mysql-5.1版本之前默认引擎是MyISAM，之后是innoDB

数据库默认隔离级别： mysql ---repeatable,oracle,sql server ---read commited

为什么Mysql不选择读已提交(Read Commited)作为默认隔离级别，而选择可重复读(Repeatable Read)作为默认的隔离级别呢？

Why?Why?Why?

这个是有历史原因的，当然要从我们的主从复制开始讲起了！

主从复制，是基于什么复制的?

是基于binlog复制的！这里不想去搬binlog的概念了，就简单理解为binlog是一个记录数据库更改的文件吧～

binlog有几种格式?

OK，三种，分别是

- statement:记录的是修改SQL语句
- row：记录的是每行实际数据的变更
- mixed：statement和row模式的混合

那Mysql在5.0这个版本以前，binlog只支持 `STATEMENT` 这种格式！而这种格式在读已提交(Read Commited)这个隔离级别下主从复制是有bug的，因此Mysql将可重复读(Repeatable Read)作为默认的隔离级别！

此时在主(master)上执行下列语句

```
select * from test;
```

输出如下

```
+---+
| b |
+---+
| 3 |
+---+
1 row in set
```

但是，你在此时在从(slave)上执行该语句，得出输出如下

```
Empty set
```

这样，你就出现了主从不一致性的问题！原因其实很简单，就是在master上执行的顺序为先删后插！而此时binlog为STATEMENT格式，它记录的顺序为先插后删！从(slave)同步的是binglog，因此从机执行的顺序和主机不一致！就会出现主从不一致！

解决方案：

1）隔离级别设为可重复读(Repeatable Read),在该隔离级别下引入间隙锁。

2）将binglog的格式修改为row格式，此时是基于行的复制，自然就不会出现sql执行顺序不一样的问题！奈何这个格式在mysql5.1版本开始才引入。因此由于历史原因，mysql将默认的隔离级别设为可重复读(Repeatable Read)，保证主从复制不出问题！

究竟隔离级别是用读已经提交呢还是可重复读？

接下来对这两种级别进行对比，讲讲我们为什么选读已提交(Read Commited)作为事务隔离级别！

1）*在RR隔离级别下，存在间隙锁，导致出现死锁的几率比RC大的多！*

2）在RR隔离级别下，条件列未命中索引会锁表！而在RC隔离级别下，只锁行

此时执行语句

3）*在RC隔离级别下，半一致性读(semi-consistent)特性增加了update操作的并发性！*

详解mysql的默认隔离级别

知识点总结

1.数据库默认隔离级别：mysql ---repeatable,oracle,sql server ---read commited

2.mysql binlog的格式三种：statement,row,mixed

3.为什么mysql用的是repeatable而不是read committed:在 5.0之前只有statement一种格式，而主从复制存在了大量的不一致，故选用repeatable

4.为什么默认的隔离级别都会选用read commited 原因有二：repeatable存在间隙 锁会使死锁的概率增大，*在RR隔离级别下，条件列未命中索引会锁表！而在RC隔离级别下，只锁行*

2.在RC级用别下，主从复制用什么binlog格式： row格式，是基于行的复制！

https://www.cnblogs.com/cxy2020/p/13797727.html

美团技术团队

https://tech.meituan.com/2014/06/30/mysql-index.html

数据库的DDL、DML和DCL的区别与理解

- DML（data manipulation language）： 它们是SELECT、UPDATE、INSERT、DELETE，就象它的名字一样，这4条命令是用来对数据库里的数据进行操作的语言
- DDL（data definition language）： DDL比DML要多，主要的命令有CREATE、ALTER、DROP等，DDL主要是用在定义或改变表（TABLE）的结构，数据类型，表之间的链接和约束等初始化工作上，他们大多在建立表时使用
- DCL（Data Control Language）： 是数据库控制功能。是用来设置或更改数据库用户或角色权限的语句，包括（grant,deny,revoke等）语句。在默认状态下，只有sysadmin,dbcreator,db_owner或db_securityadmin等人员才有权力执行DCL
- TCL - Transaction Control Language: 事务控制语言，COMMIT - 保存已完成的工作，SAVEPOINT - 在事务中设置保存点，可以回滚到此处，ROLLBACK - 回滚，SET TRANSACTION - 改变事务选项

Sql语句优化

1）在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致（比如对字符串的模糊查询就要最好保证最左边的字符是确定的，否则不能使用联合索引）

2）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

```
select id from t where num is null
```

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

```
select id from t where num=0
```

3）应尽量避免在 where 子句中使用!=或<>操作符，否则将导致引擎放弃使用索引而进行全表扫描

4）应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描。可以将sql语句分成多个部分然后使用Union拼接起来

```
select id from t where num=10 or num=20
```

可以这样查询：

```
select id from t where num=10
union all
select id from t where num=20
```

5）in 和 not in 也要慎用，否则会导致全表扫描。能用between的地方就尽量不要用in

6）应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：

```
select id from t where num/2=100
```

应改为:

```
select id from t where num=100*2
```

7）应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：

```
select id from t where substring(name,1,3)='abc'--name以abc开头的id
```

应改为:

```
select id from t where name like 'abc%'
```

8）不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。

b树和b+树区别

- B树每个节点都存储数据，所有节点组成这棵树。B+树只有叶子节点存储数据（B+数中有两个头指针：一个指向根节点，另一个指向关键字最小的叶节点），叶子节点包含了这棵树的所有数据，所有的叶子结点使用链表相连，便于区间查找和遍历，所有非叶节点起到索引作用。
- B树中叶节点包含的关键字和其他节点包含的关键字是不重复的，B+树的索引项只包含对应子树的最大关键字和指向该子树的指针，不含有该关键字对应记录的存储地址。
- B树中每个节点（非根节点）关键字个数的范围为[m/2(向上取整)-1,m-1](根节点为[1,m-1])，并且具有n个关键字的节点包含（n+1）棵子树。B+树中每个节点（非根节点）关键字个数的范围为[m/2(向上取整),m](根节点为[1,m])，具有n个关键字的节点包含（n）棵子树。
- B+树中查找，无论查找是否成功，每次都是一条从根节点到叶节点的路径。

B树的优点

1.B树的每一个节点都包含key和value，因此经常访问的元素可能离根节点更近，因此访问也更迅速。

B+树的优点

1. 所有的叶子结点使用链表相连，便于区间查找和遍历。B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。
2. b+树的中间节点不保存数据，能容纳更多节点元素。

B树和B+树的共同优点

考虑磁盘IO的影响，它相对于内存来说是很慢的。数据库索引是存储在磁盘上的，当数据量大时，就不能把整个索引全部加载到内存了，只能逐一加载每一个磁盘页（对应索引树的节点）。所以我们要减少IO次数，对于树来说，IO次数就是树的高度，而“矮胖”就是b树的特征之一，m的大小取决于磁盘页的大小。

1000万条数据在B+树中有几层?

假设一行数据的大小是1k，那么一个页可以存放16行这样的数据。

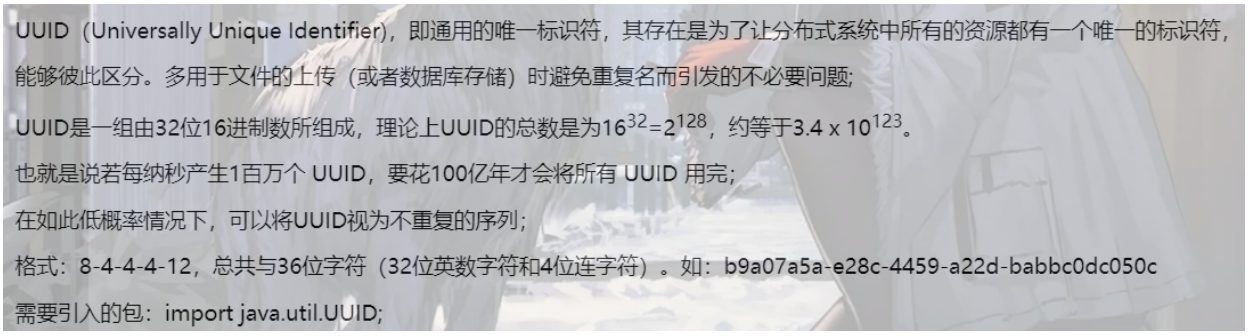
其实这也很好算，我们假设主键ID为bigint类型，长度为8字节，而指针大小在InnoDB源码中设置为6字节，这样一共14字节，我们一个页中能存放多少这样的单元，其实就代表有多少指针，即（InnoDB最小存储单元---页的大小）16KB（16*1024=16384 byte）16384/14=1170（索引个数）。那么可以算出一棵高度为2的B+树，能存放1170*16=18720条这样的数据记录。

一个高度为 3 的 B+ 树大概可以存放 1170 × 1170 × 16 = 21902400 行数据，已经是千万级别的数据量了。

主键可以是uuid吗?

可以

uuid简介：



优点：

能够保证独立性，程序可以在不同的数据库间迁移，效果不受影响。

保证生成的ID不仅是表独立的，而且是库独立的，这点在你想切分数据库的时候尤为重要。

缺点：

比较占地方，和INT类型相比，存储一个UUID要花费更多的空间。索引查询时也会因为空间更大而降低查询效率

使用UUID后，URL显得冗长，不够友好。

有三张表

学生信息表:s(sno,sname,ssex,sdep,sclass)

课程信息表:c(cno,cname,teacher)

成绩表 :sc(sno,cno,grade)

怎样查询每门课分数最高的学生

查询结果中需显示三个字段： 学生姓名(sname),课程名(cname),成绩(grade)

```
select sname,cname,degree from students t1,courses t2,scores t3 WHERE
t1.sno=t3.sno and t2.cno=t3.cno GROUP BY t2.cno HAVING degree=max(degree)
```

为什么用 b+ 树而不是 b树

1）B+树空间利用率更高，因为B+树只在叶子结点存放数据，其他结点全是索引，所以能容纳更多结点数据，所以使整棵树更矮，减少了I/O次数，磁盘读写代价更低

2）所有的叶子结点使用链表相连，便于区间查找和遍历。B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

mysql 的 MVCC 以及是否解决幻读 （ReadView机制 undo log 版本链）

不可重复读：在一个事务内，连续两次查询同一条数据，查到的结果前后不一样

幻读：幻读特指后面的查询比前面的查询的记录条数多，看到了前面没看到的数据，就像产生幻觉一样，因此称之为幻读

在事务开启的时候，会基于当前系统中数据库的数据，为每个事务生成一个快照，也叫做 ReadView，后面这个事务所有的读操作都是基于这个 ReadView 来读取数据，这种读称之为快照读。「我们在实际的工作中，所使用的 SQL 查询语句基本都是快照读。」

通过前面介绍的 undo log 版本链，我们知道，每行数据可能会有多个版本，如果每次读取时，「我们都强制性的读取最新版本的数据，这种读称之为当前读，也就是读取最新的数据」。什么样的 SQL 查询语句叫做当前读呢？例如在 select 语句后面加上「for update 或者 lock in share mode」等。

「需要说明的是，在 MySQL 可重复读隔离级别下，幻读问题确实不存在。但是 MVCC 机制解决的是快照读的幻读问题，并不能解决当前读的幻读问题。当前读的幻读问题是通过间隙锁解决的

「只会在第一次查询的时候生成 ReadView 快照，这一点和读提交隔离级别是最大的区别」。

redo log，bin log，undo log　　https://www.jianshu.com/p/68d5557c65be

为什么RDB 要 fork 子进程而不是线程

主要是出于Redis性能的考虑，(1)Redis RDB持久化机制会阻塞主进程，这样主进程就无法响应客户端请求。(2)我们知道Redis对客户端响应请求的工作模型是单进程和单线程的，如果在主进程内启动一个线程，这样会造成对数据的竞争条件，为了避免使用锁降低性能。基于以上两点这就是为什么Redis通过启动一个进程来执行RDB了。

(1)RDB持久化机制启用方式：

Ⅰ：在Redis的配置文件中开启如下设置

```
save 900 1          #900秒内如果超过1个key被修改，则发起快照保存
save 300 10         #300秒内容如超过10个key被修改，则发起快照保存
save 60 10000
```

Ⅱ:也可以通过手动执行SAVE和BGSAVE命令来执行保存快照到磁盘，SAVE和BGSAVE两个命令都会调用rdbSave函数,但它们调用的方式各有不同：

SAVE 直接调用rdbSave，阻塞Redis主进程看，直到保存完成为止。在主进程阻塞期间，服务器不能处理客户端的任何请求。
BGSAVE 则fork 出一个子进程，子进程负责调用rdbSave ，并在保存完成之后向主进程发送信号，通知保存已完成。因为rdbSave 在子进程被调用，所以Redis 服务器在BGSAVE 执行期间仍然可以继续处理客户端的请求。

sorted set（redis中的有序集合）的底层结构

zset底层的存储结构包括ziplist或skiplist，在同时满足以下两个条件的时候使用ziplist，其他时候使用skiplist，两个条件如下：

- 有序集合保存的元素数量小于128个
- 有序集合保存的所有元素的长度小于64字节

当ziplist作为zset的底层存储结构时候，每个集合元素使用两个紧挨在一起的压缩列表节点来保存，第一个节点保存元素的成员，第二个元素保存元素的分值。

当skiplist作为zset的底层存储结构的时候，使用skiplist按序保存元素及分值，使用dict来保存元素和分值的映射关系。

事务的四大特性（acid）

1. 原子性 （atomicity）:强调事务的不可分割.事务中的操作要么都发生，要么都不发生
2. 一致性 （consistency）:事务的执行的前后数据的完整性保持一致.
3. 隔离性 （isolation）:一个事务执行的过程中,不应该受到其他事务的干扰
4. 持久性（durability） :事务一旦结束,数据就持久到数据库

事务运行模式（3种）

1. 自动提交事务：默认事务管理模式。如果一个语句成功地完成，则提交该语句；如果遇到错误，则回滚该语句。
2. 显式事务：以BEGIN TRANSACTION显式开始，以COMMIT或ROLLBACK显式结束。
3. 隐性事务：当连接以此模式进行操作时，sql将在提交或回滚当前事务后自动启动新事务。无须描述事务的开始，只需提交或回滚每个事务。它生成连续的事务链。

mysql默认自增ID是从1开始

如何提高MySql插入性能

1）如果数据库中的数据已经很多(几百万条)，那么可以 加大mysql配置中的bulk_insert_buffer_size，这个参数默认为8M

1. bulk_insert_buffer_size=100M

2）改写所有 insert into 语句为 insert delayed into
这个insert delayed不同之处在于：立即返回结果，后台进行插入处理。

3）一次插入多条数据:
insert中插入多条数据，举例:
insert into table values('11','11'),('22','22'),('33','33')...;

4）尽量自己创建事务，在事务中进行插入处理
可以避免mysql在自动提交模式下重复开启事务