**Java 后台数据库相关问题**

1. **mysql的搜索引擎？**

数据库管理系统使用数据引擎进行创建，查询，更新和删除数据。查询引擎的命令：show table status

Mysql不需要在整个服务器中使用同一种引擎，可以根据具体的场景自己设计。

1. myISAM

它支持事物也不支持外键，但是访问速度快，对事物的完整性没有要求，以select和insert为主的都可以用这引擎创建这个表。它的数据文件和索引文件放置不同的目录，平均分配IO，获取更快的速度。count（\*）的时候要扫面一遍整个表来计算有多少行，写操作比较好

1. INNODB

支持事务，不保存表的具体行数，count（\*）的时候要扫面一遍整个表来计算有多少行，写操作比较好.

1. MeMory

memory使用存在内存中的内容来创建表。每个MEMORY表实际对应一个磁盘文件，格式是.frm。MEMORY类型的表访问非常快，因为它到数据是放在内存中的，并且默认使用HASH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。

1. **Mysql为什么要使用自增列作为主键？**

1）如果我们定义了主键(PRIMARY KEY)，那么InnoDB会选择主键作为聚集索引、如果没有显式定义主键，则InnoDB会选择第一个不包含有NULL值的唯一索引作为主键索引、如果也没有这样的唯一索引，则InnoDB会选择内置6字节长的ROWID作为隐含的聚集索引(ROWID随着行记录的写入而主键递增，这个ROWID不像ORACLE的ROWID那样可引用，是隐含的)。

2）数据记录本身被存于主索引（一颗B+Tree）的叶子节点上。这就要求同一个叶子节点内（大小为一个内存页或磁盘页）的各条数据记录按主键顺序存放，因此每当有一条新的记录插入时，MySQL会根据其主键将其插入适当的节点和位置，如果页面达到装载因子（InnoDB默认为15/16），则开辟一个新的页（节点）

3）如果表使用自增主键，那么每次插入新的记录，记录就会顺序添加到当前索引节点的后续位置，当一页写满，就会自动开辟一个新的页

1. 如果使用非自增主键，由于每次插入主键的值近似于随机，因此每次新纪录都要被插到现有索引页得中间某个位置，此时MySQL不得不为了将新记录插到合适位置而移动数据，甚至目标页面可能已经被回写到磁盘上而从缓存中清掉，此时又要从磁盘上读回来，这增加了很多开销，同时频繁的移动、分页操作造成了大量的碎片，得到了不够紧凑的索引结构，后续不得不通过OPTIMIZE TABLE来重建表并优化填充页面。
2. **数据库索引**

索引是帮助mysql高效获取数据的数据结构。数据库的索引是有序的，在有序的情况下查找数据是不需要遍历的；在极端的情况下数据索引的查询效率为二分查询，**趋近与log（2N）**。

1. 索引的类型

1）FULLTEXT

全文索引，目前只有MyISAM引擎支持。目前只有MyISAM引擎支持。其可以在CREATE TABLE ，ALTER TABLE ，CREATE INDEX 使用，不过目前只有 CHAR、VARCHAR ，TEXT 列上可以创建全文索引。值得一提的是，在数据量较大时候，现将数据放入一个没有全局索引的表中，然后再用CREATE INDEX创建FULLTEXT索引，要比先为一张表建立FULLTEXT然后再将数据写入的速度快很多。它的出现是为了解决WHERE name LIKE “%word%"这类针对文本的模糊查询效率较低的问题。

1. BTree索引

公司总结上有

1. Hash

**Hash**这个词，可以说，自打我们开始码的那一天起，就开始不停地见到和使用到了。其实，hash就是一种（key=>value）形式的键值对，如数学中的函数映射，允许多个key对应相同的value，但不允许一个key对应多个value。正是由于这个特性，hash很适合做索引，为某一列或几列建立hash索引，就会利用这一列或几列的值通过一定的算法计算出一个hash值，对应一行或几行数据（这里在概念上和函数映射有区别，不要混淆）。在java语言中，每个类都有自己的hashcode()方法，没有显示定义的都继承自object类，该方法使得每一个对象都是唯一的，在进行对象间equal比较，和序列化传输中起到了很重要的作用。hash的生成方法有很多种，足可以保证hash码的唯一性，例如在MongoDB中，每一个document都有系统为其生成的唯一的objectID（包含时间戳，主机散列值，进程PID，和自增ID）也是一种hash的表现。额，我好像扯远了-\_-!

由于hash索引可以一次定位，不需要像树形索引那样逐层查找,因此具有极高的效率。那为什么还需要其他的树形索引呢？

在这里愚安就不自己总结了。引用下园子里其他大神的文章：来自 14的路 的[MySQL的btree索引和hash索引的区别](http://www.cnblogs.com/vicenteforever/articles/1789613.html)

（1）Hash 索引仅仅能满足"=","IN"和"<=>"查询，不能使用范围查询。   
由于 Hash 索引比较的是进行 Hash 运算之后的 Hash 值，所以它只能用于等值的过滤，不能用于基于范围的过滤，因为经过相应的 Hash 算法处理之后的 Hash 值的大小关系，并不能保证和Hash运算前完全一样。   
（2）Hash 索引无法被用来避免数据的排序操作。   
由于 Hash 索引中存放的是经过 Hash 计算之后的 Hash 值，而且Hash值的大小关系并不一定和 Hash 运算前的键值完全一样，所以数据库无法利用索引的数据来避免任何排序运算；   
（3）Hash 索引不能利用部分索引键查询。   
对于组合索引，Hash 索引在计算 Hash 值的时候是组合索引键合并后再一起计算 Hash 值，而不是单独计算 Hash 值，所以通过组合索引的前面一个或几个索引键进行查询的时候，Hash 索引也无法被利用。   
（4）Hash 索引在任何时候都不能避免表扫描。   
前面已经知道，Hash 索引是将索引键通过 Hash 运算之后，将 Hash运算结果的 Hash 值和所对应的行指针信息存放于一个 Hash 表中，由于不同索引键存在相同 Hash 值，所以即使取满足某个 Hash 键值的数据的记录条数，也无法从 Hash 索引中直接完成查询，还是要通过访问表中的实际数据进行相应的比较，并得到相应的结果。   
（5）Hash 索引遇到大量Hash值相等的情况后性能并不一定就会比B-Tree索引高。   
对于选择性比较低的索引键，如果创建 Hash 索引，那么将会存在大量记录指针信息存于同一个 Hash 值相关联。这样要定位某一条记录时就会非常麻烦，会浪费多次表数据的访问，而造成整体性能低下。

1. mysql的联合索引

联合索引是两个或更多个列上的索引。对于联合索引:Mysql从左到右的使用索引中的字段，一个查询可以只使用索引中的一部份，但只能是最左侧部分。例如索引是key index (a,b,c). 可以支持a 、 a,b 、 a,b,c 3种组合进行查找，但不支持 b,c进行查找 .当最左侧字段是常量引用时，索引就十分有效。

利用索引中的附加列，您可以缩小搜索的范围，但使用一个具有两列的索引 不同于使用两个单独的索引。复合索引的结构与电话簿类似，人名由姓和名构成，电话簿首先按姓氏对进行排序，然后按名字对有相同姓氏的人进行排序。如果您知 道姓，电话簿将非常有用；如果您知道姓和名，电话簿则更为有用，但如果您只知道名不姓，电话簿将没有用处。

1. 什么情况下应不建或少建索引

1）表记录太少

2）经常插入、删除、修改的表

3）数据重复且分布平均的表字段，假如一个表有10万行记录，有一个字段A只有T和F两种值，且每个值的分布概率大约为50%，那么对这种表A字段建索引一般不会提高数据库的查询速度。

4）经常和主字段一块查询但主字段索引值比较多的表字段

1. mysql分区
2. 什么是表分区？

表分区，是指根据一定规则，将数据库中的一张表分解成多个更小的，容易管理的部分。从逻辑上看，只有一张表，但是底层却是由多个物理分区组成。

表分区与分表的区别

分表：指的是通过一定规则，将一张表分解成多张不同的表。比如将用户订单记录根据时间成多个表。

分表与分区的区别在于：分区从逻辑上来讲只有一张表，而分表则是将一张表分解成多张表。

表分区有什么好处？

分区表的数据可以分布在不同的物理设备上，从而高效地利用多个硬件设备。 2. 和单个磁盘或者文件系统相比，可以存储更多数据

优化查询。在where语句中包含分区条件时，可以只扫描一个或多个分区表来提高查询效率；涉及sum和count语句时，也可以在多个分区上并行处理，最后汇总结果。

分区表更容易维护。例如：想批量删除大量数据可以清除整个分区。

可以使用分区表来避免某些特殊的瓶颈，例如InnoDB的单个索引的互斥访问，ext3问价你系统的inode锁竞争等。

分区表的限制因素

一个表最多只能有1024个分区

MySQL5.1中，分区表达式必须是整数，或者返回整数的表达式。在MySQL5.5中提供了非整数表达式分区的支持。

如果分区字段中有主键或者唯一索引的列，那么多有主键列和唯一索引列都必须包含进来。即：分区字段要么不包含主键或者索引列，要么包含全部主键和索引列。

分区表中无法使用外键约束

MySQL的分区适用于一个表的所有数据和索引，不能只对表数据分区而不对索引分区，也不能只对索引分区而不对表分区，也不能只对表的一部分数据分区。

如何判断当前MySQL是否支持分区？

命令：show variables like '%partition%' 运行结果:

1. MySQL支持的分区类型有哪些？

RANGE分区： 这种模式允许将数据划分不同范围。例如可以将一个表通过年份划分成若干个分区

LIST分区： 这种模式允许系统通过预定义的列表的值来对数据进行分割。按照List中的值分区，与RANGE的区别是，range分区的区间范围值是连续的。

HASH分区 ：这中模式允许通过对表的一个或多个列的Hash Key进行计算，最后通过这个Hash码不同数值对应的数据区域进行分区。例如可以建立一个对表主键进行分区的表。

KEY分区 ：上面Hash模式的一种延伸，这里的Hash Key是MySQL系统产生的。

1. 四种隔离级别

Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

Read committed (读已提交)：可避免脏读的发生。

Read uncommitted (读未提交)：最低级别，任何情况都无法保证

事物的基本要素（ACID）：

1、原子性（Atomicity）：事务开始后所有操作，要么全部做完，要么全部不做，不可能停滞在中间环节。事务执行过程中出错，会回滚到事务开始前的状态，所有的操作就像没有发生一样。也就是说事务是一个不可分割的整体，就像化学中学过的原子，是物质构成的基本单位。

2、一致性（Consistency）：事务开始前和结束后，数据库的完整性约束没有被破坏 。比如A向B转账，不可能A扣了钱，B却没收到。

3、隔离性（Isolation）：同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。比如A正在从一张银行卡中取钱，在A取钱的过程结束前，B不能向这张卡转账。

4、持久性（Durability）：事务完成后，事务对数据库的所有更新将被保存到数据库，不能回滚。

事务的并发问题

　　1、脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据

　　2、不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果 不一致。

　　3、幻读：系统管理员A将数据库中所有学生的成绩从具体分数改为ABCDE等级，但是系统管理员B就在这个时候插入了一条具体分数的记录，当系统管理员A改结束后发现还有一条记录没有改过来，就好像发生了幻觉一样，这就叫幻读。

　　小结：不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除。解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表



举例子说明：

1. 读未提交

set session transaction isolation level read uncommited //设置事物隔离级别

Start transaction //开启事物

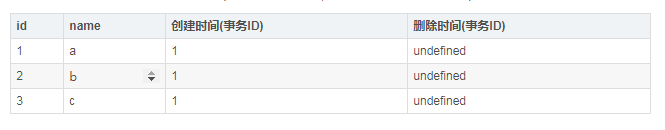
在这个时候一个客户端A读取一个表C，另一个客户端B对表C更新数据但是还没有提交，但是表A此时却能读取出数据，若表B对事物回滚，那个表A此时读取的就是脏数据。

**MySQl的MVCC机制？**

MVCC：多版本并发控制。一般情况下一般情况下，事务性储存引擎不是只使用表锁，行加锁的处理数据，而是同时结合了MVCC机制，以处理更多的并发问题。当然Mysql中Innodb引擎才支持。MVCC最大的好处：读不加锁，读写不冲突。在读多写少的OLTP应用中，读写不冲突是非常重要的，极大的增加了系统的并发性能

**Mysql如何实现MVCC**

InnoDB的MVCC,是通过在每行记录后面保存两个隐藏的列来实现的,这两个列，分别保存了这个行的创建时间，一个保存的是行的删除时间。这里存储的并不是实际的时间值,而是系统版本号(可以理解为事务的ID)，每开始一个新的事务，系统版本号就会自动递增，事务开始时刻的系统版本号会作为事务的ID.



SELECT

InnoDB会根据以下两个条件检查每行记录:

a.InnoDB只会查找版本早于当前事务版本(时间)的数据行(也就是,行的系统版本号小于或等于事务的系统版本号)，这样可以确保事务读取的行，要么是在事务开始前已经存在的，要么是事务自身插入或者修改过的.

b.行的删除版本(时间)要么未定义,要么大于当前事务版本号,这可以确保事务读取到的行，在事务开始之前未被删除.

只有a,b同时满足的记录，才能返回作为查询结果.

DELETE

InnoDB会为删除的每一行保存当前系统的版本号(事务的ID)作为删除标识. 即更改删除时间字段。

UPDATE

InnoDB执行UPDATE，实际上是新插入了一行记录，并保存其创建时间为当前事务的ID，同时保存当前事务ID到要UPDATE的行的删除时间.

**在MVCC并发控制中，读操作可以分成两类：**

快照读 (snapshot read)：读取的是记录的可见版本 (有可能是历史版本)，不用加锁（共享读锁s锁也不加，所以不会阻塞其他事务的写）。

当前读 (current read)：读取的是记录的最新版本，并且，当前读返回的记录，都会加上锁，保证其他事务不会再并发修改这条记录。

**MySql的表级锁定？**

Mysql有三种级别的锁定：表级锁定、页级锁定、行级锁定。

每次锁定一张表的机制就是表级别的锁定，它是MySQL各存储引擎中粒度最大的锁定机制。

1. 优点

实现逻辑简单，开销小。

获取锁和释放锁的速度快。

由于表级锁一次会将整个表锁定，所以能很好的避免死锁问题。

2. 缺点

由于锁粒度最大，因此出现争用被锁定资源的概率也会最高，致使并发度十分低下。

* 使用表级锁定的主要有MyISAM，MEMORY，CSV等一些非事务性存储引擎。

表级锁类型：

Mysql表级锁有两种类型：**表共享读锁和表独占写锁。**

锁模式的兼容性：

对MyISAM表的读操作，不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写操作；

对MyISAM表的写操作，则会阻塞其他用户对同一表的读和写操作；

如何加表锁：

显示加写锁：lock table test write

显示加读锁：lock table test read

显示释放锁：unlock tables

在同一个SQL session里，如果已经获取了一个表的锁定，则对没有锁的表不能进行任何操作，否则会报错。MyISAM引擎的默认方式是会给同一个session里的所有表都加上锁的。

查看表级锁的争用情况：show status like ‘table%’;其中Table\_locks\_immediate：产生表级锁定的次数。

**优化表级锁定：**

1. 通过减少查询时间优化锁定时间

尽量减少大的、复杂的Query，将复杂Query分拆成几个小的Query分步执行；

* 尽可能的建立足够高效的索引，让数据检索更迅速；
* 尽量让MyISAM存储引擎的表只存放必要的信息，控制字段类型；
* 利用合适的机会优化MyISAM表数据文件。

1. 设置可并发插入：concurrent\_insert=2

MyISAM存储引擎有一个控制是否打开Concurrent Insert（并发插入）功能的参数选项：concurrent\_insert，取值范围为0，1，2。

concurrent\_insert=0，不允许并发插入。

concurrent\_insert=1，如果MyISAM表中没有空洞（即表的中间没有被删除的行），MyISAM允许在一个线程读表的同时，另一个线程从表尾插入记录。这是MySQL的默认设置；

concurrent\_insert=2，无论MyISAM表中有没有空洞，都允许在表尾并发插入记录；

所以，我们可通过设置concurrent\_insert=2，同时定期在系统空闲时段执行optimize table tableName语句来整理空间碎片，收回因删除记录而没有真正释放的空间，从而提高并发。optimize参考：mysql中OPTIMIZE TABLE的作用及使用

1. 合理设置读写优先级

MyISAM存储引擎默认是写优先级大于读优先级。即使是写请求后到，写锁也会插到读锁请求之前。

但是，有时像修改文章点击数 操作是不那么重要的，我们希望的是读更快，此时我们可以这样：

UPDATE LOW\_PRIORITY article SET click\_num=134 WHERE id = 823

LOW\_PRIORITY使得系统认为update操作优化级比读操作低，如果同时出现读操作和上面的更新操作，则优先执行读操作。

MySQL提供了几个语句调节符，允许你修改它的调度策略：

LOW\_PRIORITY关键字应用于：DELETE、INSERT、LOAD DATA、REPLACE和UPDATE。

HIGH\_PRIORITY关键字应用于：SELECT、INSERT语句。

delayed(延迟)关键字应用于：INSERT、REPLACE语句。

Mysql的行级锁：

每次锁定一行数据的锁机制就是行级别锁定。行级锁定不是MySQL自己实现的锁定方式，而是由其他存储引擎自己所实现的。

优缺点：

1. 优点

由于锁粒度小，争用率低，并发高。

2. 缺点

实现复杂，开销大。

加锁慢、容易出现死锁

使用行级锁定的主要有InnoDB存储引擎，以及MySQL的分布式存储引擎NDBCluster。

行级锁同样分为两种类型：共享锁和排他锁。InnoDB也同样使用了意向锁（表级锁定）的概念，也就有了意向共享锁和意向排他锁这两种。

意向锁的作用就是当一个事务在需要获取资源锁定的时候，如果遇到自己需要的资源已经被排他锁占用的时候，该事务可以需要锁定行的表上面添加一个合适的意向锁。如果自己需要一个共享锁，那么就在表上面添加一个意向共享锁。而如果自己需要的是某行（或者某些行）上面添加一个排他锁的话，则先在表上面添加一个意向排他锁。

mysql 数据库drop，delete，与truncate的区别？

1. delete语句执行的操作是每次从表中删除一行，并且同时将该删除的操作作为事物记录在日志中保存以便进行回滚操作
2. Truncate table一次性将表中的所有数据删除并不把单独的删除操作记录入日志保存，删除行是不能恢复的。并且在删除过程中不会触发与表有关的触发器，删除速度快。
3. 当表执行truncate的后，这个表和索引所占空间就会恢复到初始大小。
4. Delete不会删除表和索引所占用的空间
5. Drop是将整个表删除掉
6. 一般而言，drop > truncate > delete
7. truncate与不带where的delete ：只删除数据，而不删除表的结构（定义）drop语句将删除表的结构被依赖的约束（constrain),触发器（trigger)索引（index);依赖于该表的存储过程/函数将被保留，但其状态会变为：invalid。

Mysql 存储过程与触发器的区别？

触发器与存储过程十分相似，触发器也是SQL语句集，触发器是在一个修改了指定表中的数据时执行的存储过程。通常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。触发器不同于存储过程，触发器主要是通过事件执行触发而被执行的，而存储过程可以通过存储过程名称名字而直接调用。当对某一表进行诸如UPDATE、INSERT、DELETE这些操作时，SQLSERVER就会自动执行触发器所定义的SQL语句，从而确保对数据的处理必须符合这些SQL语句所定义的规则。

Mysql主表与从表，级联操作？

主表（父表）  
在数据库中建立的表格即Table，其中存在主键(primary key)用于与其它表相关联，并且作为在主表中的唯一性标识。  
从表（子表）  
以主表的主键（primary key）值为外键 (Foreign Key)的表，可以通过外键与主表进行关联查询。从表与主表通过外键进行关联查询。

当有外键约束的时候，必须先修改或删除副表中的所有关联数据，才能修改或删除主表！但是，我们希望直接修改或删除主表数据，从而影响副表数据，这时候，我们就用到了级联操作，只需要在副表中添加一句：ON UPDATE CASCADE

Mysql的主从复制与读写分离？

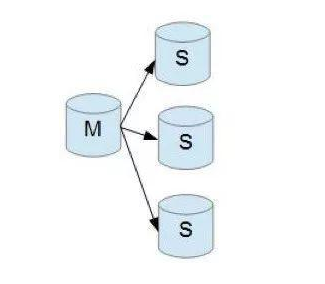
主从复制：数据可以从mysql的一个主节点复制到一个或多个从节点。Mysql默认的是异步的复制方式，这样从节点不用一直访问主节点来更新自己的数据，数据的更新可以在远程的连接上进行。

读写分离：在实际开发中可能遇到某个sql语句需要锁表，导致暂时不能使用读的服务，使用主从复制，让主库负责写，从库负责读。

Mysql的主从形式：

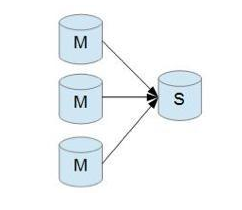
一主一从：

一主多从：



可以实现读写分离，实施起来简单有效。

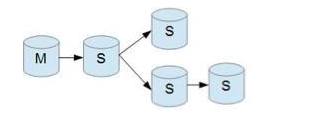
多主一从：



将多个mysql数据库备份到一台存储性能比较好的服务器上。

双主复制：双主复制，也就是互做主从复制，每个master既是master，又是另外一台服务器的slave。这样任何一方所做的变更，都会通过复制应用到另外一方的数据库中。

级联复制：



级联复制模式下，部分slave的数据同步不连接主节点，而是连接从节点。因为如果主节点有太多的从节点，就会损耗一部分性能用于replication，那么我们可以让3~5个从节点连接主节点，其它从节点作为二级或者三级与从节点连接，这样不仅可以缓解主节点的压力，并且对数据一致性没有负面影响。

Mysql主从复制原理：

Mysql的主从复制主要涉及三个线程一个在主节点其余两个在从节点。

1. 主节点binary log dump线程

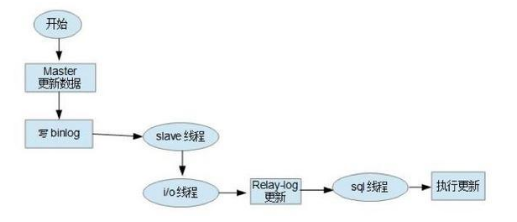
当从节点连接主节点的时候，主节点会创造一个log dump线程，用于发送bin log的内容。在读取bin log的过程中，此线程会对主节点的bin-log加锁，当读取完成时，甚至是发送给从节点之前，锁被释放。

1. 从节点I/O线程

当从节点上执行“start slave”的指令后，从节点会创建一个I/O去连接主节点，请求主控中更新bin log；I/O线程接收到主节点binlog dump 进程发来的更新之后，保存在本地relay-log中。

1. 从节点SQL线程

Sql线程负责读取relay-log中的内容。解析成具体的操作步骤实现主从复制数据的一致性。

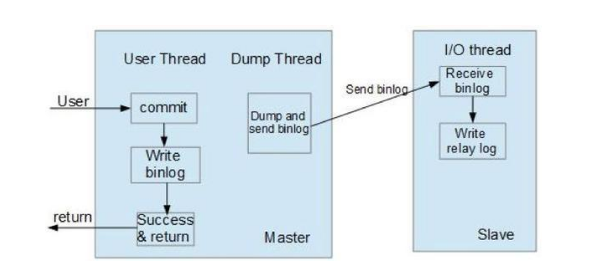


复制的过程基本如下：从节点的I/O进程连接主节点，并请求从指定文件的指定位置之后的日志。

主节点收到从节点的I/O请求后，通过复制i/o进程根据信息读取指定位置之后的日志信息，返回给从节点。返回信息包括有：这次所读取的信息外，还返回bin log file和bin log position；从节点的I/O进程收到内容后，将接收到到内存更新到本机的relay-log中，并且将读到的文件名和位置保存到master-info中，一遍下次访问的时候能确定在具体位置下读取。

Slave的SQL检测到relay-log增加了内容，会将其中增加的内容进行更新，保持主从复制的数据一致性。

异步模式：



这种模式下，主节点不会主动push log到从节点上，可能会导致在数据库服务崩溃的情况下，从节点没有即时的将bin log更新到本地。

半同步模式：

这种情况下主节点只需要接收到一台从节点返回的信息，就会commit。否则需要等待超时时间在切换为异步模式。这样做可以使得主数据库的延迟变小，并且能保证起码是有一个从节点上面传输了信息。

全同步模式：

主节点和从节点全部commit后才会提交成功。

Bin log记录格式：

基于sql语句的复制；基于行的复制；混合模式复制。

基于sql语句的复制：只会记录sql语句到bin log中，节约了I/O，提高了性能。缺点是在某些情况下，会导致主从复制的不一致性。（比如sleep（） now（））

基于行的复制：优点是不会出现某些特定情况下的存储过程、或者函数、或者trigger的调用或者触发无法被正确复制的问题。缺点是会产生大量的日志，尤其是修改table的时候会让日志暴增,同时增加bin log同步时间。也不能通过bin log解析获取执行过的sql语句，只能看到发生的data变更。

混合复制：是以上两种模式的混合，对于一般的复制使用STATEMENT模式保存到binlog，对于STATEMENT模式无法复制的操作则使用ROW模式来保存，MySQL会根据执行的SQL语句选择日志保存方式。

Mysql的底层索引的实现？

索引主要是基于Hash表或者B+树。

数据的索引采用树的结构，因为树的查询效率高还可以保证有序，但是为什么数据库索引不适用二叉树来实现呢？从逻辑上来说二叉树查找速度和比较次数都是最小的，但是为什么实际中不使用呢？

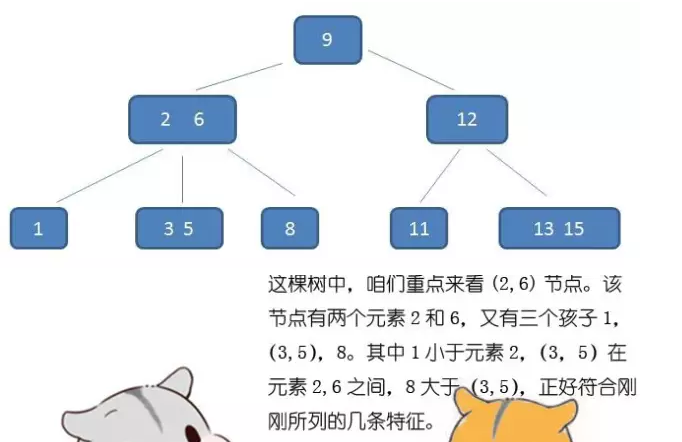
这里我们需要考虑的一个点就是磁盘的IO，数据库的索引是存在磁盘上的，当数据量比较大的时候可能需要几个G。我们在使用数据库索引的时候没法把整个索引加载到内存中，只有逐一加载每个磁盘页，这里面对的磁盘页就是对应着二叉树的节点。 所以在最快的情况下，树的高度为n，所以io的次数就是n。

插入概念：根节点：树最顶端的节点；再往下是子节点，最后没有子节点为叶子节点。

1. Tree的的思想就降低二叉树的高度，减少磁盘的io次数。

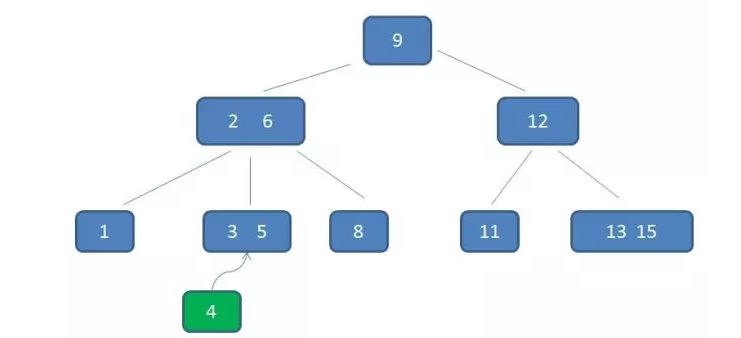
一个m阶的B树的特征为：

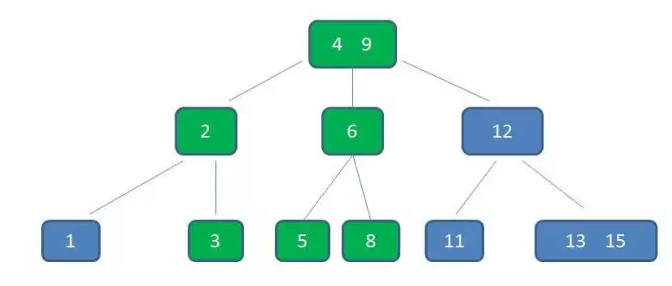
1. 根节点至少有两个儿女。
2. 每个中间节点都有k-1个元素和k个孩子； m/2 <= k <= m
3. 每一个叶子节点都包含k -1 个元素
4. 所有的叶子结点都位于同一层
5. 每个节点的元素从小到大排序

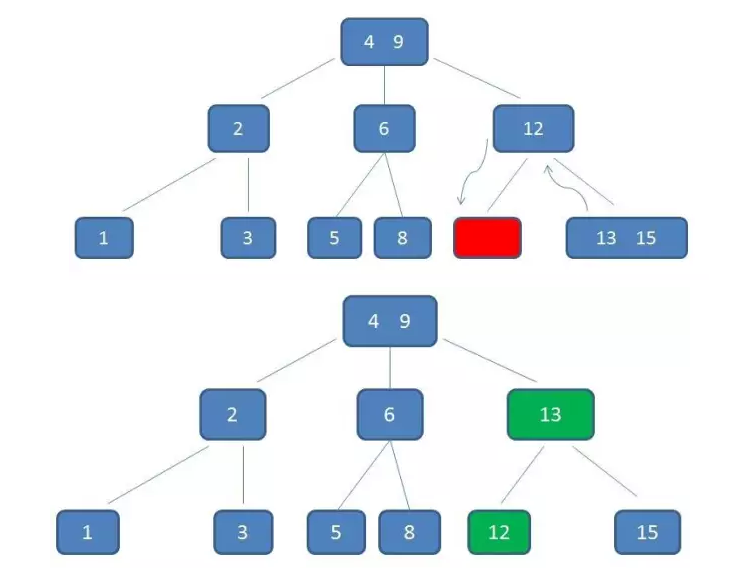


虽然B树的查询速度不一定比二叉树少，但是其磁盘的io次数大大减少，所以在一定程度上提高了系统查询的速度。

插入元素：

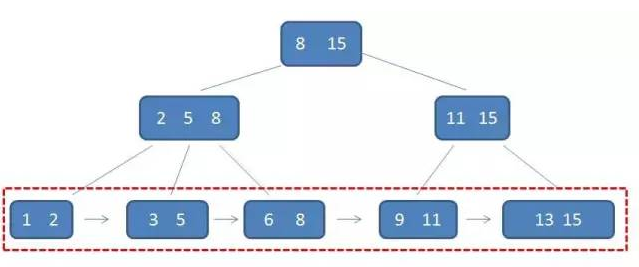






B+是B树的改善，一个m阶的B+树有以下的一些特征：

1. 有k个子节点包含有k个元素，每个元素不是用来保存数据，而是用于索引。所有的数据存放在叶子节点上。
2. 叶子节点包含有全部元素的信息，及包含这些元素的指针，且叶子节点依据元素自小到大顺序连接。
3. 所有中间节点都存放于子节点上，在子节点中是最大或者最小的元素。



B+树和B—Tree的区别？

B+树中间的节点没有卫星数据，所以同样大小的磁盘页可以存储更多的数据。同时B+树比B树更加矮胖，所以在查询的时候IO的次数更少。

1. Tree查询的情况不稳定最好的情况是根节点，最坏的情况是叶子节点。而B+树的查询相对稳定，都是在叶子节点中进行查询。

B+树所有叶子节点形成了有序链表，便于范围查询。

Mysql怎么防止sql注入？

1. 网站代码里写入过滤sql特殊字符的代码，对一些特殊字符进行转化，比如单引号，逗号，\*，（括号）AND 1=1 、反斜杠，select union等查询的sql语句都进行安全过滤，限制这些字符的输入，禁止提交到后端中去。
2. 在mysql5.1后，提供了类似于jdbc的预处理-参数化查询。它的查询方法是：

先预发送一个sql模板过去，再向mysql发送需要查询的参数。就好像填空题一样，不管参数怎么注入，mysql都能知道这是变量，不会做语义解析，起到防注入的效果，这是在mysql中完成的