隔离级别下，Read View生成的时机对可见性的影响

操作步骤

- 事务1、事务2、事务3将事务隔离级别设置为RR
- 事务1、事务2、事务3分别开启事务
- 事务1更新字段name为关羽
- 事务1更新字段name为张飞
- 事务2更新字段name为赵云
- 事务2更新字段name为诸葛亮

- 事务3查询 id =1 的name数据
- 事务1提交事务
- 事务3查询 id =1 的name数据
- 事务2提交事务
- 事务3查询 id=1 的name数据

实践过程

- 事务1、事务2、事务3将事务隔离级别设置为RR

```sql
# 事务01
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

SELECT * FROM tab_user; # 默认是刘备

# Transaction 100
BEGIN;
  UPDATE tab_user SET name = '关羽' WHERE id = 1;
  UPDATE tab_user SET name = '张飞' WHERE id = 1;
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

@@tx_isolation
REPEATABLE-READ

```sql
# 事务02
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

# Transaction 200
BEGIN;
  # 更新了一些别的表的记录
  ...
  UPDATE tab_user SET name = '赵云' WHERE id = 1;
  UPDATE tab_user SET name = '诸葛亮' WHERE id = 1;
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

@@tx_isolation
REPEATABLE-READ

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

@@tx_isolation
REPEATABLE-READ

- 事务1、事务2、事务3分别开启事务
- 事务1更新字段name为关羽
- 事务1更新字段name为张飞

```sql
# 事务01
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

SELECT * FROM tab_user ; # 默认是刘备

# Transaction 100
BEGIN;
  UPDATE tab_user SET name = '关羽' WHERE id = 1;
  UPDATE tab_user SET name = '张飞' WHERE id = 1;
COMMIT;
```

信息 | 剖析 | 状态

```
JPDATE tab_user SET name = '关羽' WHERE id = 1
> Affected rows: 1
> 时间: 0s


JPDATE tab_user SET name = '张飞' WHERE id = 1
> Affected rows: 1
> 时间: 0s
```

- 事务3查询 id =1 的name数据

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	刘备	18	蜀国

- 事务1提交事务

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	刘备	18	蜀国

- 事务2更新字段name为赵云
- 事务2更新字段name为诸葛亮
- 事务3查询 id =1 的name数据

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	刘备	18	蜀国

- 事务2提交事务
- 事务3查询 id=1 的name数据

```sql
# 事务03
-- 查询事务隔离级别：
select @@tx_isolation;

-- 设置数据库的隔离级别
set session transaction isolation level repeatable  read;

BEGIN;
  # SELECT01：Transaction 100、200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'刘备'

  # SELECT02：Transaction 100提交，Transaction 200未提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'张飞'

  # SELECT03：Transaction 100、200提交
  SELECT * FROM tab_user WHERE id = 1; # 得到的列c的值为'诸葛亮'
COMMIT;
```

信息 | 结果 1 | 剖析 | 状态

id	name	age	address
1	刘备	18	蜀国

结论

在RR隔离级别下，只在第一次读取数据时生成一个Read View，后面有事务提交之后的值也是查询不到了，只有当该事务提交之后才会看到最新的值。

4. 结论分析

RC 和 RR 隔离级别的差异本质是因为MVCC中 Read View 的生成时机不同：

- 在RC隔离级别下，事务在每次查询开始时都会生成一个独立的 Read View
- 在RR隔离级别下，只在第一次读取数据时生成一个Read View

题目02–什么是索引?

1. 优点是什么?

- 降低数据库的IO成本
- 降低排序的成本
- 提高数据检索的效率

2. 缺点是什么?

- 占用更多磁盘空间
- 降低更新效率

3. 索引分类有哪些？特点是什么?

按照索引列的数量分类：

- 单列索引：索引中只有一个列。
    - 主键索引：索引列中的值必须是唯一的，不允许有空值
    - 普通索引：允许在定义索引的列中插入重复值和空值
    - 唯一索引：索引列中的值必须是唯一的，但是允许为空值
    - 全文索引：只能在文本类型CHAR，VARCHAR，TEXT类型字段上创建全文索引
    - 空间索引：地理位置空间索引

    - 前缀索引
- 组合索引：使用2个以上的字段创建的索引。

4. 索引创建的原则是什么?

（1）频繁出现在 where 条件字段，order 排序，group by 分组字段

（2）select 频繁查询的列，考虑是否需要创建联合索引

（3）多表 join 关联查询，on字段两边的字段都要创建索引

5. 有哪些使用注意实现?

（1）表记录很少不需要创建索引

（2）一个表的索引个数不能太多

（3）频繁更新的字段不建议作为索引

（4）区分度低的字段不建议建索引

（5）在InnoDB存储引擎中，主键索引建议使用自增的长整型

（6）不建议用无序的值作为索引

（7）尽量创建组合索引

6. 如何知道是否用到索引?

使用 explain 将语句的执行计划显示出来，通过执行计划可以看到哪张表使用了哪个索引。

7. 请你解释一下索引的原理是什么?

MySQL使用B+树构建索引，B+树和B树最主要的区别在于非叶子节点是否存储数据的问题。

- B树：非叶子节点和叶子节点都会存储数据。
- B+树：只有叶子节点才会存储数据，非叶子节点至存储键值。叶子节点之间使用双向指针连接，最底层的叶子节点形成了一个双向有序链表。

优点：

- 继承了B树的优点

- 保证等值和范围查询的快速查找
- MySQL的索引就采用了B+树的数据结构

题目03–什么是MVCC?

1. Undo log 日志

InnoDB 的多版本一致性读是采用了基于回滚段的的方式来实现的。对于更新和删除操作，InnoDB 并不是真正的删除原来的记录，而是设置记录的delete mark标识设置为1。Undo log 保存了记录修改前的镜像。

undo log 分为两种： **insert undo log** 和 **update undo log**

- **insert undo log**
  - 在 insert操作中产生的 undo log。因为 insert操作的记录只对事务本身可见，对于其它事务此记录是不可见的，所以可以在事务提交后直接删除而不需要进行回收操作。
- **update undo log**
  - 在 update 或 delete 操作中产生的 undo log。因为对已经存在的记录产生了影响，因此不能在事务提交时就进行删除，而是在事务提交时放到入 history list上，等待 purge 线程进行最后的删除操作。

2. ReadView

Read View会结合Undo log的默认字段来控制那个版本的 undo log 可被用户看见。

这个 Read View 中主要包含当前系统中还有哪些活跃的读写事务，把它们的事务id放到一个列表中。

列表中四个比较重要的概念：

- m_ids：表示在生成 read view 时，当前系统中活跃的读写事务id列表；
- m_low_limit_id：事务id下限，当前系统中活跃的读写事务中最小的事务id，m_ids 事务列表中的最小事务 id。
- m_up_limit_id：事务id上限，表示生成 read view 时，系统中应该分配给下一个事务的id值；
- m_creator_trx_id：表示生成该 read view 的事务的事务id；

3. 如何判断可见性

循环判断规则：

- 如果被访问版本的 trx_id 属性值，小于 Read View 中的事务下限id，表明生成该版本的事务在生成ReadView前已经提交，所以该版本可以被当前事务访问。
- 如果被访问版本的 trx_id 属性值，等于 Read View 中的 m_creator_trx_id，可以被访问。
- 如果被访问版本的 trx_id 属性值，大于等于 Read View 中的事务上限id，在生成 Read View 后才生成，所以该版本不可以被当前事务访问。
- 如果被访问版本的 trx_id 属性值，在 事务下限id 和 事务上限id 之间，那就需要判断是不是在 m_ids列表中。
    - 如果在，说明创建 Read View 时生成该版本的事务还是活跃的，该版本不可以被访问；
    - 如果不在，说明创建 Read View 时生成该版本的事务已经被提交，该版本可以被访问。

循环判断Undo log中的版本链中某个版本是否对当前事务可见，如果循环到最后一个版本也不可见的话，那么就意味着该条记录对当前事务不可见，查询结果就不包含该记录。