2021年5月14日

15:55

Mysql目前有两种主流的引擎:Innodb,MyISAM

MyISAM:支持表锁(偏读)

Innodb:支持行锁，支持事务

Mysql的默认引擎是Innodb

Mysql是cs架构的软件，Mysql服务器可以同时相应多个客户端，每个客户端与服务器连接可以被称之为一个会话(Session)，那么服务器可能需要同时处理多个会话中的事务。当多个事务同时访问同一数据时，那么便可能会出现错误。

同时事务有一个特性，“隔离性”，理论上在某个事务对某条数据进行访问时，其他想处理该条数据的事务应该排队，当该事务提交之后，其他事务才可以继续访问这个数据.但是这样的串行处理会大大降低Mysql对事务并发处理的能力。

所以Mysql需要一种机制让系统提供不同级别处理并发事务的能力，这便是Mysql的事务隔离机制

Innodb引擎事务隔离机制有四种：

READ UNCOMMITTED :未提交读

READ COMMITTED :提交读

REPEATABLE READ :可重复读

SERIALIZABLE :串行化

下面分别分析这四种隔离机制：

首先在mysql中创建一张用于演示的的表：图(1)

Create table `ttd`(

`id` int(10) NOT NULL AUTO\_INCREMENT,

`value` varchar(10) DEFAULT NULL,

PRIMARY KEY(`id`)

)

mysql> desc ttd; 
Field I 
value I 
rows In 
Type 
I NLIU I 
Key I 
PRI 
I bigint(20) I 
varchar(32) I 
Default I 
NULL 
NULL 
Extra 
auto _ inc renent I 
YES 
set (€.0€ sec) 

图(1)

查看当前mysql设置的默认事务隔离级别:show variables like 'transaction\_isolation'(见图2);

mysql> show variables like •transaction _ isolation • ; 
Variable name 
I Value 
transaction _ isolation I æPEATABl_E -æAD I 
1 row in set (€.0€ sec) 

图(2)

**READ UNCOMMITTED**

|  |  |  |
| --- | --- | --- |
|  | SESSION-1 | SESSION-2 |
| 1 | 开启两个session-1 | session-2 |
| 2 | 查询两个session当前的事务隔离级别：select @@session.tx\_isolation;(图3) | select @@session.tx\_isolation;(图4) |
|  | mysql> select  I REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  mysql>  (图3) | mysql> select  I REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  mysql>  (图4) |
| 3 | session-1各自开启事务:start transaction/begin; | session-2 各自开启事务:start transaction/begin; |
| 4 | session-1中执行命令:select \* from ttd; | session-2中执行命令:select \* from ttd; |
| 5 | 在session-1中插入数据:insert into values(1,'a');(图5) | 在session-2中执行命令:select \* from ttd;(图6) |
|  | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  Empty set (€.0€ sec)  mysql> insert into ttd values(l,  Query OK, 1 row affected (€.0€ sec)  (图5) | mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql>  mysql>  mysql> select  Empty set (€.0€  mysql> select  Empty set (€.0€  from ttd;  sec)  from ttd;  sec)  (图6) |
| 6 |  | 当前隔离级别中  session-2中无法读到session-1中未提交事务中对数据的更新 |
| 7 |  | 设置session-2事务隔离级别为READ UNCOMMITTED，命令:set session transaction isolation level read uncommitted (图7) |
|  |  | mysql> set session transaction isolation level read uncommitted;  Query OK, 0 rows affected (0.00 sec)  mysql> select  READ -LINCOWITTED  1 row in set, 1 warning (0.00 sec)  (图7) |
| 8 | session-1各自开启事务:start transaction/begin; | session-2 各自开启事务:start transaction/begin; |
| 9 | session-1中执行命令:select \* from ttd; | session-2中执行命令:select \* from ttd; |
| 10 | 在session-1中插入数据:insert into values(2,'b');(图8) | 在session-2中执行命令:select \* from ttd;(图9) |
|  | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  mysql> insert into ttd values(2,  Query OK, 1 row affected (€.0€ sec)  mysql>  (图8) | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  mysql>  select from ttd;  value I  lid I  lla  21b  in set (€.0€ sec)  rows  mysql>  (图9) |
| 11 |  | 当session-1未提交时，  在session-2中能够读到session-1中未提交事务中对数据的更新 |
| 12 | 在session-1中执行命令:rollback;图(10) |  |
|  | mysql> select  I REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  mysql> insert into ttd values(2,  Query OK, 1 row affected (€.€1 sec)  mysql> rollback;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  mysql> commit;  Query OK, 0 rows affected (€.0€ sec)  图(10) | mysql> select  I READ-UNCOMMITTED  1 row in set, 1 warning (0.00 sec)  mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  mysql>  select from ttd;  value I  lid I  lla  21b  in set (€.0€ sec)  rows  mysql>  select from ttd;  lid I  value I  Illa  1 row in set (€.0€ sec)  mysql>  图(11) |
|  |  | 所以当事务隔离级别设置为:read uncommitted时,会读取到session-1的未提交的数据，当session-1回滚时，如果session-2采用了脏读数据进行逻辑计算，结果会出现问题,图(11) |

**READ COMMITTED**

|  |  |  |
| --- | --- | --- |
|  | SESSION-1 | SESSION-2 |
| 1 | 开启两个session-1 | session-2 |
| 2 | 查询两个session当前的事务隔离级别：select @@session.tx\_isolation;(图12) | select @@session.tx\_isolation;(图13) |
|  | mysql> select  I REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  mysql>  (图12) | mysql> select  I REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  mysql>  (图13) |
| 3 | session-1各自开启事务:start transaction/begin; | session-2 各自开启事务:start transaction/begin; |
| 4 | session-1中执行命令:select \* from ttd; | session-2中执行命令:select \* from ttd; |
| 5 | 当session-2执行完update之后且未提交，在session-2中执行命令:select \* from ttd;(图15);然后在session-2提交之后再查一遍 | 在session-2中更新数据:update set value = 'b' where id =1;(图14) |
|  | nysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  nysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  nysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  nysql> select from ttd;  I id I value I  Il la  1 row in set  (€.0€ sec)  (图15) | mysql>  select from ttd;  lid I  value I  Illa  1 row in set (€.0€ sec)  mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il la  1 row in set (€.0€ sec)  mysql> update ttd set value  •b' where id  Query OK, 1 row affected (€.0€ sec)  Rows matched: 1 Changed: 1 Warnings: 0  mysql> commit;  Query OK, 0 rows affected (€.€1 sec)  (图14) |
| 6 | 当前隔离级别中  不管session-2的更新是否提交,只要session-1还未提交都无法读到其他事务对同一数据的修改 |  |
| 7 | 设置session-1事务隔离级别为READ COMMITTED，命令:set session transaction isolation level read committed (图16) |  |
|  | mysql> set session transaction isolation level read committed;  Query OK, 0 rows affected (€.0€ sec)  mysql> select  I READ-COMMITTED  1 row in set, 1 warning (0.00 sec)  mysql>  (图16) |  |
| 8 | session-1各自开启事务:start transaction/begin; | session-2 各自开启事务:start transaction/begin; |
| 9 | session-1中执行命令:select \* from ttd; | session-2中执行命令:select \* from ttd; |
| 10 | 当session-2修改数据后但未提交前在session-1中执行命令:select \* from ttd;  待session-2提交之后再查一遍  (图18) | 在session-2中更新数据:update ttd set value = 'c' where id =1;(图17),后再提交事务 |
|  | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  111b  1 row in set (€.0€ sec)  mysql> select from ttd;  I id I value I  111b  1 row in set (€.0€ sec)  mysql> select from ttd;  I id I value I  Il lc  1 row in set (€.0€ sec)  mysql>  (图18) | mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  111b  1 row in set (€.0€ sec)  mysql> update ttd set value  •c' where id  Query OK, 1 row affected (€.0€ sec)  Rows matched: 1 Changed: 1 Warnings: €  mysql> commit;  Query OK, 0 rows affected (€.€1 sec)  mysql>  (图17) |
| 11 | 所以当事务隔离级别设置为read committed(提交读)时，另外一个事务修改了当前事务读取的数据，只有当修改数据的事务提交后，当前事务才能读取到其修改。那么会出现在当前事务尚未提交下，不同时间读取同一条数据会得到不同的结果，所以read committed又叫"不可重复读" |  |

**REPEATABLE READ**

|  |  |  |
| --- | --- | --- |
|  | SESSION-1 | SESSION-2 |
| 1 | 开启两个session-1 | session-2 |
| 2 | 查询两个session当前的事务隔离级别：select @@session.tx\_isolation;(图19) | select @@session.tx\_isolation;(图20) |
|  | mysql> select  REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  (图19) | mysql> select  REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  (图20) |
| 3 | session-1各自开启事务:start transaction/begin; | session-2 各自开启事务:start transaction/begin; |
| 4 | session-1中执行命令:select \* from ttd; |  |
| 5 | 当session-2提交之后执行命令:select \* from ttd; 图(22) | 在session-2中插入数据:insert into ttd values(2, 'd');并提交(图21) |
|  | mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il lc  1 row in set (€.0€ sec)  mysql> select from ttd;  I id I value I  Il lc  1 row in set (€.0€ sec)  mysql>  (图22) | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> insert into ttd values (2,  Query OK, 1 row affected (€.0€ sec)  mysql>  select from ttd;  value I  lid I  llc  rows  mysql>  in set (€.0€ sec)  commit ;  Query OK, 0 rows affected (€.€1 sec)  (图21) |
| 6 | 当前隔离级别中  不管session-2的更新是否提交,只要session-1还未提交都无法读到其他事务对同一数据的修改 |  |
| 7 | 执行命令提交session-1,之后再查询一遍 图(23) |  |
|  | mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd;  I id I value I  Il lc  1 row in set (€.0€ sec)  mysql> select from ttd;  I id I value I  Il lc  1 row in set (€.0€ sec)  mysql> commit;  Query OK, 0 rows affected (€.0€ sec)  mysql>  select from ttd;  value I  lid I  llc  in set (€.0€ sec)  rows  mysql>  (图23) |  |
| 8 | 还有一种情况是在session-1开始事务后并不查询某条数据 | 在session-2中对该条数据进行查询，然后修改该数据之后提交 |
| 9 | 当session-2提交之后，session-1对该数据进行查询 | session-2再次开启事务插入新数据:insert into ttd values(4, 'f');后提交 见图(24) |
| 10 | session-1再次查询数据，见 图(25) |  |
| 11 | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql>  select from ttd;  value I  lid I  llc  rows  3  mysql>  in set (€.0€ sec)  select from ttd;  value I  lid I  llc  rows  3  mysql>  In  set (€.0€ sec)  图(25) | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> insert into ttd values(3,  Query OK, 1 row affected (€.0€ sec)  mysql> commit;  Query OK, 0 rows affected (€.0€ sec)  mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> insert into ttd values(4,  Query OK, 1 row affected (€.0€ sec)  mysql>  select from ttd;  value I  lid I  llc  41 f  4 rows  mysql>  in set (€.0€ sec)  commit ;  Query OK, 0 rows affected (€.€1 sec)  mysql>  图(24) |
| 12 | repeatable read,即可重复读，当事务开始且并未提交之时，在事务内读取某条数据,**能保证该数据在事务内一直和该事务第一次读取该数据的状态保持一致** |  |

**幻读**

|  |  |  |
| --- | --- | --- |
|  | SESSION-1 | SESSION-2 |
| 1 | session-1 执行命令:ALTER TABLE `ttd` ADD COLUMN `name` varchar(20) NOT NULL default '' ;  为ttd表添加字段name 见图(26) |  |
|  | mysql> ALTER TABLE 'ttd' ADO COLUMN  Query OK, 0 mws affected (0.12 sec)  Records: € Duplicates: € Warnings •  mysql> desc ttd;  name  varchar(20) NOT NIJI_L default  I Field I  I id  I value I  Type  bigint (20)  varchar(32) I  I varchar(20) I  I NLIU I  YES  NO  Key I  PRI  Default I  NULL  NULL  Extra  auto _ inc renent I  I name  3 rows  mysql>  In  set (€.0€ sec)  select from ttd;  lid I  value I name I  llc  41 f  4 rows  In  set (€.0€  sec)  mysql>  图(26) |  |
| 2 | 查看session-1事务隔离级别 见图(27) | 查看session-2事务隔离级别 见图(28) |
|  | mysql> select  REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  图(27) | mysql> select  REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  图(28) |
|  | session-1 开启事务 | session-2开启事务 |
|  | session-1 执行命令:update ttd set value = 'a' , name = 'harry'; 见图(29) |  |
|  |  | session-2 执行命令:insert into ttd values('a', ''); 见图(30) |
|  | mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql>  select from ttd;  value I name I  lid I  lla  21b  4 rows  mysql>  in set (€.0€ sec)  update ttd set value  , name =  •harry';  Query OK, 4 rows affected (0.00 sec)  Rows matched: 4 Changed: 4 Warnings: 0  mysql>  select from  value I name  ttd;  lid I  lla  2 la  3 la  4 la  4 rows  I harry I  I harry I  I harry I  I harry I  mysql>  in set (€.0€ sec)  commit ;  Query OK, 0 rows affected (€.€1 sec)  mysql>  图(29) | mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql>  select from ttd;  value I name I  lid I  lla  21b  rows  4  mysql>  in set (€.0€ sec)  insert into ttd values(  ERROR 1136 (21S01): Column count doesn •t match value count at row 1  mysql> insert into ttd values(5,  Query OK, 1 row affected (46.51 sec)  mysql>  图(30) |
|  | 只有当session-1 提交之后 | session-2的insert 操作才执行成功,否则session-2一直被挂起，即当前mysql的repeatable read 是可以防止出现幻读 |
|  | 幻读就是在session-1执行update之后,session-2进行了insert数据，且insert成功，这个时候session再进行查询时会发现数据中的name存在两条值其中一条还是session-1修改前的""，就像产生了幻觉一样。幻读主要强调的是在事务中更新数据时，另一个事务进行插入数据操作 |  |

**SERIALIZABLE**

事务在对某一行数据进行处理时,读会加"读锁",写会加"写锁",当出现读写冲突时,后访问的事务需要等待前一个事务执行完成后才能继续执行

**实现原理**

**版本链/多版本控制(mvcc Multi-Version-Concurrency-Control)**

对于Innodb引擎来说，其聚簇索引的记录中都包含两个必要的隐藏列：

trx\_id:每次对某条聚簇索引记录进行改动时,都会把当前的事务id赋值给该行的trx\_id隐藏列

roll\_pointer:每次对某条聚簇索引记录进行改动时，都会把旧版本写入到undo日志中,该行的会保存一个指向该undo日志的指针,可以通过这条指针找到当前记录修改前的信息 见图(31)

value 
4 
name 
harry 
trx_id 
80 
roll_pointer 
inært undo 

图(31)

|  |  |  |
| --- | --- | --- |
| 步骤(按时间发生顺序) | trx 100 | trx 200 |
| 1 | begin; |  |
| 2 |  | begin; |
| 3 | update ttd set name = 'hermione' where id=4 |  |
| 4 | update ttd set name = 'ron' where id=4 |  |
| 5 | commit; |  |
| 6 |  | update ttd set name = 'luna' where id=4 |
| 7 |  | update ttd set name = 'draco' where id=4 |
| 8 |  | commit; |

每次对记录进行改动时，都会记录一条undo log，每条undo log也都有一个roll\_pointer属性(只有update操作有insert操作没有roll\_pointer)，可以将这些undo log串联起来形成一个链表,即称为版本链 见图(32)

id 
4 
id 
4 
id 
4 
id 
4 
id 
4 
value 
value 
value 
value 
value 
name 
draco 
name 
luna 
name 
ron 
name 
hermiore 
name 
harry 
trx_id 
200 
trx_id 
200 
trx_id 
100 
trx_id 
100 
trx_id 
80 
roll_pointer 
roll_pointer 
roll_pointer 
roll_pointer 
roll_pointer 

图(32)

版本链的头节点即是当前记录的最新值

**ReadView**

对于级别为read uncommitted的事务来说，直接读取记录的最新版本即可，对于级别为serializable的事务来说,使用行锁来访问记录；

而对于使用read committed和repeatable read级别的事务来说需要通过版本链来确定该行数据。当前的核心问题是不同的事务隔离级别在不同的时间能看到版本链中的的哪个版本的数据。

mysql是通过ReadView的机制来控制事务对版本链中版本的可见

ReadView中主要是包含当前系统中正在活跃的事务(**即存在但并未提交的事务**)，把这些事务的id放到一个列表中，当访问某条记录时，需要按照下面步骤来判断某个版本对当前事务是否可见:

1.如果被访问的版本的trx\_id小于ReadView列表中最小事务id时，该版本记录在ReadView生成前就已经提交，可以被当前事务访问

2.如果被访问版本的trx\_id大于ReadView列表中最大的事务Id时，表明该记录在ReadView生成之后才生成，该版本的记录无法被当前事务访问

3.如果被访问的版本的trx\_id介于ReadView列表中最大事务id和最小事务id之间，需要进一步判断该记录的trx\_id是否在ReadView中，如果在ReadView中，说明在生成ReadView时该版本事务还是活跃的，未提交的，无法被访问。如果不在，说明该版本记录已提交，可以被访问

在当前事务隔离级别下，如果访问某行记录的某个版本而却又不可见时，顺着版本链找到其下一个版本的数据，按照以上规则再次进行判断，直到找到可见的版本为止。如果直到最后一个版本也是不可见的话，那么意味着该条记录对当前事务不可见，查询结果中不包含该记录

在mysql中，read committed和repatable read主要的区别就在于其生成ReadView的时机不同

**read committed**

例如当前有id为200和100的事务正活跃

|  |
| --- |
| transaction 100 |
| begin; |
| update ttd set name = 'hermione' where id = 4; |
| update ttd set name = 'ron' where id = 4; |
| … |

|  |
| --- |
| transaction 200 |
| begin; |
| … |

注：在事务的执行过程中,只有在第一次真正修改记录时(insert,update,delete)，才会被分配一个单独的事务id，且事务id是递增的

现在有一个隔离级别为read committed的事务开始执行

|  |
| --- |
| #隔离级别为read committed |
| begin; |
| #当transaction 100,transaction 200 未提交的前提下 |
| select \* from ttd where id = 4 |
| … |

如下图：

|  |  |  |
| --- | --- | --- |
|  | SESSION-1 | SESSION-3 |
| 1 | mysql>  select from  value I name  ttd;  lid I  lla  2 la  3 la  4 la  4 rows  I harry I  I harry I  I harry I  I harry I  mysql>  in set (€.0€ sec)  select  I REPEATABLE -READ  1 row in set, 1 warning (0.00 sec)  mysql> begin;  Query OK, 0 rows affected (€.0€ sec)  mysql> update ttd set name  •hemione• where id  Query OK, 1 row affected (0.00 sec)  Rows matched: 1 Changed: 1  mysql> update ttd set name  Query OK, 1 row affected (€.0€ sec)  Rows matched: 1 Changed: 1  mysql>  4;  Warnings: €  •ron• where id  Warnings: €  4; | mysql> set session transaction isolation level read committed;  Query OK, 0 rows affected (€.0€ sec)  mysql> select  I READ-COMMITTED  1 row in set, 1 warning (0.01 sec)  mysql> strart transaction;  1064 (42000): You have an error in your SQI_ syntax; check the manual that corresponds to your  mysql> start transaction;  Query OK, 0 rows affected (€.0€ sec)  mysql> select from ttd where id = 4;  id I value  4 la  1 row in set  I name I  I harry I  (€.0€ sec) |

session-3在执行select时具体逻辑如下:

1.在执行select语句时会先生成一个ReadView，ReadView中包含的id为[100,200]

2.session-3在where id=4的记录版本链中查找当前事务可见的版本，id=4的记录版本链见图(33)

3.该记录的最新版本name = ron，trx\_id = 100,100包含在ReadView列表内，不符合可见性要求，所以根据roll\_pointer执行找到该记录的下一个版本

4.下一个版本 name = hermione，trx\_id = 100，同样不符合可见性要求，再往下

5.下一个版本name = harry ,该版本trx\_id=80，小于ReadView列表中的最小值，所以对当前事务是可见的，返回该版本记录即：name = harry

id 
4 
id 
4 
id 
4 
value 
value 
value 
name 
ron 
name 
hermiore 
name 
harry 
trx_id 
100 
trx_id 
100 
trx_id 
80 
roll_pointer 
roll_pointer 
roll_pointer 

图(33)

然后将transaction 100进行提交，然后在transaction 200中继续更新id=4的数据

|  |
| --- |
| transaction 100 |
| begin; |
| … |
| … |
| **commit;** |

|  |
| --- |
| transaction 200 |
| begin; |
| update ttd set name = 'luna' where id = 4; |
| update ttd set name = 'draco' where id = 4; |

隔离级别为read committed的事务继续执行

|  |
| --- |
| #隔离级别为read committed |
| begin; |
| #在transaction 100已提交,transaction 200 未提交的前提下 |
| select \* from ttd where id = 4 |
| … |

结果见下图:

|  |  |  |  |
| --- | --- | --- | --- |
|  | SESSION-1 | SESSION-2 | SESSION-3 |
| 1 | 计算机生成了可选文字: mysq select  丨 id value |  name  harry  harry  harry  harry  rows in set (€.00 sec }  mysqb begin;  Query ， rows affected (€.00 sec)  mysqb update 11d set name  Where  Query [ ， 1 row affected { · sec)  Rows matched ： 1 Changed ： 1 rnings ：  mysqb update 11d set name  ron ' Where  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed: 1 Warnings:  mysqb ℃ omm 置 ：  Query ， rows affected (€.01 sec)  mysqb | 计算机生成了可选文字: mysql:• Sta rt transaction;  Query ， rows affected (€.00 sec)  mysqb update 11d set name  飞 2 where id = 3 ，  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed ： 1  Warnlngs:  mysqb update 11d set name  'luna where id—4  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed ： 1  Warnlngs:  mysqb update 11d set name  raco ' where id—4  Query ， 1 row affected { .01 sec)  Rows matched ： 1 Changed ： 1  Warnlngs:  mysqb | 计算机生成了可选文字: mysql:• select @@session.tx isolation ：  丨 @@session.tx isolation 丨  丨 ÆAD -COMMITTED  1 row in set, 1 warning { · sec }  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec }  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec }  mysqb |

session-3在trx\_id=100已提交,trx\_id=200已更新数据但未提交的情况下，执行select 语句其逻辑如下

1.read committed隔离级别下,事务每次执行select之前都会重新生成ReadView，由于trx\_id=100已提交，当前活跃的事务只剩trx\_id=200,生成的ReadView:[200]

2.session-3在记录where id =4的版本链中查找当前事务可见的版本,当前id=4记录的版本链见图(34)

3.当前该记录的最新版本为trx\_id=200,name = 'draco',trx\_id=200包含在当前ReadView中，不符合可见性要求，所以根据roll\_pointer执行找到该记录的下一个版本

4.下一版本记录的trx\_id=200,name = 'luna',trx\_id=200包含当前ReadView中，不符合可见性要求，再向下继续寻找

5.再下一个版本的记录trx\_id=100,name = 'ron',trx\_id=100不包含在当前ReadView中，且小于ReadView中的最小值，符合可见性要求，返回这个版本的记录查询结果，即name = ron

id 
4 
id 
4 
id 
4 
id 
4 
id 
4 
value 
value 
value 
value 
value 
name 
draco 
name 
luna 
name 
ron 
name 
hermiore 
name 
harry 
trx_id 
200 
trx_id 
200 
trx_id 
100 
trx_id 
100 
trx_id 
80 
roll_pointer 
roll_pointer 
roll_pointer 
roll_pointer 
roll_pointer 

图(34)

**总结:read committed隔离级别生成ReadView的时机是,*在事务中每次查询前都会重新生成一次ReadView*,查询当前记录的版本trx\_id,只要trx\_id符合以下规则即返回记录：trx\_id<min(ReadView) || (min(ReadView)<trx\_id<max(ReadView) && (trx\_id != min(ReadView) && trx\_id != max(ReadView)))**

**repeatable read**

还是以transaction 100,200举例

|  |
| --- |
| transaction 100 |
| begin; |
| update ttd set name = 'hermione' where id = 4; |
| update ttd set name = 'ron' where id = 4; |
| … |

|  |
| --- |
| transaction 200 |
| begin; |
| … |

隔离级别为repeatable read的事务开始执行

|  |
| --- |
| #隔离级别为repeatable read |
| begin; |
| #当transaction 100,transaction 200 未提交的前提下 |
| select \* from ttd ; |
| … |

见下图

|  |  |  |  |
| --- | --- | --- | --- |
|  | SESSION-1 | SESSION-2 | SESSION-3 |
| 1 | 计算机生成了可选文字: mysql:• Sta rt transaction;  Query ， rows affected (€.00 sec)  mysqb update 11d set name  Query ， 1 row affected { .01 sec)  Rows matched ： 1 Changed ： 1  Warnlngs:  mysqb update 11d set name  ron ' Where  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed: 1  Warnlngs:  Where | 计算机生成了可选文字: mysql:• Sta rt transaction,  Query ， rows affected (€.00 sec)  mysqb update 11d set name  飞 2 where id—3,  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed: 1 Warnings: | 计算机生成了可选文字: mysql:• Sta rt transaction,  Query ， rows affected (€.00 sec)  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec } |

session-3(隔离级别为repeatable read)在trx\_id=100,trx\_id=200已更新数据但未提交的情况下，执行select 语句其逻辑如下

1.repeatable read隔离级别下,事务只会在事务第一次进行查询时生成ReadView，由于trx\_id=100,trx\_id=200均未提交,生成的ReadView:[100,200]

2.session-3在记录where id =4的版本链中查找当前事务可见的版本,当前id=4记录的版本链见图(35)

3.当前该记录的最新版本为trx\_id=100,name = 'ron',trx\_id=100包含在当前ReadView中，不符合可见性要求，所以根据roll\_pointer执行找到该记录的下一个版本

4.下一版本记录的trx\_id=100,name = 'hermione',trx\_id=100包含当前ReadView中，不符合可见性要求，再向下继续寻找

5.再下一个版本的记录trx\_id=80,name = 'harry',trx\_id=80不包含在当前ReadView中，且小于ReadView中的最小值，符合可见性要求，返回这个版本的记录查询结果，即name = harry

id 
4 
id 
4 
id 
4 
value 
value 
value 
name 
ron 
name 
hermiore 
name 
harry 
trx_id 
100 
trx_id 
100 
trx_id 
80 
roll_pointer 
roll_pointer 
roll_pointer 

图(35)

接下来将trx\_id=100的事务提交，在session-3中再次查询id=4的记录，结果如下图

|  |  |  |
| --- | --- | --- |
|  | SESSION-1 | SESSION-3 |
| 1 | 计算机生成了可选文字: mysql:• Sta rt transaction;  Query ， rows affected (€.00 sec)  mysqb update 11d set name  Where  Query ， 1 row affected { .01 sec)  Rows matched ： 1 Changed ： 1 rnings ：  mysqb update 11d set name  ron ' Where  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed ： 1 Warnings:  mysqb commit ：  Query ， rows affected (€.00 sec)  mysqb | 计算机生成了可选文字: mysql:• Sta rt transactlon,  Query ， rows affected (€.00 sec)  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec }  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec } |

然后再到trx\_id=200的session中对id=4的记录进行更新，在session-3中再次查询id=4的记录，结果如下图:

|  |  |  |
| --- | --- | --- |
|  | SESSION-2 | SESSION-3 |
| 1 | 计算机生成了可选文字: mysql:• Sta rt transaction,  Query ， rows affected (€.00 sec)  mysqb update 11d set name  飞 2 where id—3  Query ， 1 row affected (€.00 sec)  Rows matched ： 1 Changed ： 1  Warnlngs:  mysqb update 11d set name  'luna where id  Query OK, 1 row affected { .00 sec)  Rows matched ： 1 Changed ： 1  Warnlngs:  mysqb update 11d set name  raco ' where id  Query ， 1 row affected { .00 sec)  Rows matched ： 1 Changed ： 1  Warnlngs: | 计算机生成了可选文字: mysql:• Sta rt transaction,  Query ， rows affected (€.00 sec)  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec }  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec }  mysqb select ' from 11d ：  丨 id 丨 value 丨 name  丨 harry  丨 harry  丨 harry  丨 harry  rows in set (€.00 sec } |

session-3(隔离级别为repeatable read)在trx\_id=100已提交,trx\_id=200已更新数据但未提交的情况下，执行select 语句其逻辑如下

1.repeatable read隔离级别下,事务只会在事务第一次进行查询时生成ReadView，由于trx\_id=100已提交,trx\_id=200未提交,此次为事务中第三次进行查询，session-3事务的ReadView仍是第一次查询时生成的:[100,200]

2.session-3在记录where id =4的版本链中查找当前事务可见的版本,当前id=4记录的版本链见图(36)

3.当前该记录的最新版本为trx\_id=200,name = 'draco',trx\_id=200包含在当前ReadView中，不符合可见性要求，所以根据roll\_pointer执行找到该记录的下一个版本

4.下一版本记录的trx\_id=200,name = 'luna',trx\_id=200包含当前ReadView中，不符合可见性要求，再向下继续寻找

5.下一版本trx\_id=100,name = 'ron',trx\_id=100包含在当前ReadView中，不符合可见性要求，所以根据roll\_pointer执行找到该记录的下一个版本

6.下一版本记录的trx\_id=100,name = 'hermione',trx\_id=100包含当前ReadView中，不符合可见性要求，再向下继续寻找

7.再下一个版本的记录trx\_id=80,name = 'harry',trx\_id=80不包含在当前ReadView中，且小于ReadView中的最小值，符合可见性要求，返回这个版本的记录查询结果，即name = harry

id 
4 
id 
4 
id 
4 
id 
4 
id 
4 
value 
value 
value 
value 
value 
name 
draco 
name 
luna 
name 
ron 
name 
hermiore 
name 
harry 
trx_id 
200 
trx_id 
200 
trx_id 
100 
trx_id 
100 
trx_id 
80 
roll_pointer 
roll_pointer 
roll_pointer 
roll_pointer 
roll_pointer 

图(36)

**总结:在session-3事务隔离级别为repeatable read的情况下，其多次查询的结果都是相同的。*repeatable read在事务第一次进行查询时生成ReadView*(与在分析可重复读时的结论一致)，查询当前记录的版本trx\_id,只要trx\_id符合以下规则即返回记录：trx\_id<min(ReadView) || (min(ReadView)<trx\_id<max(ReadView) && (trx\_id != min(ReadView) && trx\_id != max(ReadView)))**