数据库锁

主要分为悲观锁和乐观锁两种。

# 悲观锁

在操作数据时，认为此操作会出现数据冲突，所以在进行每次操作时都要通过获取锁才能进行对相同数据的操作，这点跟java中的synchronized很相似，所以悲观锁需要耗费较多的时间。

共享锁和排它锁悲观锁的两种不同实现方式。  
⁃ 共享锁(share lock，s锁)

共享锁指的就是对于多个不同的事务，对同一个资源共享同一个锁。如果事务T对数据A加上共享锁后，则其他事务只能对A再加共享锁，不能加排他锁。获准共享锁的事务只能读数据，不能修改数据。

共享锁是表级的，比如select会对表加共享锁。共享锁也叫读锁。

语法:select \* from table lock in share mode;一般的select语句不带lock in share mode，也默认加共享锁。

⁃ 排它锁(exclusive lock，x锁)

如果事务T对数据A加上排它锁后，则其他事务不能再对A加任何类型的锁。获准排它锁的事务既能读数据，也能修改数据。

排它锁一般是行级的，比如DML(数据操纵语言)操作insert update delete(增该删默认加排它锁)，在执行DML操作时分两步加锁：先加共享锁，再加排它锁。在添加排它锁后，不能添加任何锁直到锁释放、commit或者rollback。 排它锁也叫写锁。

语法：select \* from table for update；

⁃ 扩展

通过DML语句对一张表的某一行进行修改，步骤分为两步：

步骤1：对这张表加一个共享锁，防止别的会话通过DDL(数据定义语言)修改这张表的表结构。DDL要修改这张表，就必须给表加上排它锁。但现在给表加了共享锁，也就排斥了DDL加排它锁；

步骤2：对修改的那一行加一个排它锁，别的会话就不能修改这样行，但是对这张表的其他加的是共享锁而不是排它锁，所以别的会话通过步骤1、2也可以修改其他行。

⁃ 示例

悲观锁实现方式。

### Mysql

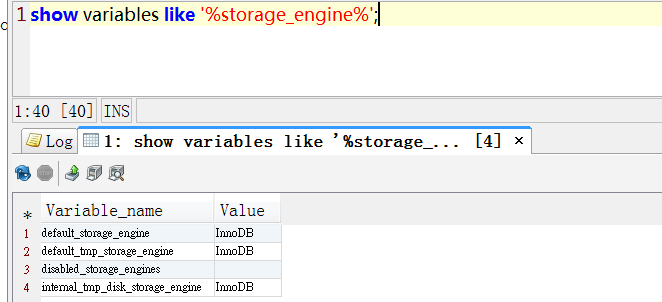
Mysql在5.5之前使用MyISAM存储引擎，之后版本使用InnoDB。MyISAM和InnoDB最大的区别是：一InnoDB支持事务；二InnoDB采用行级锁。

在Mysql中，行级锁并不是直接锁记录，而是锁索引。索引分为主键索引和非主键索引两种，如果DML操作了主键索引，Mysql就会锁定这条主键索引；如果一条语句操作了非主键索引，Mysql会先锁定非主键索引，再锁定相关主键索引。

InnoDB行锁是通过给索引项加锁实现的，如果没有索引，InnoDB会通过隐藏的聚簇索引来对记录加锁，也就是表锁，对所有数据加锁。

可通过命令查看引擎版本。如下所示：

show variables like '%storage\_engine%';



首先，建立建表及索引操作，如下所示：

drop table if exists PaymentInfo;

/\*==============================================================\*/

/\* Table: PaymentInfo \*/

/\*==============================================================\*/

create table PaymentInfo

(

Id int not null auto\_increment,

AcceptId varchar(32) not null,

TransNo varchar(32) not null,

Status int not null,

CreatTime varchar(8) not null,

Extent varchar(32),

primary key (Id)

);

1. 共享锁/排他锁加锁示例

因表数据比较少，查询速度快，无法模拟对某一资源加共享锁后，自身可读该资源，其他人也可读该资源，但无法修改。仅给出示例(参考<https://blog.csdn.net/samjustin1/article/details/52210125>)：

---------------------------------------------------开始1-----------------------------------------------------------

T1：select \* from paymentinfo(隐性加S锁);

T2：update paymentinfo set TransNo =’T0000008’;

运行过程:

T1运行 （加共享锁)

T2运行

If T1 还没执行完

T2等......

else

锁被释放

T2执行

end if

T2之所以要等，是因为T2在执行update前，试图对table表加一个排他锁，而数据库规定同一资源上不能同时共存共享锁和排他锁。所以T2必须等T1执行完，释放了共享锁，才能加上排他锁，然后才能开始执行update语句。

---------------------------------------------------结束1-----------------------------------------------------------

---------------------------------------------------开始2-----------------------------------------------------------

T1：select \* from paymentinfo

T2：select \* from paymentinfo

T2不需等待T1执行完就可马上执行，两个锁是可以同时存在于同一资源上。

---------------------------------------------------结束2-----------------------------------------------------------

---------------------------------------------------开始3-----------------------------------------------------------

T1: select \* from paymentinfo

T2: select \* from paymentinfo

T3: update paymentinfo set TransNo = T0000008’

T2不用等T1运行完就能运行，T3却要等T1和T2都运行完才能运行。因为T3必须等T1和T2的共享锁全部释放才能进行加排他锁然后执行update操作。

---------------------------------------------------结束3-----------------------------------------------------------

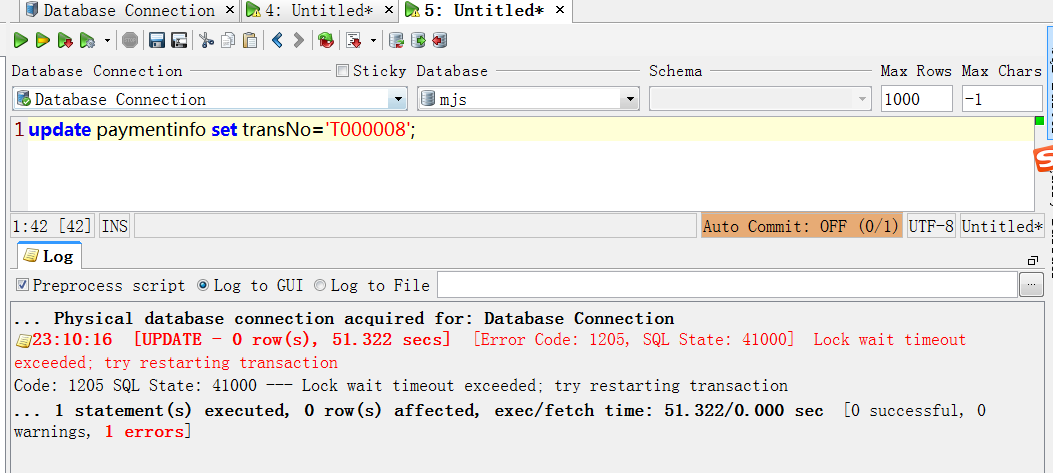
以下示例在Dbvisualizer上可示例，但首先关闭自动提交功能。

---------------------------------------------------开始4-----------------------------------------------------------

T1: update paymentinfo set TransNo = T0000008’

T2: update paymentinfo set TransNo = T0000008’

先执行T1，不进行commit提交。再执行T2，会导致T2一致等待。



---------------------------------------------------结束4-----------------------------------------------------------

---------------------------------------------------开始5-----------------------------------------------------------

死锁的产生

T1: begin;

select \* from paymentinfo lock in share mode;

select sleep(50); --假设该语句执行50s

update paymentinfo set TransNo =' T0000008';

commit;

T2: begin;

select \* from paymentinfo lock in share mode;

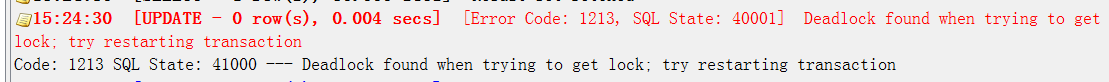
select sleep(5);

update paymentinfo set TransNo =' T0000009';

commit;正常情况：T1 的select、update都执行完后，未进行commit，此时执行T2，T2需等T1事务执行完毕或回滚才能继续执行。

假如T1和T2同时达到select，T1对paymentinfo加共享锁，T2也对paymentinfo加共享锁。当T1的select执行完，准备执行update时，根据锁机制，T1的共享锁需要升级到排它锁才能执行接下来的update，在升级排它锁前，必须等paymentinfo上的其它共享锁释放，但因为T2的共享锁未释放而导致T1在等待，同理，也因为T1的共享锁不释放而导致T2等。死锁就产生了。

执行以上语句T1执行结果如下所示。T1死锁异常抛出后，T2可继续执行。



以下语句常常出现死锁情况:

T1: begin;

update paymentinfo set TransNo =' T0000008';

T1: begin;

update paymentinfo set TransNo =' T0000009';

如果TransNo不是主键且没有索引，T1对TransNo =' T0000008'添加排它锁，T2为了找到TransNo =' T0000009'这一行数据，需要对全表扫描，那么就会预先对表加上共享锁或更新锁或排它锁。但因为T1已经为一条记录加了排它锁，导致T2的全表扫描无法继续进行，就会导致T2等待。

尽在特定情况下出现，T1和T2同时执行并上锁，否则很难重现。

如果TransNo是主键或者有索引，T1和T2互相无影响。

---------------------------------------------------结束5-----------------------------------------------------------

---------------------------------------------------开始6-----------------------------------------------------------

利用更新锁解决死锁：

T1：begin;

select \* from paymentinfo for update;

select sleep(50);

update paymentinfo set TransNo =' T0000008';

commit;

T2：begin;

select \* from paymentinfo for update;

select sleep(5);

update paymentinfo set TransNo =' T0000009';

commit;

T3: select \* from paymentinfo;

T1执行select时加更新锁(即既获取了共享锁，也获取了排它锁)，T2执行时只能等待T1执行完毕后才能继续执行。但是不影响T1执行。T3执行完后加共享锁，不需要等T1和T2执行完毕就可以查询返回结果。

注，如果T1和T2的select…for update增加查询条件，若作用域为为索引，且分为不同行，则更新锁不需要等待。若为非索引，锁作用域为整表，就得等待执行完毕才能继续。

---------------------------------------------------结束6-----------------------------------------------------------

其他:for update对数据库表进行加锁，如果其他事务进行修改，必须等该事务提交或回滚才能进进行。如果条件id为索引，仅进行锁行操作，如果条件id为非索引，进行锁表操作。

行级锁是基于索引的。如果一条 SQL 语句用不到索引是不会使用行级锁的，会使用表级锁把整张表锁住。

# **乐观锁**

乐观锁假设数据在一般情况下不会存在冲突。所以在数据提交后才正式对数据冲突与否进行检测，如果冲突了返回错误信息，让客户决定如何去做。

乐观锁不使用数据库提供的锁机制。一般实现乐观锁的方式是记录版本。

CAS就是一项乐观锁技术，当多个线程尝试使用CAS同时更新同一个变量时，只有其中一个线程能更新变量的值，而其他线程都失败，失败的线程并不会被挂起，而是被告知该次竞争中失败，并可以再次尝试。

## ⁃ 示例

以增加版本号方式来实现乐观锁。

执行以下语句：

alter table paymentinfo add version int(11);

update paymentinfo set version=0 where id=1;

commit;

T1：update paymentinfo set TransNo='T000010',version=1 where version=0;

select sleep(10);

commit;

T2：update paymentinfo set TransNo='T000010',version=1 where version=0;

commit;

先执行T1，再执行T2。发现T2必须在T1执行完毕后才能继续执行update操作(T2等待T1释放排它锁)。T1的DML update 结果是rows为1，T2结果rows为0。

假设T1和T2同时执行，T1和T2只有一条执行成功，其他都是执行失败的。

### 扩展

负载均衡服务多台服务器上只有一台执行数据库跑批任务，其余的不进行跑批。常常使用的做法是增加同步标识字段来控制：

Sql语句表达式:

drop index Uk\_ParamInfo;

drop table if exists ParamInfo;

/\*==============================================================\*/

/\* Table: ParamInfo \*/

/\*==============================================================\*/

create table ParamInfo

(

Id int not null auto\_increment,

ParamCode varchar(32) not null,

primary key (Id)

);

create unique index Uk\_ParamInfo on ParamInfo (

ParamCode ASC

);

利用唯一索引特性，相同插入语句只能执行一个，其他只能执行失败。

--查询是否有该标志

select \* from ParamInfo where ParamCode='DataSynFlag';

//if(不存在){

insert into ParamInfo (ParamCode) values('DataSynFlag');

//if(插入成功){

//return true

//}else{插入失败,说明已有其他服务器锁定或数据库异常

// return false;

//}

//}else{存在

//return false;

//}