1. 一张表，里面有 ID 自增主键，当 insert 了 17 条记录之后，删除了第 15,16,17 条记录，再把 Mysql 重启，再 insert 一条记录，这条记录的 ID 是 18 还是 15 ？

如果存储引擎类型是MyIsam，那么结果是18，因为MyIsam会把自增主键id的最大值存在文件中，服务重启也不会丢失。如果是InnoDB，那么结果是15，因为InnoDB是把自增主键的最大值记录到内存中，所以重启服务会丢失。但是mysql8.0版本之后，innodb的结果是18了。

1. Heap 表是什么？

它是基于memory存储引擎创建的表，数据存在于内存中，用于临时高速存储。不支持自增主键和事务，索引不能为NULl,支持b-tree索引和hash索引。memory存储引擎又叫heap存储引擎。

1. FLOAT 和 DOUBLE、decimal的区别

Float：浮点型，含字节数为4，数值只有7个有效位，如果不指定精度，默认会按照实际精度存储；

Double：双精度类型，含字节数为8，数值有15个有效位，如果不指定精度，默认会按照实际精度存储；

Decimal：数字类型，不存在精度损失，28个有效位,如果不指定精度，则默认整数位为10，小数位为0。如果超过了指定的长度和精度，会提示警告。

超过精度的都会四舍五入。

1. char与varchar的区别：

char：长度固定，即char(M)类型的数据列里，每个值都占用M个字节，如果某个长度小于M，MySQL就会在它的右边用空格字符补足。

varchar:长度是可变的，即在varchar(M)类型的数据列里，每个值只占用刚好够用的字节再加上一个用来记录其长度的字节（即总长度为L+1字节）。

声明为char的列长度是固定的,char的长度可选范围在0-255之间.也就是char最大能存储255个字符.如果该列是utf8编码,则该列所占用的字节数=字符数\*3.如果是gbk编码则该列所占用的字节数=字符数\*2。

声明为varchar的列长度是可变的,在mysql5.0.3之前varchar的长度范围为0-255,mysql5.0.3之后varchar的长度范围为0-65535个字节.采用varchar类型存储数据需要1-2个字节(长度超过255时需要2个字节)来存储字符串的实际长度.如果该列的编码为gbk,每个字符最多占用2个字节,最大长度不能超过32766个字符.如果该列的编码为utf8,每个字符最多占3个字节,最大字符长度为21845.

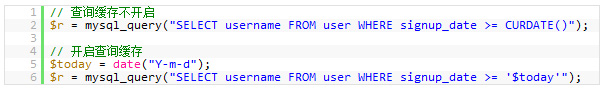
1. mysql字符串类型的Enum、blob、set

blob是二进制类型，可以容纳可变数量的数据，最大65k；

Enum与set都是枚举类型，enum只能存一个值，set可以存多个

1. 性能优化：

①如果开启了查询缓存，sql中如果有内置函数作为条件，那么改为用变量替换掉查询语句中的内置函数。如：



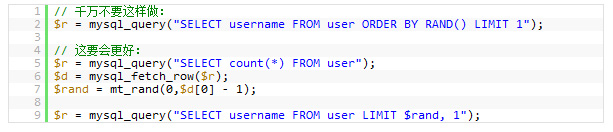
②使用explain解析查询语句，然后分析新能瓶颈

③当只要查询一行数据时，再查询语句后面加上limit 1;

④为搜索字段加索引；

⑤如果是join查询，为join字段加索引；

⑥千万不要用order by rand(),rand()是内置函数，执行的时候非常耗性能。就算加上limit 1也无济于事。



⑦避免使用select (\*) 和select count(\*)

⑧能用Enum就尽量不要用varchar，因为数据库实际上保存的tinyint类型。

⑨字段尽量不要设置成默认为Null，因为null需要额外的空间，并且比较的时候更复杂。

⑩从 PROCEDURE ANALYSE() 取得建议。

1. 唯一索引比普通索引快吗？为什么？

不一定，还可能慢。

1. 查询时，在未使用limit 1的情况下，唯一索引查询到一条数据后立马返回，普通索引会继续匹配下一条数据，发现没了匹配数据之后才返回，虽然普通索引多花了时间但是这个时间差可能可以忽略不计，如果使用了limit 1，则都是立马返回，查询时间差不多。
2. 新增、修改数据时，唯一索引还需要额外校验字段的唯一性,需要花费更长的时间.
3. MySQL怎么恢复半个月前的数据？

通过整库备份+binlog日志进行恢复，前提是要有定期整库备份，且保存了binlog日志。

1. MyISAM和InnoDB\*\*的区别有哪些？

InnoDB支持事务, MyISAM不支持.

InnoDB支持行级锁, MyISAM支持表级锁.

InnoDB支持多版本并发控制(MVVC), MyISAM不支持.

InnoDB支持外键, MyISAM不支持.

MyISAM支持全文索引, InnoDB部分版本不支持(但可以使用Sphinx插件)

1. mysql架构或者说是由那几个部分组成？

1. Server

连接器: 管理连接, 权限验证.

分析器: 词法分析, 语法分析.

优化器: 执行计划生成, 索引的选择.

执行器: 操作存储引擎, 返回执行结果.

1. 存储引擎: 存储数据, 提供读写接口.
2. MySQL查询缓存有什么弊端, 应该什么情况下使用, 8.0版本对查询缓存有什么变更?

查询缓存可能会失效非常频繁, 对于一个表, 只要有更新, 该表的全部查询缓存都会被清空. 因此对于频繁更新的表来说, 查询缓存不一定能起到正面效果。对于读远多于写的表可以考虑使用查询缓存.8.0版本的查询缓存功能被删了。

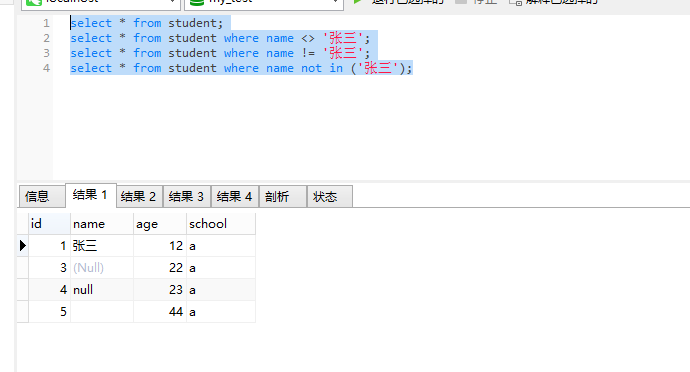
1. 一千万条数据的表如何分页？

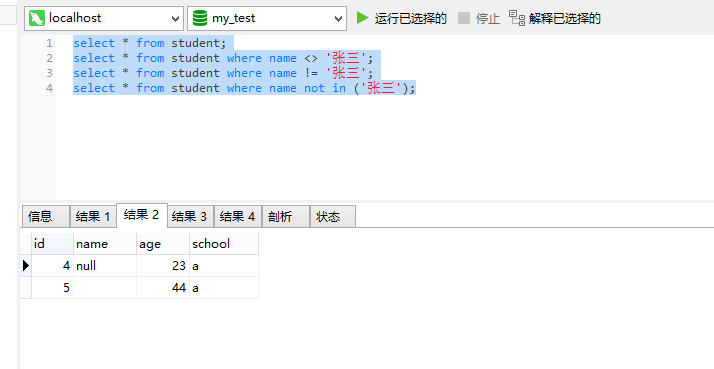
数据量过大的情况下，limit offse会由于扫描数据太多导致越往后查询越慢，可以适当配合最后当前页最后一条主键id进行判断，即where条件后加上主键id >或<,比如：SELECT \* FROM T WHERE id > #{ID} LIMIT #{LIMIT} . 当然, 这种情况下ID必须是有序的, 这也是有序ID的好处之一。

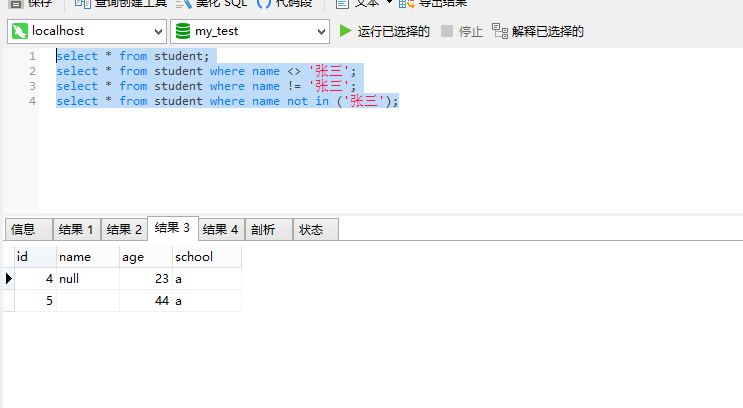
1. 订单表数据越来越大导致查询慢，如何处理？

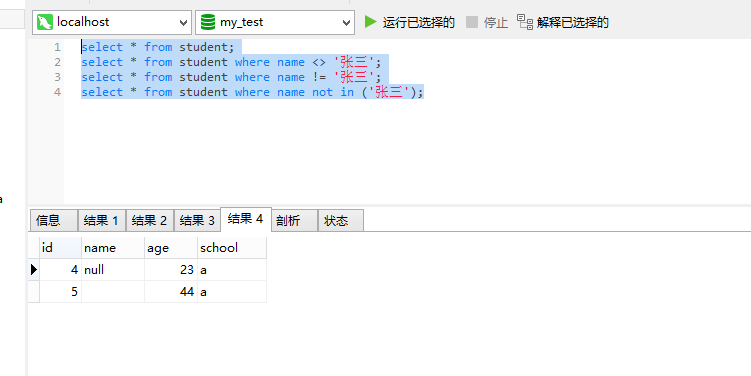
分库分表，由于历史订单的使用率不高，因此可以将订单表按时间进行拆分，根据数据量的大小考虑按月分表或按年分表. 订单ID最好包含时间(如根据雪花算法生成), 此时既能根据订单ID直接获取到订单记录, 也能按照时间进行查询。

1. 在mysql查询语句中where用<>、!=、not in()时会过滤当前字段为空（is null）的数据，如下图：









1. 整数类型数据的unsigned有什么作用?

MySQL 中的整数类型可以使用可选的 UNSIGNED 属性来表示不允许负值的无符号整数,即无符号整数。使用 UNSIGNED 属性可以将正整数的上限提高一倍，因为它不需要存储负数值。

例如， TINYINT UNSIGNED 类型的取值范围是 0 ~ 255，而普通的 TINYINT 类型的值范围是 -128 ~ 127。INT UNSIGNED 类型的取值范围是 0 ~ 4,294,967,295，而普通的 INT 类型的值范围是 -2,147,483,648 ~ 2,147,483,647。

对于从 0 开始递增的 ID 列，使用 UNSIGNED 属性可以非常适合，因为不允许负值并且可以拥有更大的上限范围，提供了更多的 ID 值可用。

1. 为什么不推荐使用text和blob类型?

因为text、blob的长度范围都是0~65535，而varchar的范围也是0~65535，相比之下，text、blob不能有默认值，并且不能创建索引，可能会消耗大量的网络和IO,所以，建议把text和blob类型改为varchar。

1. NULL 和 '' 的区别是什么?

NULL 代表一个不确定的值,就算是两个 NULL,它俩也不一定相等。例如，SELECT NULL=NULL的结果为 false，但是在我们使用DISTINCT,GROUP BY,ORDER BY时,NULL又被认为是相等的。

''的长度是 0，是不占用空间的，而NULL 是需要占用空间的。

NULL 会影响聚合函数的结果。例如，SUM、AVG、MIN、MAX 等聚合函数会忽略 NULL 值。 COUNT 的处理方式取决于参数的类型。如果参数是 \*(COUNT(\*))，则会统计所有的记录数，包括 NULL 值；如果参数是某个字段名(COUNT(列名))，则会忽略 NULL 值，只统计非空值的个数。

查询 NULL 值时，必须使用 IS NULL 或 IS NOT NULLl 来判断，而不能使用 =、!=、 <、> 之类的比较运算符。而''是可以使用这些比较运算符的。

1. Boolean 类型如何表示？

MySQL 中没有专门的布尔类型，而是用 TINYINT(1) 类型来表示布尔值。TINYINT(1) 类型可以存储 0 或 1，分别对应 false 或 true。

1. 存储引擎MyIsam与InnoDB中索引的实现方式异同点？

· InnoDB 支持行级别的锁粒度，MyISAM 不支持，只支持表级别的锁粒度。

· MyISAM 不提供事务支持。InnoDB 提供事务支持，实现了 SQL 标准定义了四个隔离级别。

· MyISAM 不支持外键，而 InnoDB 支持。

· 虽然 MyISAM 引擎和 InnoDB 引擎都是使用 B+Tree 作为索引结构，但是两者的实现方式不太一样。

他们的索引都是以B+Tree作为索引结构。Innodb中索引分为聚簇索引和非聚簇索引，除了主键索引或者第一个非空Unique列的唯一索引或者隐藏的row-id是聚簇索引，其他索引都是非聚簇索引，且只能有一个聚簇索引。而MyIsam中所有的索引都是非聚簇索引。

聚簇索引的索引和数据是放在一起的，数据存放在索引结构的叶子节点上。

非聚簇索引的和数据都是以文件的形式分开存放的，索引结构的叶子节点指向了数据对应的位置，索引结构的叶子节点存放的是数据的地址。

InnoDB使用缓冲池(buffer pool)存储索引页及数据页，而MyIsam使用键缓存(Key cache)存储索引页。

· MyISAM 不支持数据库异常崩溃后的安全恢复，而 InnoDB 支持（redo-log）。

1. 索引失效的情况？

使用了组合索引，但是没有遵循最左匹配原则；

在索引列进行计算、函数、类型转换操作;

以%开头的like查询，如 like `%abc`;

查询条件中有or，且or的前后条件中有一个列没有建索引，涉及的索引都不会被使用到；

in/not in的取值范围较大时，索引失效;

1. InnoDB的行锁类型？

记录锁（Record Lock）：单行上的锁；

间隙锁（Gap Lock）： 锁定一个范围，不包含记录本身；

临键锁（Next-Key Lock）:它实际上是Record Lock+Gap Lock,锁定一个范围，包括记录本身。为了解决幻读问题。

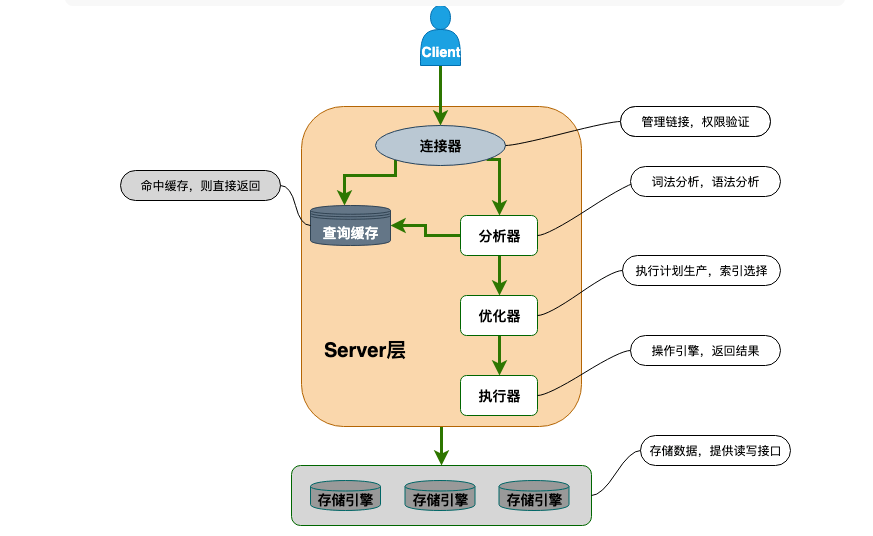
在InnoDB默认隔离级别是Repeatable Read下，行锁默认是使用临键锁（Next-Key Lcok）,但是如果索引是唯一索引或者主键索引，InnoDB会对行锁进行优化，将其降级为Record Lock,仅锁住索引本身，不再是锁住范围。

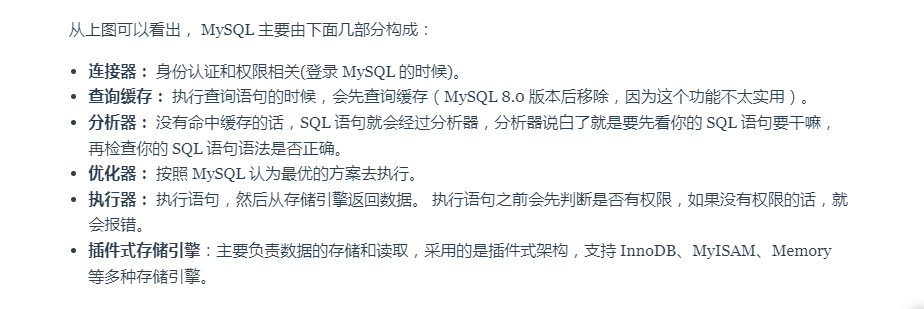
1. Mysql语句显式加共享锁和排他锁(读锁和写锁)





1. Mysql的基础架构，或者sql语句在mysql中的执行过程是怎样的？





1. MVCC实现原理

MVCC的实现依赖于隐藏列、undo log和Read View。

隐藏列：

每一行都有隐藏列:

DB\_TRX\_ID: 最后一次修改该记录的事务ID,读操作时不会产生DB\_TRX\_ID。

DB\_ROLL\_PTR: 回滚指针，本质上就是指向一个该条记录对应的undo log的一个指针，通过这个指针找 到之前版本的数据。

DB\_ROW\_ID: 隐藏的主键，如果一个表中没定义主键，会默认生成一个自增的隐藏主键。

另外，还有一个删除标记隐藏列，当数据执行delete操作时，并不是立即删掉该数据，而是将该数据的删除标记字段设为true,由purge线程 来对这些输出标记为true的数据进行物理删除。

purge线程：

它是定期执行，根据系统的负载自动调整线程的活动频率。比如，系统负载高时，提高purge线程的活跃度，避免undo log积累过多；系统低负载时，降低purge线程的活跃度，减少资源竞争。

purge线程运行时会执行一下操作:1、先根据undo log日志中的信息，找到对应的记录并执行物理删除操作。2、找到删除标记为true的undo log日志。当undo log不再被任何事务需要时，就会被标记为可删除状态。

**undo log：**

又叫回滚日志。

当不同的事务对同意数据进行更新时，会产生多个历史快照，这些快照数据存储在undo log中，这些undo log日志通过DB\_ROLL\_PTR回滚指针串联在一起，即所说的版本链。

任何修改数据的操作（insert、update、delete）都会产生undo log日志，维护了事务的原子性。

1、insert操作时，undo log记录的是这次插入操作的反向操作，即如何从数据库中移除这条插入的记录。

2、update操作时，undo log记录的是更新前的数据状态，如果事务回滚，可以利用该undo log的内容恢复到修改前的值。

3、delete操作时，undo log记录的是被删除数据的信息。

undo log的处理又分两种情况：

1、insert操作时，undo log只有在事务回滚时需要，如果事务提交成功，undo log就会被删除(被标记为可删除状态)，由purge线程清理。

2、update/delete操作时，undo log不仅在事务回滚时需要，在快照读时也需要，所以update/delete操作成功后，undo log不会被立即删除(被标记为可删除状态),等到这个undo log不再被用到时，才会被删除被标记为可删除状态)，由purge线程清理。

**Read View：**

当事务进行快照读时，会对当前记录产生一个读视图，我们把他叫做Read View。它用来判断数据对当前事务的可见性。该Read View有以下几个属性：trx\_list、up\_limit\_id、low\_limit\_id。

trx\_list：DB\_TRX\_ID数组，记录的是生成该Read View时，当前系统中所有活跃的DB\_TRX\_ID。

up\_limit\_id:记录的是trx\_list中最小的db\_trx\_id。

Low\_limit\_id:记录的是当前系统中尚未分配的db\_trx\_id,即db\_trx\_id+1。

Read uncommitted隔离级别下，只有写操作加锁，读操作不加锁，所以，该隔离级别下不会生成Read view,会有脏读的问题。

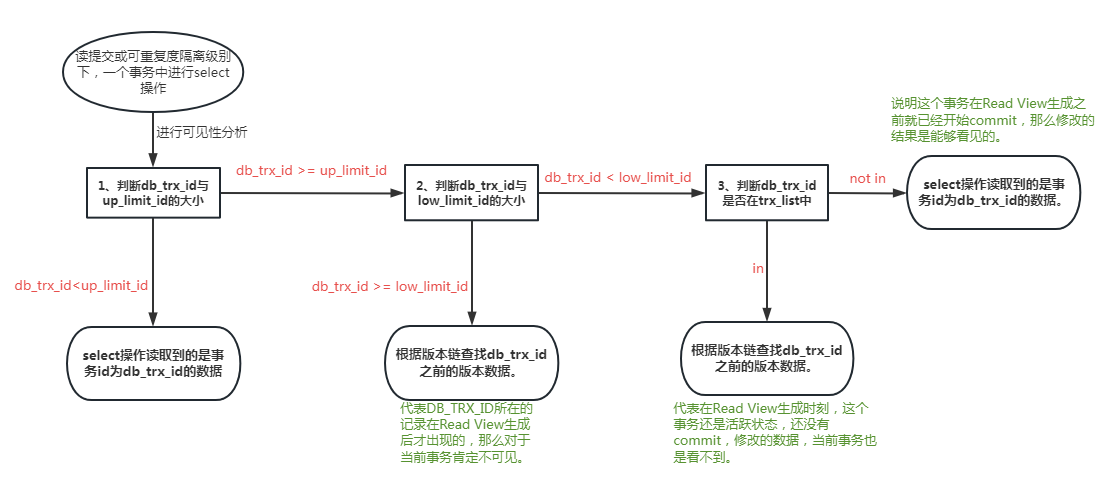
Read committed隔离级别下，对于读操作不会显示的加锁，而是采用”快照读”的策略,每次快照读都会生成Read view，能读取到其他事务提交的最新的数据，有不可重复读的问题。

Repeatable Read隔离级别下，对于读操作不会显示的加锁，也是采用”快照读”的策略,但是在同一个事务中只有第一次快照读会生成Read view，该事务中的其他快照读也读取该Read View,这样实现了可重复度。

Serializable隔离级别下，读操作时通过共享锁机制来实现的，所以他没有快照读，而是当前读（读操作时不允许其他事务修改当前数据）。

**可见性算法过程**:

1. 首先比较DB\_trx\_id 是否小于up\_limit\_id,如果小于，则当前事务的读操作(select语句)能看到DB\_trx\_id所在的记录，如果大于等于则进入下一个判断。
2. 接着第一步，判断DB\_trx\_id是否大于等于low\_limit\_id,如果是，则代表db\_trx\_id所在的记录在Read View生成后才出现的，那么对于当前事务不可见；如果小于，则进入下一步判断。
3. 接着第二步，判断db\_trx\_id是否在trx\_list中，如果在，则表示在Read View生成时刻，这个事务还处于活跃状态(没有commit),其他事务修改的数据，当前事务也看不到，如果不在，说明这个事务在Read view生成之前就已经开始commit,那么修改结果当前事务的读操作能看到。



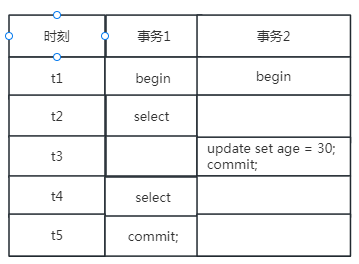
检验算法：

现有student表，有如下图数据：



此时该数据中的db\_trx\_id=1。

并有以下并发事务：



假设事务2的db\_trx\_id为2，因为事务1没有修改操作，所以事务1的dx\_trx\_id=0）。

①在提交度Read Committed(RC)隔离级别下：

事务1执行t2时刻的快照读时，生成Read View(RV),RV的属性值为别为：

trx\_list:为0;

Up\_limit\_id:为0;

Low\_limit\_id的值为2(db\_trx\_id的值+1);

DB\_trx\_id：1

根据可见性分析：

1. DB\_trx\_id > Up\_limit\_id ->
2. DB\_trx\_id < Low\_limit\_id ->
3. db\_trx\_id not in trx\_list -> 说明这个事务在RV生成之前就已经开始commit,那么修改结果当前事务的读操作能看到,即当前select，读取到的是DB\_trx\_id=1的数据（查找过程：根据db\_trx\_id=1找到DB\_trx\_id=1对应的回滚指针db\_roll\_pt，根据回滚指针找到对应的undo log日志）。

事务2在t3时刻update并commit后，

此时生成了undo log, db\_trx\_id=2。

事务1执行t4时刻select快照读，生成新的Read View(RV2),RV2的属性值为别为：

trx\_list:为0;

Up\_limit\_id:为0;

Low\_limit\_id的值为3(db\_trx\_id的值+1);

DB\_trx\_id：2

根据可见性分析：

1. DB\_trx\_id > Up\_limit\_id ->
2. DB\_trx\_id < Low\_limit\_id ->
3. db\_trx\_id not in trx\_list -> 说明这个事务在RV生成之前就已经开始commit,那么修改结果当前事务的读操作能看到,即当前select，读取到的是DB\_trx\_id=2的数据（查找过程：根据db\_trx\_id=2找到DB\_trx\_id=2对应的回滚指针db\_roll\_pt，根据回滚指针找到对应的undo log日志）。这种情况就造成了不可重复度的情况。

②在可重复读Repeatable Read(RR)隔离级别下：

事务1执行t2时刻的快照读时，生成Read View(RV),RV的属性值为别为：

trx\_list:为0;

Up\_limit\_id:为0;

Low\_limit\_id的值为2(db\_trx\_id的值+1);

根据可见性分析：

1. DB\_trx\_id > Up\_limit\_id ->
2. DB\_trx\_id < Low\_limit\_id ->
3. db\_trx\_id not in trx\_list -> 说明这个事务在RV生成之前就已经开始commit,那么修改结果当前事务的读操作能看到,即当前select，读取到的是DB\_trx\_id=1的数据（查找过程：根据db\_trx\_id=1找到DB\_trx\_id=1对应的回滚指针db\_roll\_pt，根据回滚指针找到对应的undo log日志）。

事务2在t3时刻update并commit后，

此时生成了undo log, db\_trx\_id=2。

事务1执行t4时刻select快照读，不生成新的Read View,还是使用事务1在t2时刻生成的RV的属性值为别为：

trx\_list:为0;

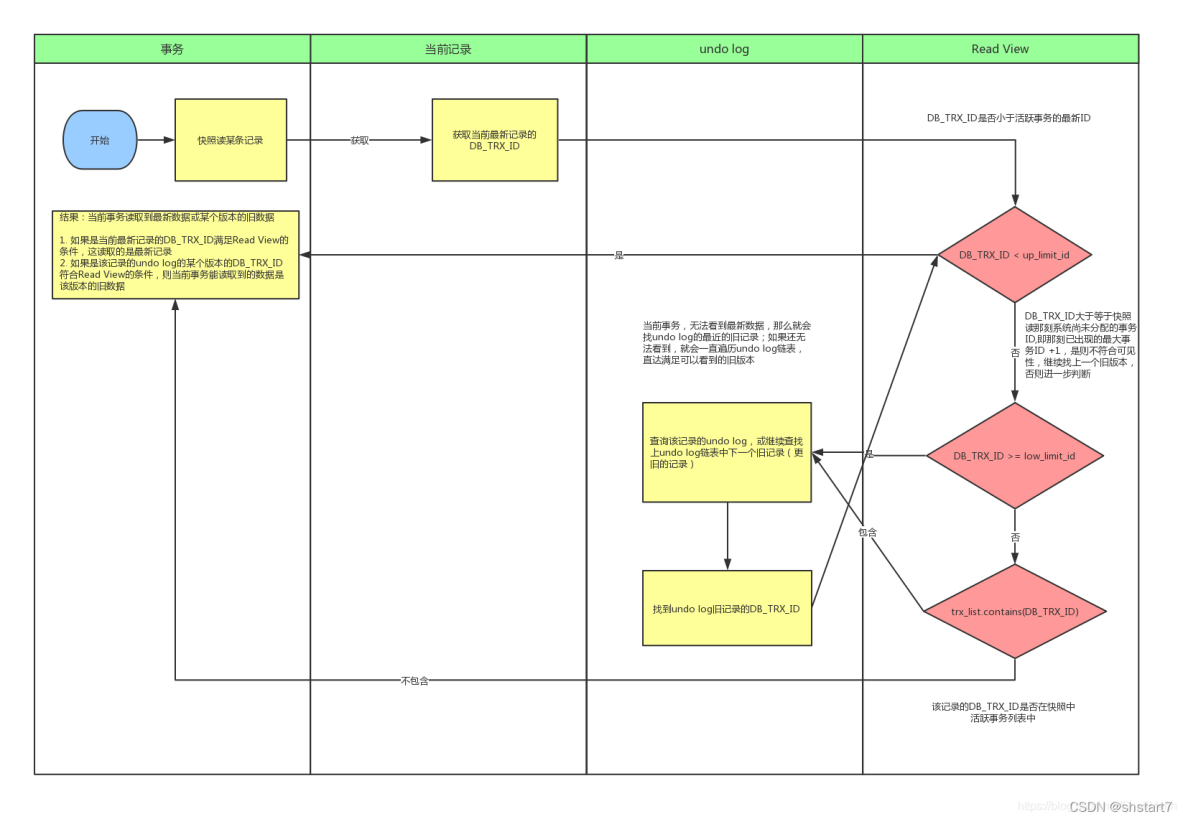
Up\_limit\_id:为0;

Low\_limit\_id的值为2;

根据可见性分析：

1. DB\_trx\_id > Up\_limit\_id ->
2. DB\_trx\_id >= Low\_limit\_id （2=2） ->

说明代表DB\_TRX\_ID所在的记录在Read View生成后才出现的，那么对于当前事务不可见,这时就查找快照undo log中的数据,根据版本链上查找，找到db\_trx\_id=2之前版本的数据。



1. Mvcc如何解决幻读问题？

利用快照读和写操作加临键锁。

1. Order by原理

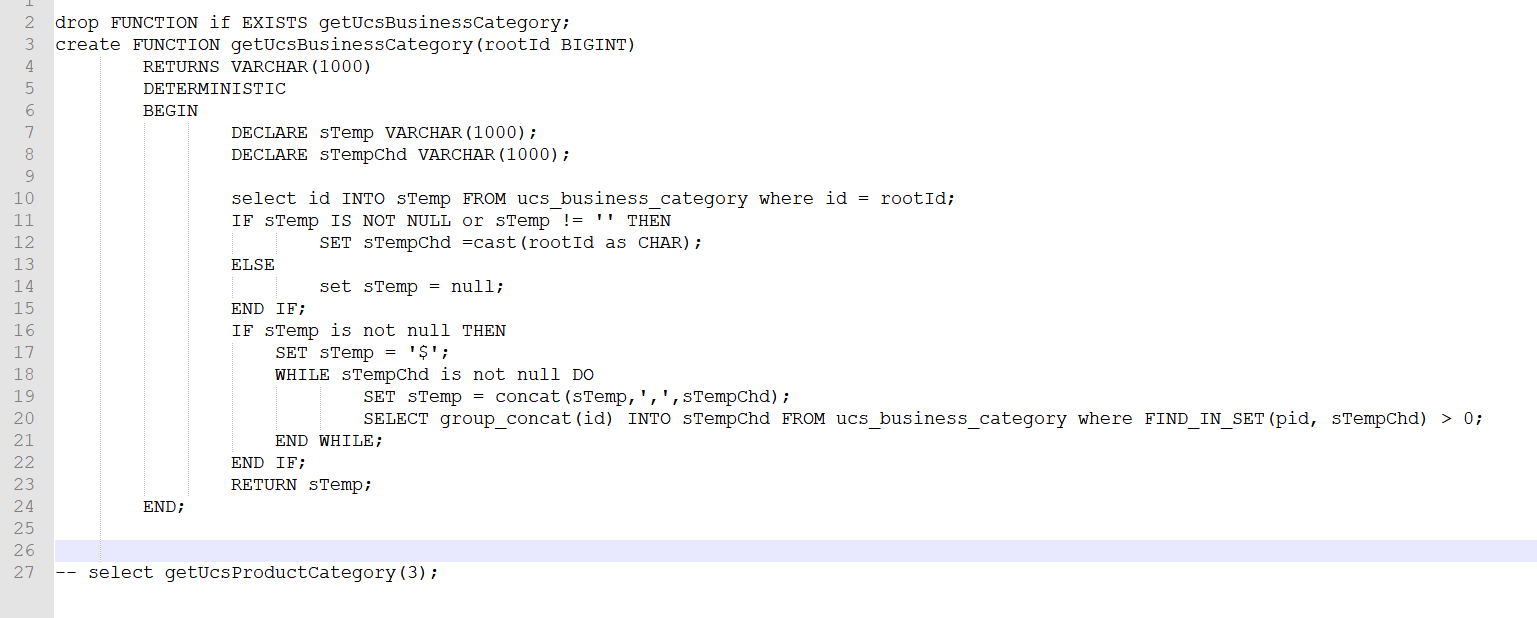
按索引排序，利用索引的有序性。

单路排序:查询的时候一次性去除满足条件的所有字段，然后在sort buffer中排序，比多路排序少一次I/o.

1. mysql中on duplicate key update的用法



1. mysql中function使用



1. mysql中，一天5w条以上的增量，预计运维三年，怎么优化？
2. 设计良好的表结构，选择合适的字段类型和存储引擎，允许部分字段冗余，尽量避免join查询；
3. 适当添加索引；
4. 主从复制读写分离；
5. 按规律分表。
6. 什么情况下设置了索引但无法使用

以”%”开头的like查询语句；

or语句前后没有同时使用索引；

数据类型出现隐式转化，如varchar不加单引号会自动转为int类型；

1. 内连接、外连接、左连接、右连接

内连接，就是两张表之间进行数据匹配是，要同时满足on的条件。

Select \* from tab1 inner join tab2 on XXX;inner可以省略。

左连接和右连接，又叫左外连接和右外连接，即left join=left outer join，right join=right outer join；

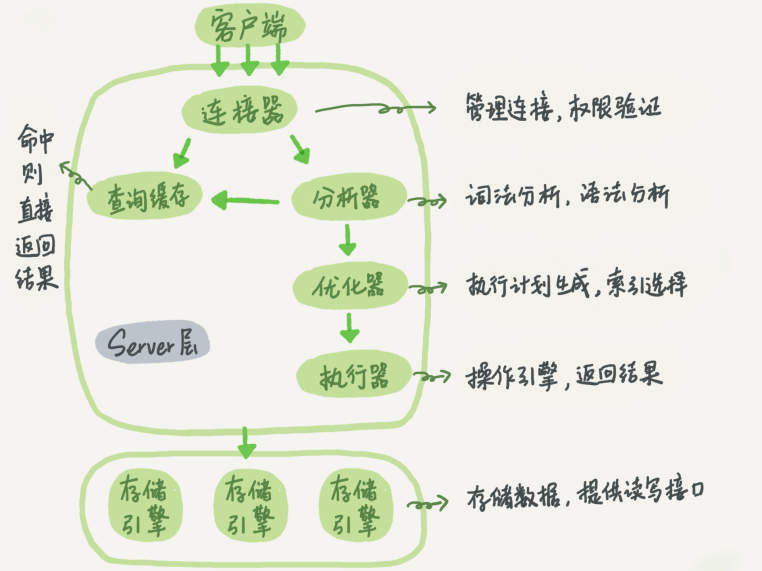
1. union操作符

用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句会删除重复的数据。



1. Mysql整体架构：

mysql在整体架构上分为server和存储引擎。其中server层包括连接器、查询缓存、分析器、优化器、执行器；存储引擎负责存储数据，提供数据的读写接口。



连接器：负责客户端与服务端建立连接，获取用户权限及维持和管理连接；

查询缓存：当接收到查询请求时，会先在查询缓存中查询是否执行过。查询缓存一般没必要设置，因为表的数据被更新时，缓存就会被清空，所以只适用于静态表，mysql8.0后缓存被废除；

分析器：识别表名、字段名是否存在，语句是否符合mysql语法；

优化器：确定索引的使用，join表的连接顺序等，选择最优化的方案；

执行器：在具体执行语句前，会先进行权限的检查，通过后使用数据引擎提供的接口，进行查询。不预先检查的原因：如像存储过程、触发器等情况，需要在执行器阶段才能确定权限，在优化器阶段无法验证。

参考文档：<https://www.cnblogs.com/michael9/p/12497992.html>

34、存储引擎：

主流的有MyIsam、Innodb、Memery、Archive、CSV等共8种。

查看数据库各个表的存储引擎：

SELECT table\_name, table\_type, engine FROM information\_schema.tables WHERE table\_schema = 'ucs'

Innodb:

底层结构为B+树，每个节点对应innodb的一个page，大小固定默认为16K.

使用场景：

①经常更新的表，适合处理多重并发的更新请求；

②支持事务；

③外键约束，至于innodb支持外键；

④支持自增主键；

⑤可以通过bin-log日志从灾难中恢复数据。

适用于提供提交、回滚、崩溃恢复能力的事物安全（ACID兼容）能力，并要求实现并发控制的设计。

存储文件有frm(表定义文件)、ibd(数据文件)。Innodb的行锁是所在索引上的，而不是所在物理行记录上，如果访问表时没有命中索引，也无法使用行锁，将要变为表锁。

MyIsam:

Mysql的默认引擎（mysql5.5之前）。

不提供事务，不支持行级锁和外键，因此插入和修改数据时锁表，效率低，但查询效率高。

存储文件有frm(表定义文件)、myd(数据文件)、myi是索引文件。

Memory:

内容存放在堆内存中，所以访问速度很快。默认使用HASH索引，同时也支持B+Tree索引，但是一旦服务器关闭，表中的数据就会丢失，但表还会继续存在。

适合临时存放数据，数据量不大，且数据安全性不高的设计

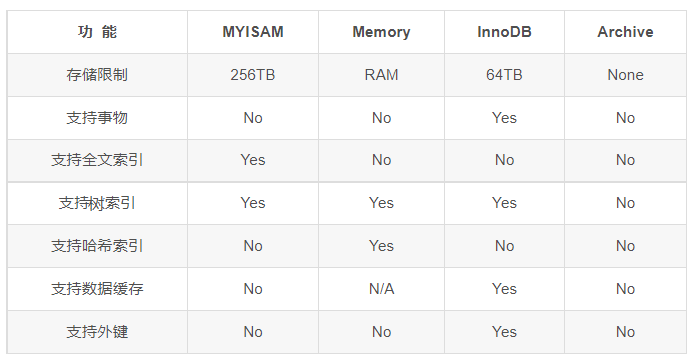
Achive:

只支持inserthe select。插入快速，非事务安全。

适合存储归档数据，如记录日志信息或者历史记录。

CSV:

把数据以逗号分隔的格式存储在文本文件中。



MyIsam与innodb的区别：

①**事务：**Myisam不支持事务，Innodb支持事务；

②**外键：**Myisam不支持外键，Innodb支持外键；

③**锁：**Myisam只支持表级锁，Innodb支持行级锁和表级锁，默认行级锁；

④**全文索引：**Myisam支持全文索引，Innodb不支持全文索引，从mysql5.6开始Innodb支持全文索引；

**表主键**：Myisam允许没有主键的表存在，InnoDB如果么有创建主键，就会自动生成一个6字节的主键(用户不可见)；

⑥**表的行数**：MyIsam中内置了一个计数器，可以直接读取select count(\*),InnoDB中select count(\*)需要扫描全表

；

一张表,里面有 ID 自增主键,当 insert 了 17 条记录之后,删除了第 15,16,17 条记录,再把 Mysql 重启,再 insert 一条记录,这条记录的 ID 是 18 还是 15 ？

如果表的类型是 MyISAM， 那么是 18。因为 MyISAM 表会把自增主键的最大 ID 记录到数据文件里， 重启MySQL自增主键的最大 ID 也不会丢失。

如果表的类型是 InnoDB， 那么是 15。InnoDB 表只是把自增主键的最大 ID 记录到内存中， 所以重启数据库会导致最大 ID 丢失。

35、索引类型：

Innodb中非主键索引使用的是B-Tree数据结构，而主键索引使用的是B+Tree。

FULLTEXT(全文索引)

再检索长文本的时候，效果最好，innbMyIsam支持。

NOMAL(普通索引)

大多情况下都能使用，目的是通过索引提高查询效率。

SPATIAL(空间索引)

空间索引是堆空间数据类型(geometry、point、linestring、polygon)建立的索引。创建空间索引的列。必须将其声明为not null,且只能在存储引擎为MyIsam的表中创建。

UNIQUE(唯一索引)

唯一索引是在表上一个或者多个字段组合建立的索引，建立唯一索引的字段不能重复。

聚集索引与非聚集索引：

两者的根本区别是表记录的排列顺序和索引的排列顺序是否一致。即索引与行记录是存储在一起的称为聚集索引，否则称之为非聚集索引。所以InnoDB中必须要要有聚集索引，如果定义了主键，则主键就是聚集索引，如果没有定义主键，第一个非空Unique列索引就是聚集索引，否则InnoDB会创建一个隐藏的row-id作为聚集索引。一个表中聚集索引有且只能有一个。

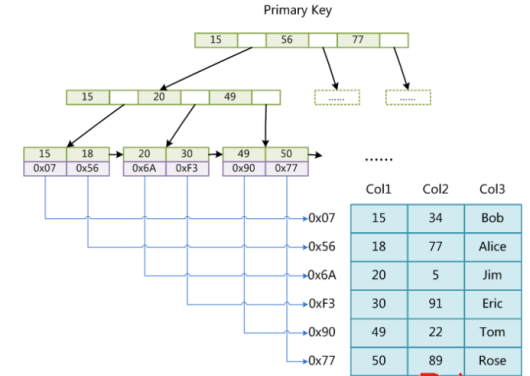
主键注意点：

①不建议使用较长的列做主键，例如varchar(64)，因为所有的普通索引都会存储主键，会导致普通索引过于庞大；

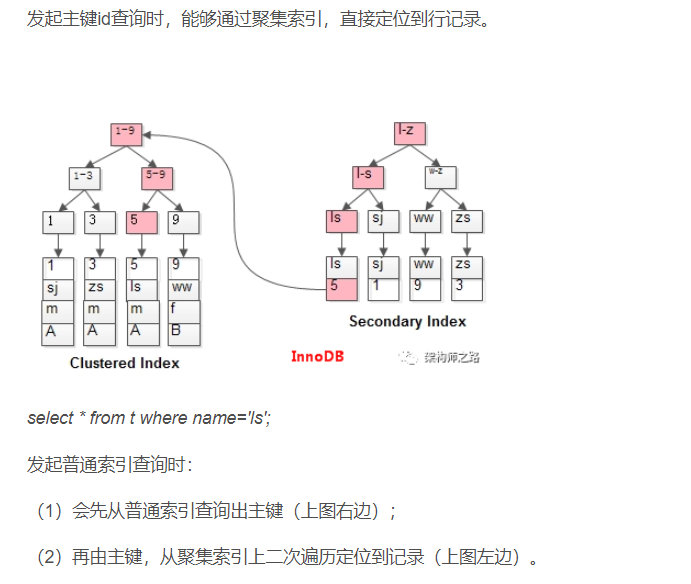
②建议使用自增id作为主键；

索引方法有BTREE、HASH：

MyIsam的B+Tree的叶子节点上的data并不是数据本身，而是数据存放的地址，主键索引与普通索引区别不大。



Innodb的BTree的叶子节点的data就是数据本身。如下图：



Hash索引仅仅能满足”=”、”in”、”<=>”查询,所以对like查询无效;不能使用范围查询;不支持索引排序，索引值和计算出来的hash值大小不一定相等。

为什么InnoDB推荐使用自增ID作为主键？

自增id可以保证每次插入时BTree索引是从右边扩展的，可以避免B+Tree频繁合并和分裂(对比使用UUID)。如果使用字符串之间和随机主键，会是的数据随机插入，效率比较差。

36、索引建立原则：

①最左前缀匹配原则：mysql 会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，范围查询会导致组合索引半生效。比如 a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，c 可以用到索引，d 是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d 的顺序可以任意调整。where范围查询要放在最后 （这不绝对，但可以利用一部分索引）。例如索引是(a,b,c),可以支持a|a,b|a,b,c 3种组合进行查找

②特别注意：and 之间的部分可以乱序，比如 a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式。where 字句有 or 还是会遍历全表。

③不在索引列做运算或者使用函数。

④尽量扩展索引，不要新建索引。比如表中已经有 a 的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可。

⑤where子句中经常使用的字段应该创建索引，分组字段或者排序字段应该创建索引，两个表的连接字段应该创建索引。

⑥like模糊查询中，只有右模糊查询(like ‘苹果%’)会使用索引。

⑦经常修改的字段不要建立索引

⑧索引不会包含有NULL值的列。只要列中包含有NULL值都将不会被包含在索引中，复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为NULL。

⑨使用短索引。对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的 列，如果在前10 个或20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

⑩不要使用not in，可以使用not exists代替。

符合索引的顺序必须和order by子句的顺序一致。

天堂2013 的CSDN博客

https://blog.csdn.net/qq\_34436819/category\_7825147.html

<https://blog.csdn.net/qq_34436819/article/details/105664333>

https://blog.csdn.net/qq\_34436819/article/details/105664297

MySQL中order by语句的实现原理以及优化手段：

<https://blog.csdn.net/z69183787/article/details/105867288>

**37、事务及其隔离级别：**

Mysql的事务满足ACID特性：

原子性(A):事务里的所有操作要么都执行要么都不执行；

一致性(C)：数据的完整性不被破坏，要么都执行成功，要么都不成功；

隔离性(I):对于同一条数据，可以允许多个事务同时对它进行读写操作；

永久性(D):事务处理结束后，对数据的修改是永久的，即使系统故障也不会丢失。

mysql自动默认提交事务，innoDB默认的事务级别是可重复读。

并发事务带来的问题：

①脏读：

读取了其他事务未提交的数据。

eg:公司发工资了，领导把5000元打到A的账号(正常工资2千)上，但是该事务并未提交，而A正好去查看账户，发现工资已经到账，是5000元整，非常高兴。但是领导随后发现给A的工资发多了，于是迅速回滚了事务，修改金额后，将事务提交，最后A实际的工资只有2000元，A空欢喜一场。

②不可重复读：

一个事务内多次读取到的数据不一致，A事务中第一次读到值是1, B事务把1修改成2，并且提交了，A事务中第二次读到值也变成 了2，这样就导致A事务中2次读取的结果不一样，因此称之为不可重复度。针对update操作。

eg:在事务A中，读取到张三的工资为5000，操作没有完成，事务还没提交。与此同时，事务B把张三的工资改为8000，并提交了事务。随后，在事务A中，再次读取张三的工资，此时工资变为8000。在一个事务中前后两次读取的结果并不致，导致了不可重复读。

③幻读：

一个事务内多次读取到的数据不一致，第一次读到1条，第二次读到2条。针对insert，delete，数据行数发生了变化。

eg:A的老婆在银行工作，她可以很方便的查看A的信用卡消费记录。月末了，她正在查询并打印A当月的消费情况(同一个事务) 为80元，而A此时正好在某收银台买单，消费1000元，即新增了一条1000元的消费记录（insert transaction ... ），并提交了事务，随后A的老婆将A当月信用卡消费的明细打印到A4纸上，却发现消费记录多了一条1000元的消费记录，A的老婆很诧异，以为出 现了幻觉，幻读就这样产生了。



可串行化隔离级别：它通过强制事务进行排序，使之不可能相冲突，从而解决幻读问题。简言之，在每个读数据上添加共享锁。会导致大量超时现象和锁竞争。

Mysql中查看和设置隔离级别：

1.查看当前会话隔离级别

select @@tx\_isolation;

2.查看系统当前隔离级别

select @@global.tx\_isolation;

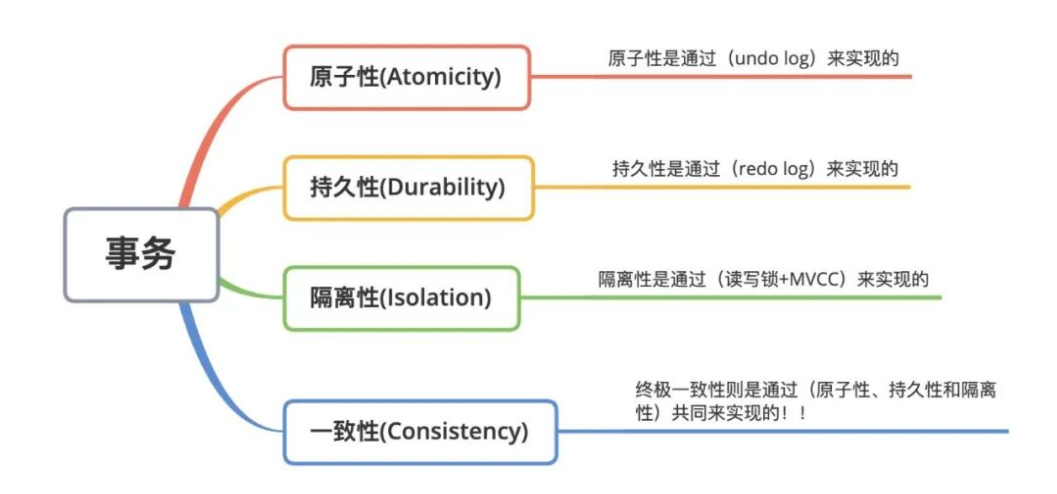
3.设置当前会话隔离级别

set session transaction isolatin level repeatable read;

4.设置系统当前隔离级别

set global transaction isolation level repeatable read;

**38、事务的实现：**



锁的类型：

共享锁(读锁S)：多个读请求可以同时共享一把锁来读取数据，而不会造成阻塞。

排他锁(独占锁/写锁X)：写锁会排斥其他所有获取锁的请求，一直阻塞，直到完成写入并释放锁。读写锁可以做到读读并行，但是无法做到读写、写写并行。事务的隔离性就是根据读写锁来实现的。

锁的粒度：

行锁、表锁。允许行锁和表锁共存。

意向锁(Intention lock)：表级别的锁。先提前声明一个意向，并获取表级别的意向(读/写)锁(IS/IX)，如果获取成功，才能执行操作。

插入意向锁(IX):

假设存在值为4和7的索引记录。尝试分别插入值为5和6的两个独立事务，在获得所插入行上的X锁之前，每个事务都使用IX锁定4和7之间的间隙，但不会阻塞彼此，因为这些行不冲突。

意向锁协议：

①在事务能够获取行的共享锁(S)之前，必须先获取表的意向共享锁(IS)；

②在事务能够获取行的排他锁(X)之前，必须先获取表的意向排他锁(IX)。



间隙锁(gap lock,锁定一个范围，但是不包括记录本身)：

也可以叫做范围锁，他是锁定一定范围之内的记录。一般作用于我们的范围筛选查询>、<、between等。例如：Select \* from tab\_user where userId between 1 and 4 for update；阻止其他事务将userId=3的数据查到表中，因为userId between 1 and 4之间的间隙都是锁定的。

Next-key锁(行锁 + 间隙锁 左开右闭)：

是行锁与间隙锁的组合。innodb中默认隔离级别(RR)下，next key Lock自动开启。

比如，存在一个查询匹配b=3的行(b上有个非唯一索引)，那么所谓nest-key锁就是：在b=3的行加了行锁，并且使用间隙锁锁定了b=3之前(“之前”：索引排序)的所有行记录。

对于有主键和唯一索引的事务，会产生行锁，不会产生表锁、间隙锁；

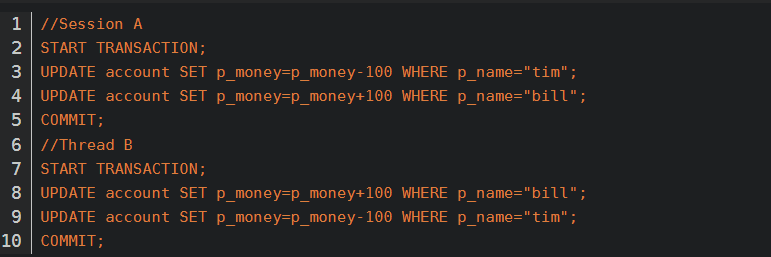
对于没有创建索引的事务，会产生表锁；

对于创建了普通索引的事务，会产生next-key锁。

参考文档：<https://blog.csdn.net/hongtaolong/article/details/120123693>

死锁以及如何尽可能避免死锁：

场景：



当线程A执行到第一条语句UPDATE account SET p\_money=p\_money-100 WHERE p\_name=“tim”;锁定了p\_name="tim"的行数据；并且试图获取p\_name="bill"的数据；此时，恰好，线程B也执行到第一条语句：UPDATE account SET p\_money=p\_money+100 WHERE p\_name=“bill”;锁定了 p\_name="bill"的数据，同时试图获取p\_name="tim"的数据；此时，两个线程就进入了死锁，谁也无法获取自己想要获取的资源，进入无线等待中，直到超时！

如何避免死锁:

①以固定的顺序访问表和行。比如两个更新数据的事务，事务A更新数据的顺序为1，2；事务B更新数据的顺序也写成1，2.

②大事务拆小，事务越庞大，执行的操作越多，越容易出现死锁；

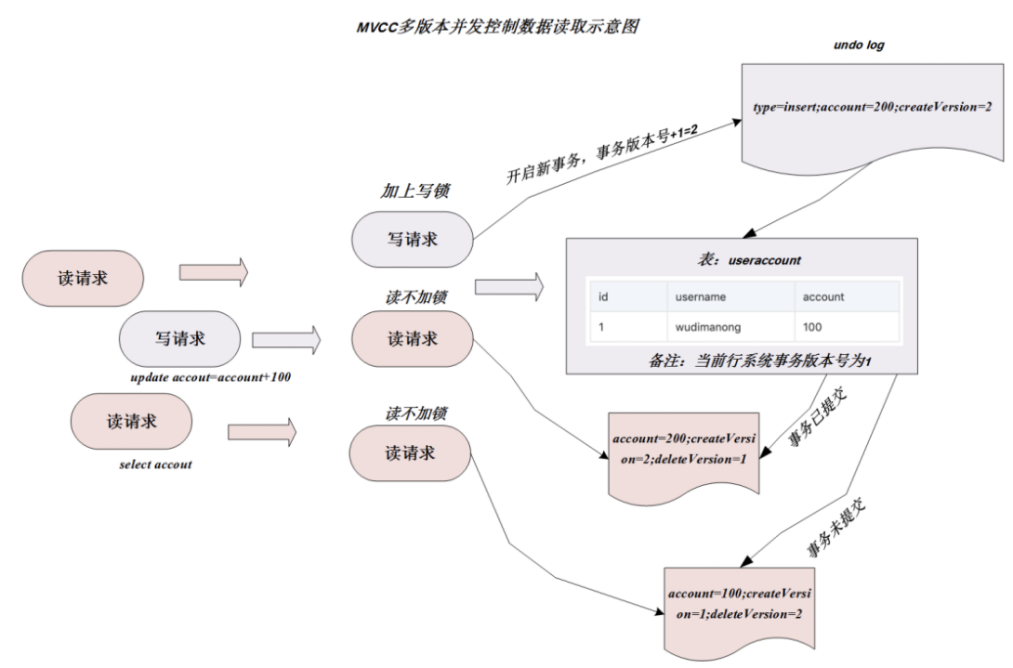
③如果业务允许，可以降低事务隔离级别。从默认可重复度(RR)降低到提交读(RC)，可以避免掉很多因为gap锁造成的死锁。

④合理创建索引，降低死锁的概率。

**39、多版本并发控制(MVCC):**

MVCC(Multi Version Concurrency Control)实现了读写分离，从而实现不加锁读取数据进而做到了读写并行。下面讲一下InnoDB的MVCC实现：

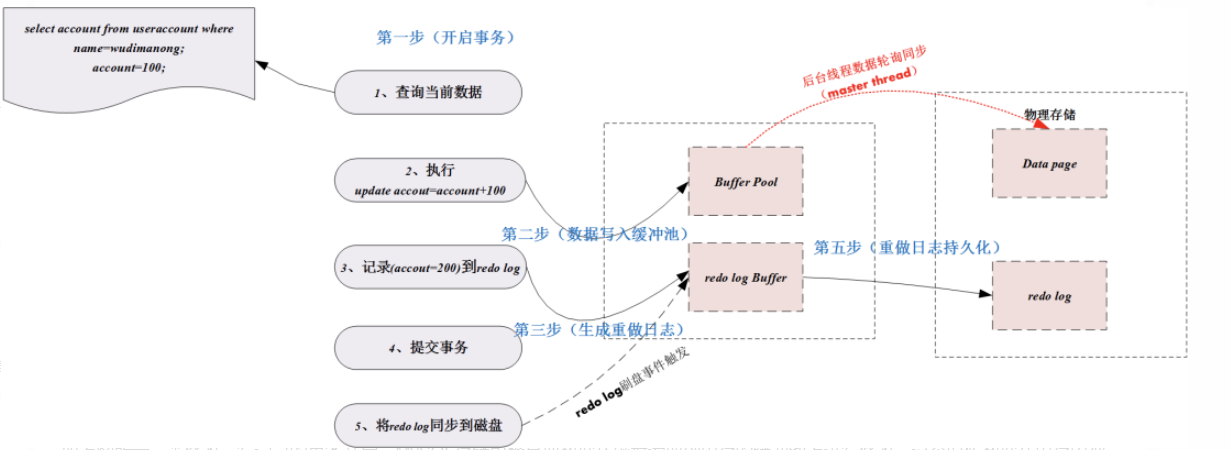
通过在每行记录后面保存两个隐藏列来实现的。这两个列，一个保存行的创建时间另一个保存行的过期时间。它们并不是实际的时间值，而是一个版本号，每开启一个新的事务，版本号都会递增；事务开始时刻的版本号会作为事务的版本号，用来和查询到的每行记录的版本号进行比较。



版本链：mysql对每一条记录都会维护多个版本，会将这些版本按照一定顺序串联起来形成一个链表，实现多版本控制还要借助于undo log和read view。

redo log:重做日志，是实现事务持久化的关键。redo log日志文件主要由2部分组成:重做日志缓冲(redo log buffer)、重做日志文件(redo log file)。

在mysql中为了提升数据库性能并不会把每次的修改都实时同步到磁盘，而是会先存到一个叫做”Buffer pool”的缓冲池中，之后会再使用后台线程轮询去把缓冲池的数据同步到磁盘。这样会造成一个问题，当数据还没来得及从缓冲池同步到磁盘，此时断电或者宕机了，这样会导致已经提交的事务修改操作实际不执行。所以redo log的主要作用就是记录已成功提交事务的修改信息，并且会在事务提交后实时将redo log持久化到磁盘，这样再系统重启之后就可以读取redo log来恢复最新的数据。



undo log：回滚日志，用于记录某行数据的多个旧版本的数据，主要作用是恢复数据。每次写入数据或者修改数据之前存储引擎都会将修改前的信息记录到undo log。如果由于系统错误或者rollback操作而回滚的话就可以根据undo log来将数据回滚到没被修改之前的状态。

参考文档：<https://blog.csdn.net/hongtaolong/article/details/120104126>