mysql笔记：

脏读，不可重复读，幻读：

脏读 :脏读就是指当一个事务正在访问数据,并且对数据进行了修改,而这种修改还没有提交到数据库中,这时,另外一个事务也访问 这个数据,然后使用了这个数据。

不可重复读 :是指在一个事务内,多次读同一数据。在这个事务还没有结束时,另外一个事务也访问该同一数据。那么,在第一个事务中的两 次读数据之间,由于第二个事务的修改,那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的,因此称为是不 可重复读。例如,一个编辑人员两次读取同一文档,但在两次读取之间,作者重写了该文档。当编辑人员第二次读取文档时,文档已更改。原始读取不可重复。如果 只有在作者全部完成编写后编辑人员才可以读取文档,则可以避免该问题。

幻读 : 是指当事务不是独立执行时发生的一种现象,例如第一个事务对一个表中的数据进行了修改,这种修改涉及到表中的全部数据行。 同时,第二个事务也修改这个表中的数据,这种修改是向表中插入一行新数据。那么,以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行,就好象 发生了幻觉一样。例如,一个编辑人员更改作者提交的文档,但当生产部门将其更改内容合并到该文档的主复本时,发现作者已将未编辑的新材料添加到该文档中。 如果在编辑人员和生产部门完成对原始文档的处理之前,任何人都不能将新材料添加到文档中,则可以避免该问题。

Mysql中的事务默认是AUTOCOMMIT模式，这意味着除非显示的开始一个事物，否则他将把每个查询视为一个单独事务自动执行。

SET AUTOCOMMIT =1/ON是开启

SET AUTOCOMMIT =0/OFF是关闭

如果设置为0，用户将一直处于某个事物中，指导用户执行一条COMMIT或ROLLBACK语句，之后，mysql将立即开始一个新事务。

此外，mysql允许通过SET SESSION ISOLATION LEVEL命令设置隔离级别。

如： SET SESSION ISOLATION LEVEL REPEATABLE READ;

架构优化和索引：

优化通常需要权衡取舍：

1：更改某个查询或者架构的一部分可能会对其他部分造成影响。

2：为了加快数据读取而增加的索引会减慢更新的速度。

3：非规范化的架构能加快某些类型的查询，但却会让其他类型的查询变慢。等等

数据类型使用原则：只分配真正需要的空间。

可以使用ENUM代替字符串类型.

ENUM列可以存储65535个不同的字符串。mysql以非常紧凑的方式保存了他们，根据列表中值得数量，mysql会把他们压缩到1到2个字节中。mysql在内部把每个值都保存为整数，用于表示值在列表中的位置，并且还保留了一份查找表来表示证书和字符串的.frm中的映射关系。

ENUM字段在内部是按照顺序进行排序的，而不是按字符串顺序。

ENUM最不好的一面是字符串列表是固定的，并且添加或者删除字符串必须使用ALTER TABLE

日期和事件类型：

DATETIME：

这个类型能保存大范围的值，从1001年到9999年，精度为秒。封装格式为：

YYYYMMDDHHMMSS的整数，与时区无关，使用8字节存储空间。

默认情况下，mysql以一种可排序，清楚的格式显示DATETIME的值，如2008-1-16 22：37：08

TIMESTAMP

保持了1970年1月1日午夜以来的秒数，与UNIX时间戳相同。只使用了4字节的存储空间，所以它的范围比DATETIME小得多，他只能表示从1970到2038年。依赖于时区。

如果插入的行没有定义TIMESTAMP的值，mysql也会自动更新它。默认是not null。

SET和ENUM类型一般不适合做标识列，因为他们虽然以整数存储，但是比较的时候会转换为字符串类型。

存储引擎使用了不同的方式把索引保存到磁盘上，它们会影响性能。

InnoDB不会压缩索引，因为他不能把压缩索引用于某些优化，InnoDB按照主键值引用行。

B-Tree索引能很好的用于全键值，减值范围，或者建前缀查询。他们只有在查找使用了索引的最左前缀的时候才有用。

从BTree/B+Tree查找的角度

上面提到BTree/B+Tree中一次检索最多需要h-1次I/O，这里肯定是需要从根节点开始搜索BTree/B+Tree，根节点就在内存中，下面先贴出一个高度为3的B+Tree

当需要查找key为20的节点时：

先从内存中对更节点进行搜索找到下一个节点的指针，由于一个节点上的数据是有序的，因此可以用二分查找，定位到第二层坐左边的节点。

将该页的数据从磁盘上读取到内存，再按照二分查找定位到最后一层中间的节点。

从磁盘上读取该节点到内存中，找到20，取出数据

这一过程进行了2次I/O操作，3次内存查找操作，并且二分查找的时间复杂度为O(log2n)。

InnoDB引擎中一棵B+Tree可以存放多少行数据？

InnoDB引擎中的B+Tree结构在MySQL索引详解一文中已经讲解过，这里我再次把图贴出来为了方便看。

B+Tree中，一个节点设计为一个页的大小16K，假设一行记录的数据大小为nK，B+Tree的高度为t（t的值一般取1~3），主键的长度为p字节，指针大小在InnoDB源码中设置为6字节

先计算非叶子节点能存放多少个指针和键值的组合？

指针和键值的组合占p+6字节，因此答案是[16\*1024]/（p+6）个，这个结果记为A

再把数的高度带入计算，得到InnoDB引擎中一棵B+Tree可以存放多少行数据？

答案是： (A ^t) \*（16/n）

举例

一行记录的大小是1k，则一页能存16条记录，B+Tree高度是3，主键id设为bigint（8字节），一个节点最大能存16\*1024=16384字节的数据，指针在InnoDB中是6字节，则主键+指针的大小是14字节，一个非叶子节点能存16384/14=1170个主键+指针的组合，因此对于高度为3的B+Tree，能存1170 \* 1170 \* 16=21902400条记录。

————————————————

版权声明：本文为CSDN博主「IT\_10-」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/IT\_10/article/details/95530577

**非聚集索引和聚集索引的区别**在于， 通过聚集索引可以查到需要查找的数据， 而通过非聚集索引可以查到记录对应的主键值 ， 再使用主键的值通过聚集索引查找到需要的数据

**不管以任何方式查询表， 最终都会利用主键通过聚集索引来定位到数据， 聚集索引（主键）是通往真实数据所在的唯一路径。**

**索引分类：**

[**https://www.cnblogs.com/s42-/p/13596212.html**](https://www.cnblogs.com/s42-/p/13596212.html)

我们这里创建一个用户表，表中有字段name，并且在name上有索引

1 create table t\_user (

2 id bigint(**20**) not null auto\_increment ,

3 name varchar(**255**) not null,

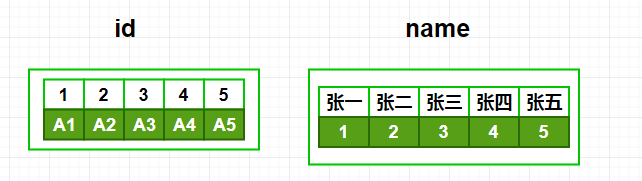
4 primary key (id),

5 index index\_name (name) using btree)

6 engine=innodb

7 default character set=utf8 collate=utf8\_general\_ci

两棵树的示例示意图如下



1.主键索引：主键索引的叶子节点保存着主键即对应行的全部数据。在InnoDB里，主键索引也被称为聚簇索引

2.二级索引（非主键索引）： 二级索引树中的叶子结点保存着索引值和主键值，当使用二级索引进行查询时，需要进行回表操作。在InnoDB里，非主键索引也被称为二级索引

如何区分主键索引和普通索引的查询

1 select \* from t\_user where id=**1**

2 //即主键查询方式，则只需要搜索id这棵B+树

1 select \* from t\_user where name="张三"

2 //即普通索引查询方式，则需要先搜索name索引树，得到id的值为3，再到id索引树搜索一次。这个过程称为回表

所以基于二级索引（非主键索引）的查询需要多扫描一棵索引树。因此，我们在应用中应该尽量使用主键查询

3.覆盖索引

1 A: select id from user\_table where name= '张三'

2 B: select password from user\_table where name= '张三'

语句A： 因为 name索引树 的叶子结点上保存有 name和id的值 ，所以通过 name索引树 查找到id后，因此可以直接提供查询结果，不需要回表，也就是说，在这个查询里面，索引name 已经 “覆盖了” 我们的查询需求，我们称为 覆盖索引

语句B： name索引树 上 找到 name=‘张三’ 对应的主键id, 通过回表在主键索引树上找到满足条件的数据

因此我们可以得知，当sql语句的所求查询字段（select列）和查询条件字段（where子句）全都包含在一个索引中（联合索引），可以直接使用索引查询而不需要回表。这就是覆盖索引。

**事务隔离级别有四种，mysql默认使用的是可重复读，mysql是怎么实现可重复读的？为什么会出现幻读？是否解决了幻读的问题？**

**一、事务的隔离级别**

**Read Uncommitted（未提交读）**

**在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。读取未提交的数据，也被称之为脏读（Dirty Read）。该级别用的很少。**

**Read Committed（提交读）**

**这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变，换句话说就是事务提交之前对其余事务不可见。这种隔离级别也支持不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select查询可能返回不同结果。**

**Repeatable Read（可重复读）**

**这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题（mysql彻底解决了幻读问题？请往下看）。**

**Serializable（可串行化）**

**这是最高的隔离级别，它强制事务都是串行执行的，使之不可能相互冲突，从而解决幻读问题。换言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。**

**事务隔离级别 脏读 不可重复读 幻读**

**读未提交（read-uncommitted） 是 是 是**

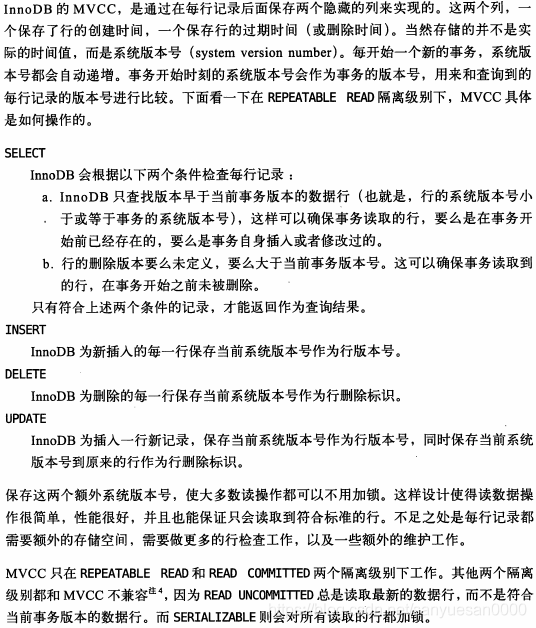
**不可重复读（read-committed） 否 是 是**

**可重复读（repeatable-read） 否 否 是**

**串行化（serializable） 否 否 否**

**在MySQL的众多存储引擎中，只有InnoDB支持事务，所有这里说的事务隔离级别指的是InnoDB下的事务隔离级别。**

**二、mysql怎么实现的可重复读**

**MVCC多版本并发控制(Multi-Version Concurrency Control)是MySQL中基于乐观锁理论实现隔离级别的方式，用于实现读已提交和可重复读取隔离级别。**

**在《高性能MySQL》中对MVCC的解释如下**

**举例说明MVCC的实现**

**新建一张表test\_zq如下**

**id test\_id DB\_TRX\_ID DB\_ROLL\_PT**

**MVCC逻辑流程-插入**

**在插入数据的时候，假设系统的全局事务ID从1开始，以下SQL语句执行分析参考注释信息：**

**begin;-- 获取到全局事务ID**

**insert into `test\_zq` (`id`, `test\_id`) values('5','68');**

**insert into `test\_zq` (`id`, `test\_id`) values('6','78');**

**commit;-- 提交事务**

**复制代码**

**当执行完以上SQL语句之后，表格中的内容会变成：**

**id test\_id DB\_TRX\_ID DB\_ROLL\_PT**

**5 68 1 NULL**

**6 78 1 NULL**

**可以看到，插入的过程中会把全局事务ID记录到列 DB\_TRX\_ID 中去**

**MVCC逻辑流程-删除**

**对上述表格做删除逻辑，执行以下SQL语句（假设获取到的事务逻辑ID为 3）**

**begin；--获得全局事务ID = 3**

**delete test\_zq where id = 6;**

**commit;**

**复制代码**

**执行完上述SQL之后数据并没有被真正删除，而是对删除版本号做改变，如下所示：**

**id test\_id DB\_TRX\_ID DB\_ROLL\_PT**

**5 68 1 NULL**

**6 78 1 3**

**MVCC逻辑流程-修改**

**修改逻辑和删除逻辑有点相似，修改数据的时候 会先复制一条当前记录行数据，同事标记这条数据的数据行版本号为当前是事务版本号，最后把原来的数据行的删除版本号标记为当前是事务。**

**执行以下SQL语句：**

**begin;-- 获取全局系统事务ID 假设为 10**

**update test\_zq set test\_id = 22 where id = 5;**

**commit;**

**复制代码**

**执行后表格实际数据应该是：**

**id test\_id DB\_TRX\_ID DB\_ROLL\_PT**

**5 68 1 10**

**6 78 1 3**

**5 22 10 NULL**

**MVCC逻辑流程-查询**

**此时，数据查询规则如下：**

**查找数据行版本号早于当前事务版本号的数据行记录**

**也就是说，数据行的版本号要小于或等于当前是事务的系统版本号，这样也就确保了读取到的数据是当前事务开始前已经存在的数据，或者是自身事务改变过的数据**

**查找删除版本号要么为NULL，要么大于当前事务版本号的记录**

**这样确保查询出来的数据行记录在事务开启之前没有被删除**

**根据上述规则，我们继续以上张表格为例，对此做查询操作**

**begin;-- 假设拿到的系统事务ID为 12**

**select \* from test\_zq;**

**commit;**

**复制代码**

**执行结果应该是：**

**id test\_id DB\_TRX\_ID DB\_ROLL\_PT**

**6 22 10 NULL**

**这样，同一个事务中，就实现了可重复读。**

**三、幻读**

**什么是幻读，如下：**

**InnoDB实现的RR通过mvcc机制避免了这种幻读现象。**

**另一种幻读：**

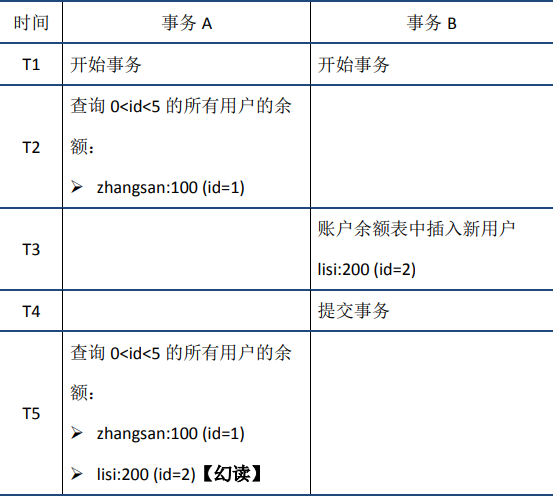
**姑且把左边的事务命名为事务A，右边的命名为事务B。**

**事务B执行后，在事务A中查询没有查到B添加的数据行，这就是可重复读。**

**但是，在事务A执行了update后，再查询时就查到了事务A中添加的数据，这就是幻读。**

**这种结果告诉我们其实在MySQL可重复读的隔离级别中并不是完全解决了幻读的问题，而是解决了读数据情况下的幻读问题。而对于修改的操作依旧存在幻读问题，就是说MVCC对于幻读的解决是不彻底的。**

**原以为这也是一种幻读，但经过多次研究资料，这只是对数据修改的操作(update、insert、delete)当前读产生的结果，他其实不是幻读。**

**快照读和当前读**

**出现了上面的情况我们需要知道为什么会出现这种情况。在查阅了一些资料后发现在RR级别中，通过MVCC机制，虽然让数据变得可重复读，但我们读到的数据可能是历史数据，不是数据库最新的数据。这种读取历史数据的方式，我们叫它快照读 (snapshot read)，而读取数据库最新版本数据的方式，叫当前读 (current read)。**

**select 快照读**

**当执行select操作是innodb默认会执行快照读，会记录下这次select后的结果，之后select 的时候就会返回这次快照的数据，即使其他事务提交了不会影响当前select的数据，这就实现了可重复读了。快照的生成当在第一次执行select的时候，也就是说假设当A开启了事务，然后没有执行任何操作，这时候B insert了一条数据然后commit,这时候A执行 select，那么返回的数据中就会有B添加的那条数据。之后无论再有其他事务commit都没有关系，因为快照已经生成了，后面的select都是根据快照来的。**

**当前读**

**对于会对数据修改的操作(update、insert、delete)都是采用当前读的模式。在执行这几个操作时会读取最新的版本号记录，写操作后把版本号改为了当前事务的版本号，所以即使是别的事务提交的数据也可以查询到。假设要update一条记录，但是在另一个事务中已经delete掉这条数据并且commit了，如果update就会产生冲突，所以在update的时候需要知道最新的数据。也正是因为这样所以才导致幻读。**

**四、如何解决幻读**

**在快照读情况下，mysql通过mvcc来避免幻读。**

**在当前读情况下，mysql通过X锁或next-key来避免其他事务修改:**

**使用串行化读的隔离级别**

**(update、delete)当where条件为主键时，通过对主键索引加record locks(索引加锁/行锁)处理幻读。**

**(update、delete)当where条件为非主键索引时，通过next-key锁处理。next-key是record locks(索引加锁/行锁) 和 gap locks(间隙锁，每次锁住的不光是需要使用的数据，还会锁住这些数据附近的数据)的结合。**

**Next-Key Lock即在事务中select时使用如下方法加锁，这样在另一个事务对范围内的数据进行修改时就会阻塞（为什么有共享锁会阻塞？不能在有共享锁的记录上加X锁）：**

**select \* from table where id<6 lock in share mode;--共享锁**