狂神说MySQL06：事务和索引

秦疆 狂神说 2020-05-03

狂神说MySQL系列连载课程，通俗易懂，基于MySQL5.7.19版本，欢迎各位狂粉转发关注学习。禁止随意转载，转载记住贴出B站视频链接及公众号链接！

上课视频同步文档

事务和索引

事务

> 什么是事务

- 事务就是将一组SQL语句放在同一批次内去执行
- 如果一个SQL语句出错,则该批次内的所有SQL都将被取消执行
- MySQL事务处理只支持InnoDB和BDB数据表类型

> 事务的ACID原则 百度 ACID

原子性(Atomic)

- 整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（ROLLBACK）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性(Consist)

- 一个事务可以封装状态改变（除非它是一个只读的）。事务必须始终保持系统处于一致的状态，不管在任何给定的时间并发事务有多少。也就是说：如果事务是并发多个，系统也必须如同串行事务一样操作。其主要特征是保护性和不变性(Preserving an Invariant)，以转账案例为例，假设有五个账户，每个账户余额是100元，那么五个账户总额是500元，如果在这个5个账户之间同时发生多个转账，无论并发多少个，比如在A与B账户之间转账5元，在C与D账户之间转账10元，在B与E之间转账15元，五个账户总额也应该还是500元，这就是保护性和不变性。

隔离性(Isolated)

- 隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请求，使得在同一时间仅有一个请求用于同一数据。

持久性(Durable)

- 在事务完成以后，该事务对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

基本语法

```
-- 使用set语句来改变自动提交模式
SET autocommit = 0;   /*关闭*/
SET autocommit = 1;   /*开启*/

-- 注意:
--- 1.MySQL中默认是自动提交
--- 2.使用事务时应先关闭自动提交

-- 开始一个事务,标记事务的起始点
START TRANSACTION

-- 提交一个事务给数据库
COMMIT

-- 将事务回滚,数据回到本次事务的初始状态
ROLLBACK

-- 还原MySQL数据库的自动提交
SET autocommit =1;

-- 保存点
SAVEPOINT 保存点名称 -- 设置一个事务保存点
ROLLBACK TO SAVEPOINT 保存点名称 -- 回滚到保存点
RELEASE SAVEPOINT 保存点名称 -- 删除保存点
```

测试

```
/*
课堂测试题目

A在线买一款价格为500元商品,网上银行转账.
A的银行卡余额为2000,然后给商家B支付500.
商家B一开始的银行卡余额为10000

创建数据库shop和创建表account并插入2条数据
*/

CREATE DATABASE `shop`CHARACTER SET utf8 COLLATE utf8_general_ci;
```

```
USE `shop`;

CREATE TABLE `account` (
`id` INT(11) NOT NULL AUTO_INCREMENT,
`name` VARCHAR(32) NOT NULL,
`cash` DECIMAL(9,2) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8

INSERT INTO account (`name`,`cash`)
VALUES('A',2000.00),('B',10000.00)

-- 转账实现
SET autocommit = 0; -- 关闭自动提交
START TRANSACTION;  -- 开始一个事务,标记事务的起始点
UPDATE account SET cash=cash-500 WHERE `name`='A';
UPDATE account SET cash=cash+500 WHERE `name`='B';
COMMIT; -- 提交事务
# rollback;
SET autocommit = 1; -- 恢复自动提交
```

索引

> 索引的作用

- 提高查询速度
- 确保数据的唯一性
- 可以加速表和表之间的连接 , 实现表与表之间的参照完整性
- 使用分组和排序子句进行数据检索时 , 可以显著减少分组和排序的时间
- 全文检索字段进行搜索优化.

> 分类

- 主键索引 (Primary Key)
- 唯一索引 (Unique)
- 常规索引 (Index)
- 全文索引 (FullText)

> 主键索引

主键 : 某一个属性组能唯一标识一条记录

特点 :

- 最常见的索引类型
- 确保数据记录的唯一性
- 确定特定数据记录在数据库中的位置

> 唯一索引

作用：避免同一个表中某数据列中的值重复

与主键索引的区别

- 主键索引只能有一个
- 唯一索引可能有多个

```
CREATE TABLE `Grade`(
  `GradeID` INT(11) AUTO_INCREMENT PRIMARYKEY,
  `GradeName` VARCHAR(32) NOT NULL UNIQUE
   -- 或 UNIQUE KEY `GradeID` (`GradeID`)
)
```

> 常规索引

作用：快速定位特定数据

注意：

- index 和 key 关键字都可以设置常规索引
- 应加在查询找条件的字段
- 不宜添加太多常规索引,影响数据的插入,删除和修改操作

```
CREATE TABLE `result`(
   -- 省略一些代码
  INDEX/KEY `ind` (`studentNo`,`subjectNo`) -- 创建表时添加
)
```

```
-- 创建后添加
ALTER TABLE `result` ADD INDEX `ind`(`studentNo`,`subjectNo`);
```

> 全文索引

百度搜索：全文索引

作用：快速定位特定数据

注意：

- 只能用于MyISAM类型的数据表
- 只能用于CHAR , VARCHAR , TEXT数据列类型
- 适合大型数据集

```
/*
#方法一：创建表时
      CREATE TABLE 表名 （
                字段名1 数据类型 [完整性约束条件…],
                字段名2 数据类型 [完整性约束条件…],
                [UNIQUE | FULLTEXT | SPATIAL ]   INDEX | KEY
                [索引名] (字段名[(长度)] [ASC |DESC])
                ）;


#方法二：CREATE在已存在的表上创建索引
       CREATE [UNIQUE | FULLTEXT | SPATIAL ] INDEX 索引名
                    ON 表名 (字段名[(长度)] [ASC |DESC]) ;


#方法三：ALTER TABLE在已存在的表上创建索引
       ALTER TABLE 表名 ADD [UNIQUE | FULLTEXT | SPATIAL ] INDEX
                              索引名 (字段名[(长度)] [ASC |DESC]) ;


#删除索引：DROP INDEX 索引名 ON 表名字;
#删除主键索引：ALTER TABLE 表名 DROP PRIMARY KEY;


#显示索引信息：SHOW INDEX FROM student;
*/

/*增加全文索引*/
ALTER TABLE `school`.`student` ADD FULLTEXT INDEX `studentname` (`StudentName`);

/*EXPLAIN : 分析SQL语句执行性能*/
EXPLAIN SELECT * FROM student WHERE studentno='1000';

/*使用全文索引*/
-- 全文搜索通过 MATCH() 函数完成。
-- 搜索字符串作为 against() 的参数被给定。搜索以忽略字母大小写的方式执行。对于表中的每个记录行，MATCH()返回一个相关性值。即，在搜索字符串与记录行在 MATCH() 列表中指定的列的文本之间的相似性尺度。
EXPLAIN SELECT *FROM student WHERE MATCH(studentname) AGAINST('love');

/*
开始之前，先说一下全文索引的版本、存储引擎、数据类型的支持情况

MySQL 5.6 以前的版本，只有 MyISAM 存储引擎支持全文索引；
MySQL 5.6 及以后的版本，MyISAM 和 InnoDB 存储引擎均支持全文索引；
只有字段的数据类型为 char、varchar、text 及其系列才可以建全文索引。
测试或使用全文索引时，要先看一下自己的 MySQL 版本、存储引擎和数据类型是否支持全文索引。
*/
```

> 拓展：测试索引

建表app_user：

```
CREATE TABLE `app_user` (
`id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(50) DEFAULT '' COMMENT '用户昵称',
`email` varchar(50) NOT NULL COMMENT '用户邮箱',
`phone` varchar(20) DEFAULT '' COMMENT '手机号',
`gender` tinyint(4) unsigned DEFAULT '0' COMMENT '性别（0:男；1：女）',
`password` varchar(100) NOT NULL COMMENT '密码',
`age` tinyint(4) DEFAULT '0' COMMENT '年龄',
`create_time` datetime DEFAULT CURRENT_TIMESTAMP,
`update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='app用户表'
```

批量插入数据：100w

```
DROP FUNCTION IF EXISTS mock_data;
DELIMITER $$
CREATE FUNCTION mock_data()
RETURNS INT
BEGIN
DECLARE num INT DEFAULT 1000000;
DECLARE i INT DEFAULT 0;
WHILE i < num DO
  INSERT INTO app_user(`name`, `email`, `phone`, `gender`, `password`, `age`)
   VALUES(CONCAT('用户', i), '24736743@qq.com', CONCAT('18', FLOOR(RAND()*(999999999-
100000000)+100000000)),FLOOR(RAND()*2),UUID(), FLOOR(RAND()*100));
  SET i = i + 1;
END WHILE;
RETURN i;
END;
SELECT mock_data();
```

索引效率测试

无索引

```
SELECT * FROM app_user WHERE name = '用户9999'; -- 查看耗时
SELECT * FROM app_user WHERE name = '用户9999';
SELECT * FROM app_user WHERE name = '用户9999';

mysql> EXPLAIN SELECT * FROM app_user WHERE name = '用户9999'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: app_user
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 992759
```

```
    filtered: 10.00
      Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

创建索引

```
CREATE INDEX idx_app_user_name ON app_user(name);
```

测试普通索引

```
mysql> EXPLAIN SELECT * FROM app_user WHERE name = '用户9999'\G
*************************** 1. row ***************************
          id: 1
select_type: SIMPLE
       table: app_user
  partitions: NULL
        type: ref
possible_keys: idx_app_user_name
         key: idx_app_user_name
     key_len: 203
         ref: const
        rows: 1
    filtered: 100.00
      Extra: NULL
1 row in set, 1 warning (0.00 sec)

mysql> SELECT * FROM app_user WHERE name = '用户9999';
1 row in set (0.00 sec)

mysql> SELECT * FROM app_user WHERE name = '用户9999';
1 row in set (0.00 sec)

mysql> SELECT * FROM app_user WHERE name = '用户9999';
1 row in set (0.00 sec)
```

> 索引准则

- 索引不是越多越好
- 不要对经常变动的数据加索引
- 小数据量的表建议不要加索引
- 索引一般应加在查找条件的字段

> 索引的数据结构

```
-- 我们可以在创建上述索引的时候，为其指定索引类型，分两类
hash类型的索引：查询单条快，范围查询慢
btree类型的索引：b+树，层数越多，数据量指数级增长（我们就用它，因为innodb默认支持它）

-- 不同的存储引擎支持的索引类型也不一样
```

```
InnoDB 支持事务，支持行级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；
MyISAM 不支持事务，支持表级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；
Memory 不支持事务，支持表级别锁定，支持 B-tree、Hash 等索引，不支持 Full-text 索引；
NDB 支持事务，支持行级别锁定，支持 Hash 索引，不支持 B-tree、Full-text 等索引；
Archive 不支持事务，支持表级别锁定，不支持 B-tree、Hash、Full-text 等索引；
```

视频同步更新

如果觉得帮助到了您，不妨赞赏支持一下吧！

“赠人玫瑰，手有余香”

狂神说 的赞赏码

狂神说