MySQL进阶

视图

> 作用：将select的查询结果存储成一张表，方便后续直接查看

```
CREATE VIEW test AS select * FROM student WHERE sex = '男';

UPDATE test SET `name` = '小明' WHERE sid = 1;

DROP VIEW test;

# 加上WITH CHECK OPTION之后，如果对里面的数据进行修改，那么就会检测修改后的数据是否还满足这个视图的条件，不满足就不允许更改
CREATE VIEW test AS select * FROM student WHERE sex = '男' WITH CHECK OPTION;
```

> 在一些情况下，试图是不允许更新的
>
> 1. 若视图是由两个以上基本表导出的，则此视图不允许更新
> 2. 若视图的字段来自字段表达式或常数，则不允许对此视图执行INSERT和UPDATE操作，但允许执行DELETE
> 3. 若视图字段来自聚集函数AVG、SUM等，则此视图不允许更新
> 4. 若视图定义中含有GROUP BY HAVING子句，则此视图不允许更新
> 5. 若视图中含有DISTINCT语句，则此视图不允许更新
> 6. 若视图定义中含有嵌套查询，并且内层查询的FROM子句涉及的表也是导出该视图的基本表，则此视图不允许更新
> 7. 一个不允许更新的视图上定义的视图也不允许更新

索引

> 作用：提高查询效率，如同哈希表，能够快速定位到元素存放的位置，带来速度的同时也会占用磁盘资源

```
CREATE INDEX i ON student(`name`);

DROP INDEX i ON student;

# 唯一索引
CREATE UNIQUE INDEX o on student(`name`);
```

> 需要注意的是，索引带来快速的同时也会消耗比较大的资源，所以索引不是创建的越多越好，而是要根据实际情况，能不用就不用

触发器

> 作用： 在执行某种操作的时候，自动触发执行预先设置好的内容，通常用于检查内容的安全性，相比于设置约束更加灵活
>
> 可以理解为自动任务

> 触发器依赖的基本表成为触发器表，当执行select、update、delete时，会自动生成两张表，分别为new表和old表，
>
> 这两张表只能由触发器操作
>
> 例如当执行delete操作后，触发器可以从old表中获取到已删除的内容

```
# 创建触发器
CREATE TRIGGER t BEFORE DELETE ON student FOR EACH ROW DELETE FROM teach WHERE
old.sid = teach.sid;
# 展示触发器
SHOW TRIGGERS;
# 删除触发器
DROP TRIGGER t;
```

事务

> 当操作设计多张表，多个数据的时候，需要设计多个sql语句去完成这些操作，那么这些sql语句就可以构成一个事务，支持事务的引擎只有InnoDB,可以通过下面的语句查看所有引擎

```
SHOW ENGINES;
```

事务的四大特性：

1. 原子性：事务中的sql语句要么全部完成，要么全部不完成，中间如果有一条sql语句执行出错，那么这个事务就会回滚到最开始的地方，相当与这个事务没有执行过一样
2. 一致性：事务执行完成后，数据库的完整性没有被破坏，意思就是事务中的sql语句要符合数据库的预设规则，不能随意篡改
3. 隔离性：数据库允许多个事务同时对数据进行读写和修改的能力，这里的事务隔离可以解决并发事务所带来的一系列问题，事务隔离又分为四个级别：读未提交、读提交、可重复度、串行化
4. 持久性：事务处理结束后，就不能再进行回滚

```
# 开启事务
BEGIN;
# 删除一条数据，执行完后真实的数据库中还能看到这条数据，此条语句在bak表中执行
DELETE FROM student WHERE sid = 2;
# 提交事务，此时才能在真实的数据库中看到上一条sql的执行结果
COMMIT;

# 开启事务
BEGIN;
# 删除一条数据
```

```
DELETE FROM student WHERE sid = 456;
# 此时保存一个还原点，可以理解为存档
SAVEPOINT tttt;
# 再删除一条数据
DELETE FROM student WHERE sid = 789;
# 此时查看一下bak表中的数据，发现两条数据顺利被删除
SELECT * FROM student;
# 然后还原到之前的存档点
ROLLBACK TO tttt;
# 发现一条数据已经回来了
SELECT * FROM student;
# 接着全部还原
ROLLBACK;
# 发现删除的数据都已经恢复
SELECT * FROM student;
# 最后提交事务
COMMIT;
```

MySQL高级

字符串函数

- substring(字段，起始位置，结束位置)
- left(字段，长度) 从左边开始切割
- right(字段，长度) 从右边开始切割

```
# 注意和Java不一样的是，这里的起始位置从1开始，不是从0开始
SELECT SUBSTRING(`name`, 2,2) FROM student;

SELECT LEFT(`name`, 2) FROM student;

SELECT RIGHT(`name`, 1) FROM student;
```

> 不只是字段，也可以将字段换成字符串进行切割

```
SELECT SUBSTRING('123456',2,3);
```

- upper（将查询出来的内容中的小写字母全部替换为大写字母）
- lower（将查询出来的内容中的大写字母全部替换为小写字母）

```
SELECT UPPER('hello world');

SELECT LOWER('hello world');
```

- replace(字段，原文，替换文)

```
SELECT REPLACE(`name`,'小','大') FROM student;
```

> 需要注意的是，以上函数都是对结果进行修改，不是对数据库原文进行修改

- concat(字符串1，字符串2) 拼接字符串

```
SELECT CONCAT(`name`,'字') FROM student;
```

- length 获取字段的长度，注意如果使用的utf8编码，那么一个汉字占三个字节

```
SELECT LENGTH(`name`) FROM student;
```

日期函数

- date_add(日期，interval增量，单位)

```
# 向后一年
SELECT DATE_ADD('2020-3-2',INTERVAL 1 YEAR);

# 向前五天
SELECT DATE_ADD('2020-3-2',INTERVAL -5 DAY);
```

- curdate() 获取当前日期
- curtime() 获取当前时间
- now() 获取当前日期和时间

```
SELECT CURDATE();
SELECT CURTIME();
SELECT NOW();
```

- day(日期)
- month(日期)
- year(日期)

> 以上三个函数分别获取某个日期的年月日

```
SELECT DAY(NOW());
SELECT MONTH(NOW());
SELECT YEAR(NOW());
```

数学函数

函数	作用
abs(x)	求绝对值
PI()	返回圆周率
sqrt(x)	x的平方根
mod(x,y)	x除以y的余数
ceil()、ceiling()	返回大于或者等于x的最小整数
floor(x)	返回小于或者等于x的最小整数
rand() rand(x)	返回0~1的随机数，x相同时返回值相同
round(x) round(x,y)	返回整数，四舍五入。返回x保留到小数点后y位的值
truncate(x,y)	truncate(x,y) 截断返回x保留到小数点后y位的值（不四舍五入）
sign(x)	返回x的符号，负数、0、正数分别返回-1、0、1
pow(x,y) power(x,y)	pow(x,y) power(x,y) 返回x的y次方
exp(x)	返回e的x次方
log(x)	返回x的自然对数
log10(x)	返回以10为底的对数
radians(x)	将角度转换为弧度
degrees(x)	将弧度转换为角度
sin(x)	正弦函数，返回正弦值，x是弧度
asin(x)	反正弦函数，返回反正弦值，x是弧度
cos(x)	余弦函数，返回余弦值，x是弧度
acos(x)	反余弦函数，返回反余弦值，x是弧度
tan(x)	正切函数，返回正切值，x是弧度
atan(x) atan2(x)	反正切函数，返回反正切值，x是弧度
cot(x)	余切函数，返回余切值，tan(x)的倒数

类型转换函数

```
SELECT 1 + '2'; # 3

SELECT CONCAT(1,'2'); # 12
```

- cast(数据， as 数据类型)

```
SELECT CAST(PI() as UNSIGNED);

SELECT CAST('123abcd123' AS UNSIGNED); # 结果为123 能转换到哪就到哪
```

流程控制函数

- if(条件表达式,结果1,结果2)
- ifnull(a， b) 如果a的值为null，则返回b，反之则返回a
- nullif(a,b) 如果a和b相等，那么返回null
- case...when...

```
SELECT CASE value # 判断value的值，当value等于1时返回0，反之则返回1，类似于if else结构
    WHEN 1 THEN
        0
    ELSE
        1
END;
```

sleep(sec)

休眠函数，类似于Thread.sleep，以秒为单位

自定义函数

> 优点：可以将一些逻辑封装起来保存，想要再次实现只需要调用自定义函数即可
>
> 缺点：自定义函数不能修改，如果出现错误只能删除然后重新编写

```
CREATE FUNCTION test() RETURN INT
BEGIN
RETURN 666;
END
```

> 以上是标准的函数定义流程，但是存在一个问题，分号一般用于语句结束，也就是说这个定义函数的语句执行到return 666的时候就会强行结束，并不会往下执行
>
> 所以这里需要先修改一下结束符号，让其不为分号，待函数定义完毕后改回分号即可

```
delimiter $$
CREATE FUNCTION test() RETURN INT
BEGIN
```

```
RETURN 666;
END $$
delimiter ;
```

> 这种代码方式就非常繁琐，所以一般都用navicat去编写

```
# 定义一个带参数的函数，返回参数的平方
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...

    RETURN i * i;
END
```

```
# 定义一个带参数的函数，在函数内部定义一个局部变量并为其赋值，返回参数和变量的乘积
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...
    DECLARE a INT;
    SET a = 10;
    RETURN i * a;
END
# 给变量一个默认值
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...
    DECLARE a INT DEFAULT 20;
    RETURN i * a;
END
```

> 在函数内部使用sql语句

```
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`() RETURNS int(11)
BEGIN
    #Routine body goes here...
    DECLARE a INT DEFAULT 20;
    SELECT COUNT(*) INTO a FROM student;
    RETURN a;
END
```

> 流程控制语句

```
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...
```

```
    IF(i < 10) THEN
        RETURN i;
    ELSE
        RETURN i - 10;
    END IF;
END
# 使用exists 判断查询结果是否存在
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...
    IF NOT EXISTS(SELECT * FROM student WHERE sid=100) THEN
        RETURN i;
    ELSE
        RETURN i - 10;
    END IF;
END
```

```
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...
    CASE i
    WHEN i < 10 THEN
        RETURN i;
    ELSE
        RETURN i-10;
END CASE;

END
```

```
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
    #Routine body goes here...
    DECLARE a INT DEFAULT 10;
    lp1: LOOP
        lp2: LOOP
            SET a = a - 1;
            IF a < 5 THEN
                LEAVE lp1;
            END IF;
        END LOOP lp2;
    END LOOP lp1;
    RETURN a;
END
```

```
# repeat循环，类似于Java中的do while循环结构，会先去执行循环体中的内容，再进行判断
CREATE DEFINER=`root`@`localhost` FUNCTION `test1`(i INT) RETURNS int(11)
BEGIN
```

```
    #Routine body goes here...
    DECLARE a INT DEFAULT 10;
    REPEAT
    SET a = a-1;
UNTIL a < 5 END REPEAT;

    RETURN a;
END
```

全局变量

> 全局变量可以用于一次会话内部，不局限于函数内部，整个会话都可以进行访问

```
SET @x = 456;

SELECT @x;

SELECT * FROM student WHERE sid = @x;
```

锁机制

> 作用：在多线程同时操作表中数据的时候，需要避免潜在的并发问题，如一个线程对此表数据进行删除，另一个线程对此表数据进行查询，那么这个时候就会出现并发问题，事务的隔离级别就可以处理此类问题，那么在事务隔离中就使用了锁机制

- 读未提交（RU）：能够读到其它事务中未提交的内容，存在脏读问题
- 读已提交（RC）：只能读到其他事务已提交的内容，存在不可重复读问题
    - 这里解释一下，只能够读取到其他事务已提交的内容，也就是说如果有一个线程在你读取的瞬间进行了事务提交，那么你就读取到了后面事务提交后的内容，并没有读取到你想要的内容，所以就是说可能两次读取的结果不一致，也就是不可重复读（一般会把一套查询编写成事务，也就是说在你的事务里面的两次查询查询出来的结果不一致，这也就是不可重复读问题）
- 可重复读（RR）：在你编写事务的时候，读取某行后，不允许其它事务操作此行数据，直至你编写的事务结束，这样就解决了不可重复读问题，但是会带来新的问题 -- **幻读**
    - 幻读：RR只是限制了你的update操作，但是不限制你的insert操作，也就是说我可以对此表插入一行数据，那么你两次查询的时候可能也会出现多一行数据或者少一行数据的问题，仿佛出现了幻觉，也就是**幻读**
- 串行读（Serializable）：不存在同时事务的问题，一个事务必须等待另一个事务的完成（非常简单暴力）

切换事务隔离级别

```
# 将事务自动提交关闭
set autocommit = false;

# 切换事务隔离级别为读未提交
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
# 此时开启两个mysql窗口或者在navicat中新建两个查询
# 第一个
SELECT * from student;
# 第二个
UPDATE student SET sex = '女' WHERE sid = 456

# 此时在第一个查询再次查询
SELECT * from student; # 可以发现查询出来的数据是已经被修改过的数据，但是我们关闭了自动提交事务

# 在第二个查询回滚
ROLLBACK

# 再次在第一个查询进行查询
SELECT * from student; # 可以发现数据再次变化
```

读已提交demo

```
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
Query OK, 0 rows affected (0.00 sec)

mysql> set autocommit = false;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from student;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
|      4 | 小亮   | 女  |
|    456 | 李四   | 男  |
|    789 | 李浩   | 女  |
| 123457 | 小红   | 女  |
+--------+--------+-----+
4 rows in set (0.00 sec)

mysql> select * from student;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
|      4 | 小亮   | 女  |
|    456 | 李四   | 男  |
|    789 | 李浩   | 女  |
| 123457 | 小红   | 女  |
+--------+--------+-----+
4 rows in set (0.00 sec)
```

```
mysql> use mysql_study;
Database changed
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> set autocommit =false;
Query OK, 0 rows affected (0.00 sec)

mysql> update student set sex = '男' where sid=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

可重复读（默认隔离级别）

```
mysql> set autocommit = false;
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from student;
ERROR 1046 (3D000): No database selected
mysql> use mysql_study;
Database changed
mysql> select * from student;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
|      4 | 小亮   | 女  |
|    456 | 李四   | 男  |
|    789 | 李浩   | 女  |
| 123457 | 小红   | 女  |
+--------+--------+-----+
4 rows in set (0.00 sec)

mysql> select * from student;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
|      4 | 小亮   | 女  |
|    456 | 李四   | 男  |
|    789 | 李浩   | 女  |
| 123457 | 小红   | 女  |
+--------+--------+-----+
4 rows in set (0.00 sec)

mysql> select * from student;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
|      4 | 小亮   | 女  |
|    456 | 李四   | 男  |
|    789 | 李浩   | 女  |
```

```
| 123457 | 小红   | 女  |
+--------+--------+-----+
4 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from student;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
|      4 | 小亮   | 男  |
|    456 | 李四   | 男  |
|    789 | 李浩   | 女  |
| 123457 | 小红   | 女  |
+--------+--------+-----+
4 rows in set (0.00 sec)
```

```
mysql> set autocommit = false;
Query OK, 0 rows affected (0.00 sec)

mysql> begin
    -> ;
Query OK, 0 rows affected (0.00 sec)

mysql> use mysql_study;
Database changed
mysql> update student set sex = '男' where sid=4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

> 这里可以看到MySQL默认的事务隔离级别就是在开启一个事务的时候，就相当于对你操作的表进行了一个锁定，其它事务就算修改表中数据，在你正在进行的事务中数据都不会进行改变，只有当你的事务提交后，其它事务做出的改变你才能看得到

> 关于幻读的问题，MySQL并没有真正解决幻读，在你的事务中查询不出新插入的数据只是看不到而已，当你对整个表进行修改的时候，还是会影响到新插入的数据

在RR级别下，MySQL在一定程度上解决了幻读的问题：

- 在快照读（不加锁）情况下，MySQL通过mvcc避免幻读
- 在当前读（加锁）情况下，MySQL通过next-key避免幻读

读锁和写锁

按照操作类型来区分，锁分为读锁和写锁

- 读锁：也叫共享锁，就是说当你的事务添加了读锁后，其它事务也可以添加读锁和读取表中数据，但是不能对数据进行写入操作，只能等到所有读锁全部释放
- 写锁：也叫排他锁，就是说当你的事务添加了写锁后，其它事务就不能读也不能写也不能添加锁，只能等待当前事务释放锁

全局锁、表锁和行锁

从锁的作用范围来说：

- 全局锁：锁作用于全局，整个数据库的所有操作都会受到限制
- 表锁：锁作用于整个表，所有对表的操作都会受到限制
- 行锁：锁作用域表中的某一行，只会通过锁限制某一行的操作（仅InnoDB支持）

```
# 开启全局读锁
mysql> flush tables with read lock;
Query OK, 0 rows affected (0.01 sec)
# 释放
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
# 给student表添加表锁
mysql> lock table student read;
Query OK, 0 rows affected (0.00 sec)
# 释放
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
# 添加写锁
mysql> lock table student write;

mysql> set autocommit = false;
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)
# 添加行锁
mysql> select * from student where sid = 123456 lock in share mode;
+--------+--------+-----+
| sid    | name   | sex |
+--------+--------+-----+
| 123456 | 小刚   | 男  |
+--------+--------+-----+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

记录一下关于添加排他锁发生的一场：一次MySQL异常排查：Query execution was interrupted - 码上快乐 (codeprj.com)

下面尝试不使用id进行选择，使用name进行选择

```
mysql> select * from student where name = '李四' lock in share mode;
+-----+--------+-----+
| sid | name   | sex |
+-----+--------+-----+
| 456 | 李四   | 男  |
+-----+--------+-----+
1 row in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

> 中途会出现问题，当对其它行进行修改的时候，也无法修改
>
> 这里就是锁进行了升级，因为name字段并没有添加索引，所以它默认给所有行都加上了锁

记录锁、间隙锁和临键锁

- 记录锁：当使用唯一索引进行精确匹配并且表中存在记录的时候，这个时候这一行就有记录锁，仅仅锁住这一行
- 间隙锁：当使用唯一索引进行精确匹配并且表中不存在记录的时候，这个时候就上了间隙锁，间隙锁锁的是一个区间，例如id自增到10，匹配了id为200的数据，那么这个时候不存在200这条数据，间隙锁就给11-199这个区间加锁，只要是id在这个区间内就无法添加
- 临键锁（next-key）：记录锁和间隙锁的结合，封锁范围既包含索引记录也包含索引区间