**社群面试真题1：MySQL单个实例buffer数据和磁盘数据是如何保证强一致性的**

具体问题： 由于MySQL底层有buffer存在，MySQL单个实例buffer数据和磁盘数据是如何保证强一致性的。

社群小伙伴说明：  
我的答案是不能保证强一致性，然后如果宕机了只能通过redolog和binlog进行数据恢复。但是好像面试官说这个是错的，然后底层有机制保证。请问下大家知道是什么机制吗？我查了下书本，好像INNODB有个double write机制，但是也保证不了强一致性

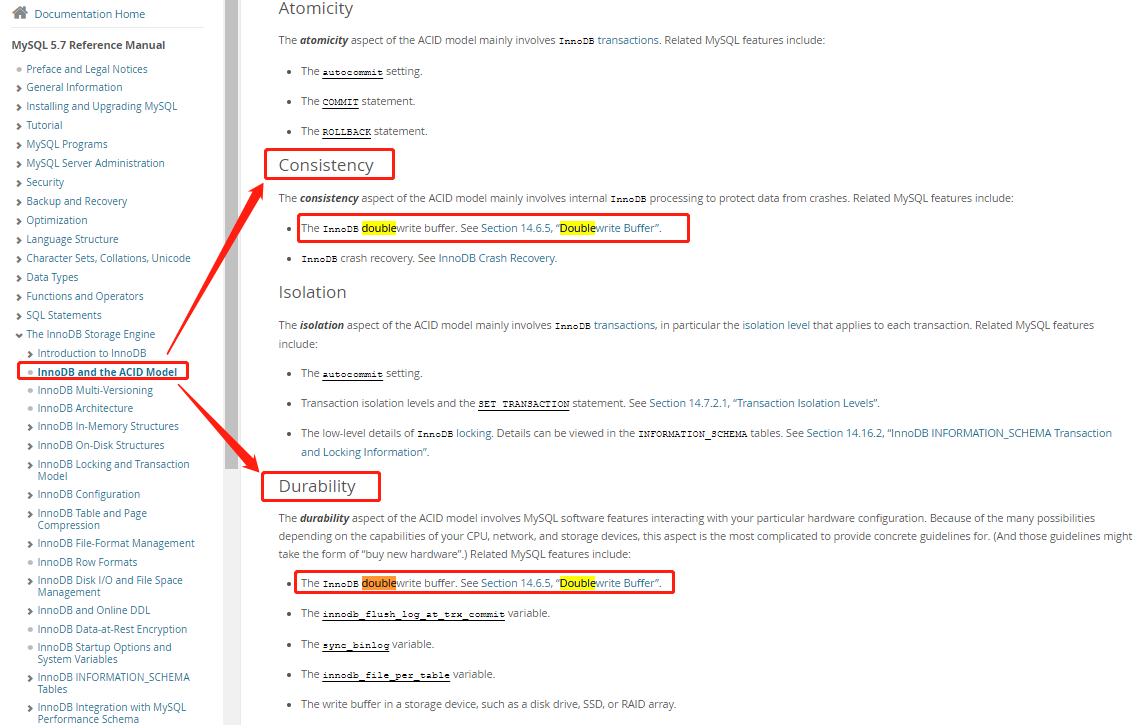
**参考答案：**

innodb三大特性的之一，双写缓冲区（double write）。 另外， redolog解决不了部分页写入问题，因为mysql的页大小是16k，操作系统是4k，写了一半就断电，redolog没法恢复。

双写缓冲区是InnoDB的三大特性之一，还有两个是 Buffer Pool简称BP、自适应Hash索引。doublewrite缓冲区是一个存储区，在该存储区中，**InnoDB**将页面写入**InnoDB**数据文件中的适当位置之前，先从缓冲池中刷新页面 。如果在页面写入过程中存在操作系统，存储子系统或意外的[**mysqld**](https://dev.mysql.com/doc/refman/5.7/en/mysqld.html)进程退出，则**InnoDB**可以在崩溃恢复期间从doublewrite缓冲区中找到页面的良好副本。注意：系统恢复后，MySQL 可以根据redolog 进行恢复，而mysql在恢复的过程中是检查page的checksum，checksum就是pgae的最后事务号，发生partial page write 问题时，page已经损坏，找不到该page中的事务号，就无法恢复。

​ 为什么需要双写？个人理解宏观上还是与InnoDB需要支持事务（ACID）特性有关，而底层的原因是为了解决Partial Write Page问题。

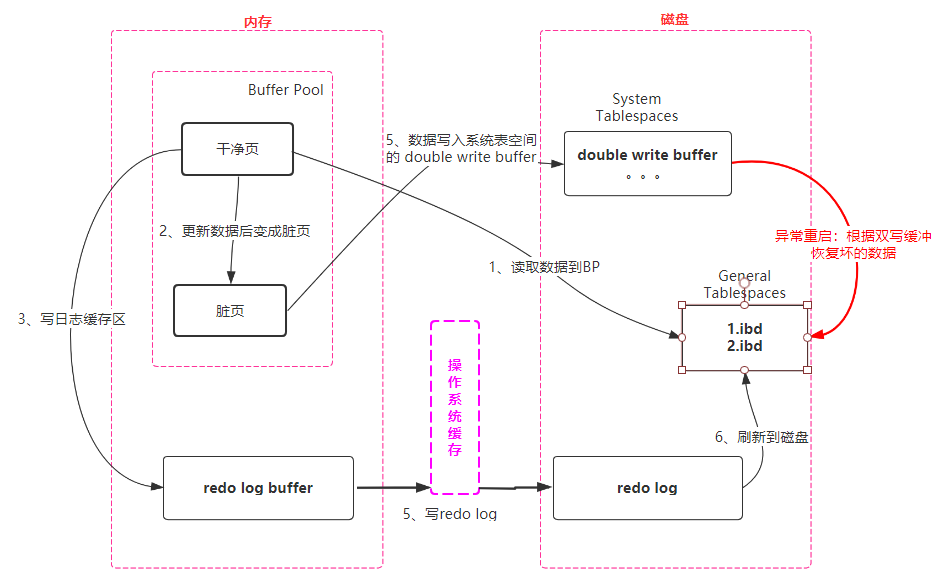
​ MYsql 为了实现事务InnoDB引入了比较多的组件，设计的特别复杂，InnoDB级别包括：（行锁、临建锁、间隙锁）锁和加锁规则、MVCC、redo log、undo log、视图（Read View）。而官方文档也在隔离型和持久性上面明确指向了数据双写机制，如下图

[](https://img-blog.csdnimg.cn/20210325012904731.png)

​ InnoDB的页大小默认为16K，可以使用参数innodb\_page\_size设置， 可设置的值有： 64KB，32KB，16KB（默认），8KB和4KB。并且在数据校验时也针对页进行计算，即他们是一个整个对待，包括把数据持久化到磁盘的操作。而计算机的硬件和操作系统在极端情况下（比如断电、系统崩溃）时，刚写入了4K或8K数据，那么就不能保证该操作的原子性，称为**部分页面写问题（Partial Write Page）**。

​ 此时就引入了双写缓存区的机制，当发生极端情况时，可以从系统表空间的Double Write Buffer【磁盘上】进行恢复，下面是InnoDB的架构图、双写和恢复流程图。为了方便对比，将组件放在了相同的位置：

[](https://img-blog.csdnimg.cn/20210325014100172.png)

[](https://img-blog.csdnimg.cn/20210325132058917.png)

​ 这样在极端情况下也能解决 Partial Write page问题了，但是如果我自己的系统本身数据要求没有那么高（比如日志数据库），这样的话毕竟双写是有一定的性能开销的。可以通过参数**innodb\_doublewrite** = 0进行关闭，设置为1表示开启。官方认为，尽管需要写入两次数据，但是写缓冲区不需要两次的 io开销或操作，因为只需要调用一次操作系统的fsync() 就可以将批量数据顺序写入磁盘 -> 系统表空间的Double Write Buffer（如上图），这里是顺序写而不是随机写（性能可以保证），当然前提是配置刷盘策略参数**innodb\_flush\_method**为默认的O\_DIRECT。其实还有一点就是真正提交的时候会使用组提交，我们可以用参数控制：**binlog\_group\_commit\_sync\_delay**：组提交执行fsync() 延迟的微妙数，延迟时间越长批量数据越多，磁盘io越少性能越高。**binlog\_group\_commit\_sync\_no\_delay\_count**：组提交执行fsync的批个数。

注：更多的 面试真题，请来疯狂创客圈 微信群交流， 入群请加尼恩 微信，具体请参见 疯狂创客圈 社群 总入口。

**史上最全 Java 面试题：Mysql 篇**

**1、MySQL 中有哪几种锁？**

（1）表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最 高，并发度最低。

（2）行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最 低，并发度也最高。

（3）页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表 锁和行锁之间，并发度一般。

**2、MySQL 中有哪些不同的表？**

共有 5 种类型的表：

（1）MyISAM

（2）Heap

（3）Merge

（4）INNODB

（5）ISAM

**3、简述在 MySQL 数据库中 MyISAM 和 InnoDB 的区别**

**MyISAM：**

（1）不支持事务，但是每次查询都是原子的；

（2）支持表级锁，即每次操作是对整个表加锁；

（3）存储表的总行数；

（4）一个 MYISAM 表有三个文件：索引文件、表结构文件、数据文件；

（5）采用菲聚集索引，索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致，但是辅索引不用保证唯一性。

**InnoDb：**

（1）支持 ACID 的事务，支持事务的四种隔离级别；

（2）支持行级锁及外键约束：因此可以支持写并发；

（3）不存储总行数：

（4）一个 InnoDb 引擎存储在一个文件空间（共享表空间，表大小不受操作系统控制，一个表可能分布在多个文件里），也有可能为多个（设置为独立表空，表大小受操作系统文件大小限制，一般为 2G），受操作系统文件大小的限制；

（5）主键索引采用聚集索引（索引的数据域存储数据文件本身），辅索引的数据域存储主键的值；因此从辅索引查找数据，需要先通过辅索引找到主键值，再访问辅索引；最好使用自增主键，防止插入数据时，为维持 B+树结构，文件的大调整。

**4、MySQL 中 InnoDB 支持的四种事务隔离级别名称，以及逐级之间的区别**

SQL 标准定义的四个隔离级别为：

（1）read uncommited ：读到未提交数据

（2）read committed：脏读，不可重复读

（3）repeatable read：可重读

（4）serializable ：串行事物

**5、CHAR 和 VARCHAR 的区别**？

（1）CHAR 和 VARCHAR 类型在存储和检索方面有所不同

（2）CHAR 列长度固定为创建表时声明的长度，长度值范围是 1 到 255 当 CHAR值被存储时，它们被用空格填充到特定长度，检索 CHAR 值时需删除尾随空格。

**6、主键和候选键有什么区别？**

表格的每一行都由主键唯一标识,一个表只有一个主键。

主键也是候选键。按照惯例，候选键可以被指定为主键，并且可以用于任何外键引用。

**7、myisamchk 是用来做什么的？**

它用来压缩 MyISAM 表，这减少了磁盘或内存使用。

MyISAM Static 和 MyISAM Dynamic 有什么区别？

在 MyISAM Static 上的所有字段有固定宽度。动态 MyISAM 表将具有像 TEXT，BLOB 等字段，以适应不同长度的数据类型。

MyISAM Static 在受损情况下更容易恢复。

**8、如果一个表有一列定义为 TIMESTAMP，将发生什么？**

每当行被更改时，时间戳字段将获取当前时间戳。

列设置为 AUTO INCREMENT 时，如果在表中达到最大值，会发生什么情况？

它会停止递增，任何进一步的插入都将产生错误，因为密钥已被使用。

怎样才能找出最后一次插入时分配了哪个自动增量？

LAST\_INSERT\_ID 将返回由 Auto\_increment 分配的最后一个值，并且不需要指定表名称。

**9、你怎么看到为表格定义的所有索引？**

索引是通过以下方式为表格定义的：

SHOW INDEX FROM ;

**10、LIKE 声明中的％和\_是什么意思？**

％对应于 0 个或更多字符，\_只是 LIKE 语句中的一个字符。

如何在 Unix 和 MySQL 时间戳之间进行转换？

UNIX\_TIMESTAMP 是从 MySQL 时间戳转换为 Unix 时间戳的命令

FROM\_UNIXTIME 是从 Unix 时间戳转换为 MySQL 时间戳的命令

**11、列对比运算符是什么？**

在 SELECT 语句的列比较中使用=，<>，<=，<，> =，>，<<，>>，<=>，AND，OR 或 LIKE 运算符。

**12、BLOB 和 TEXT 有什么区别？**

BLOB 是一个二进制对象，可以容纳可变数量的数据。TEXT 是一个不区分大小写的 BLOB。

BLOB 和 TEXT 类型之间的唯一区别在于对 BLOB 值进行排序和比较时区分大小写，对 TEXT 值不区分大小写。

**13、MySQL\_fetch\_array 和 MySQL\_fetch\_object 的区别是什么？**

以下是 MySQL\_fetch\_array 和 MySQL\_fetch\_object 的区别：

MySQL\_fetch\_array（） – 将结果行作为关联数组或来自数据库的常规数组返回。

MySQL\_fetch\_object – 从数据库返回结果行作为对象。

**14、MyISAM 表格将在哪里存储，并且还提供其存储格式？**

每个 MyISAM 表格以三种格式存储在磁盘上：

（1）·“.frm”文件存储表定义

（2）·数据文件具有“.MYD”（MYData）扩展名

（3）索引文件具有“.MYI”（MYIndex）扩展名

**15、MySQL 如何优化 DISTINCT？**

DISTINCT 在所有列上转换为 GROUP BY，并与 ORDER BY 子句结合使用。

SELECT DISTINCT t1.a FROM t1,t2 where t1.a=t2.a;

**16、如何显示前 50 行？**

在 MySQL 中，使用以下代码查询显示前 50 行：

SELECT\*FROM

LIMIT 0,50;

**17、可以使用多少列创建索引？**

任何标准表最多可以创建 16 个索引列。

**18、NOW（）和 CURRENT\_DATE（）有什么区别？**

NOW（）命令用于显示当前年份，月份，日期，小时，分钟和秒。

CURRENT\_DATE（）仅显示当前年份，月份和日期。

**19、什么是非标准字符串类型？**

（1）TINYTEXT

（2）TEXT

（3）MEDIUMTEXT

（4）LONGTEXT

**20、什么是通用 SQL 函数？**

（1）CONCAT(A, B) – 连接两个字符串值以创建单个字符串输出。通常用于将两个或多个字段合并为一个字段。

（2）FORMAT(X, D)- 格式化数字 X 到 D 有效数字。

（3）CURRDATE(), CURRTIME()- 返回当前日期或时间。

（4）NOW（） – 将当前日期和时间作为一个值返回。

（5）MONTH（），DAY（），YEAR（），WEEK（），WEEKDAY（） – 从日期值中提取给定数据。

（6）HOUR（），MINUTE（），SECOND（） – 从时间值中提取给定数据。

（7）DATEDIFF（A，B） – 确定两个日期之间的差异，通常用于计算年龄

（8）SUBTIMES（A，B） – 确定两次之间的差异。

（9）FROMDAYS（INT） – 将整数天数转换为日期值。

**21、MySQL 支持事务吗？**

在缺省模式下，MySQL 是 autocommit 模式的，所有的数据库更新操作都会即时提交，所以在缺省情况下，MySQL 是不支持事务的。

但是如果你的 MySQL 表类型是使用 InnoDB Tables 或 BDB tables 的话，你的MySQL 就可以使用事务处理,使用 SETAUTOCOMMIT=0 就可以使 MySQL 允许在非 autocommit 模式，在非autocommit 模式下，你必须使用 COMMIT 来提交你的更改，或者用 ROLLBACK来回滚你的更改。

**22、MySQL 里记录货币用什么字段类型好**

NUMERIC 和 DECIMAL 类型被MySQL 实现为同样的类型，这在 SQL92 标准允许。他们被用于保存值，该值的准确精度是极其重要的值，例如与金钱有关的数据。当声明一个类是这些类型之一时，精度和规模的能被(并且通常是)指定。

例如：

salary DECIMAL(9,2)

在这个例子中，9(precision)代表将被用于存储值的总的小数位数，而 2(scale)代 表将被用于存储小数点后的位数。

因此，在这种情况下，能被存储在 salary 列中的值的范围是从-9999999.99 到9999999.99。

**23、MySQL 有关权限的表都有哪几个？**

MySQL 服务器通过权限表来控制用户对数据库的访问，权限表存放在 MySQL 数据库里，由 MySQL\_install\_db 脚本初始化。这些权限表分别 user，db，table\_priv，columns\_priv 和 host。

**24、列的字符串类型可以是什么？**

字符串类型是：

（1）SET2

（2）BLOB

（3）ENUM

（4）CHAR

（5）TEXT

**25、MySQL 数据库作发布系统的存储，一天五万条以上的增量，预计运维三年,怎么优化？**

（1）设计良好的数据库结构，允许部分数据冗余，尽量避免 join 查询，提高效率。

（2）选择合适的表字段数据类型和存储引擎，适当的添加索引。

（3）MySQL 库主从读写分离。

（4）找规律分表，减少单表中的数据量提高查询速度。

（5）添加缓存机制，比如 memcached，apc 等。

（6）不经常改动的页面，生成静态页面。

（7）书写高效率的 SQL。比如 SELECT \* FROM TABEL 改为 SELECT field\_1, field\_2, field\_3 FROM TABLE.

**26、锁的优化策略**

（1）读写分离

（2）分段加锁

（3）减少锁持有的时间

（4）多个线程尽量以相同的顺序去获取资源

不能将锁的粒度过于细化，不然可能会出现线程的加锁和释放次数过多，反而效率不如一次加一把大锁。

**27、索引的底层实现原理和优化**

B+树，经过优化的 B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此 InnoDB 建议为大部分表使用默认自增的主键作为主索引。

**28、什么情况下设置了索引但无法使用**

（1）以“%”开头的 LIKE 语句，模糊匹配

（2）OR 语句前后没有同时使用索引

（3）数据类型出现隐式转化（如 varchar 不加单引号的话可能会自动转换为 int 型）

**29、实践中如何优化 MySQL**

最好是按照以下顺序优化：

（1）SQL 语句及索引的优化

（2）数据库表结构的优化

（3）系统配置的优化

（4）硬件的优化

**30、优化数据库的方法**

（1）选取最适用的字段属性，尽可能减少定义字段宽度，尽量把字段设置 NOTNULL，例如’省份’、’性别’最好适用 ENUM

（2）使用连接(JOIN)来代替子查询

（3）适用联合(UNION)来代替手动创建的临时表

（4）事务处理

（5）锁定表、优化事务处理

（6）适用外键，优化锁定表

（7）建立索引

（8）优化查询语句

**31、简单描述 MySQL 中，索引，主键，唯一索引，联合索引的区别，对数据库的性能有什么影响（从读写两方面）**

索引是一种特殊的文件(InnoDB 数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

普通索引(由关键字 KEY 或 INDEX 定义的索引)的唯一任务是加快对数据的访问速度。

普通索引允许被索引的数据列包含重复的值。如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字 UNIQUE 把它定义为一个唯一索引。也就是说，唯一索引可以保证数据记录的唯一性。

主键，是一种特殊的唯一索引，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。

索引可以覆盖多个数据列，如像 INDEX(columnA, columnB)索引，这就是联合索引。

索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件。

**32、数据库中的事务是什么?**

事务（transaction）是作为一个单元的一组有序的数据库操作。如果组中的所有操作都成功，则认为事务成功，即使只有一个操作失败，事务也不成功。如果所有操作完成，事务则提交，其修改将作用于所有其他数据库进程。如果一个操作失败，则事务将回滚，该事务所有操作的影响都将取消。

事务特性：

（1）原子性：即不可分割性，事务要么全部被执行，要么就全部不被执行。

（2）一致性或可串性。事务的执行使得数据库从一种正确状态转换成另一种正确状态。

（3）隔离性。在事务正确提交之前，不允许把该事务对数据的任何改变提供给任何其他事务。

（4）持久性。事务正确提交后，其结果将永久保存在数据库中，即使在事务提交后有了其他故障，事务的处理结果也会得到保存。

或者这样理解：

事务就是被绑定在一起作为一个逻辑工作单元的 SQL 语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。要将有组语句作为事务考虑，就需要通过 ACID 测试，即原子性，一致性，隔离性和持久性。

**33、SQL 注入漏洞产生的原因？如何防止？**

SQL 注入产生的原因：程序开发过程中不注意规范书写 sql 语句和对特殊字符进行过滤，导致客户端可以通过全局变量 POST 和 GET 提交一些 sql 语句正常执行。

防止 SQL 注入的方式：

开启配置文件中的 magic\_quotes\_gpc 和 magic\_quotes\_runtime 设置

执行 sql 语句时使用 addslashes 进行 sql 语句转换

Sql 语句书写尽量不要省略双引号和单引号。

过滤掉 sql 语句中的一些关键词：update、insert、delete、select、 \* 。

提高数据库表和字段的命名技巧，对一些重要的字段根据程序的特点命名，取不易被猜到的。

**34、为表中得字段选择合适得数据类型**

字段类型优先级: 整形>date,time>enum,char>varchar>blob,text

优先考虑数字类型，其次是日期或者二进制类型，最后是字符串类型，同级别得数据类型，应该优先选择占用空间小的数据类型

**35、存储时期**

Datatime:以 YYYY-MM-DD HH:MM:SS 格式存储时期时间，精确到秒，占用 8 个字节得存储空间，datatime 类型与时区无关Timestamp:以时间戳格式存储，占用 4 个字节，范围小 1970-1-1 到 2038-1-19，显示依赖于所指定得时区，默认在第一个列行的数据修改时可以自动得修改timestamp 列得值

Date:（生日）占用得字节数比使用字符串.datatime.int 储存要少，使用 date 只需要 3 个字节，存储日期月份，还可以利用日期时间函数进行日期间得计算

Time:存储时间部分得数据

注意:不要使用字符串类型来存储日期时间数据（通常比字符串占用得储存空间小，在进行查找过滤可以利用日期得函数）

使用 int 存储日期时间不如使用 timestamp 类型

**36、对于关系型数据库而言，索引是相当重要的概念，请回答有关索引的几个问题：**

**（1）索引的目的是什么？**

快速访问数据表中的特定信息，提高检索速度

创建唯一性索引，保证数据库表中每一行数据的唯一性。

加速表和表之间的连接

使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间

**（2）索引对数据库系统的负面影响是什么？**

负面影响：

创建索引和维护索引需要耗费时间，这个时间随着数据量的增加而增加；索引需要占用物理空间，不光是表需要占用数据空间，每个索引也需要占用物理空间；当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

**（3）为数据表建立索引的原则有哪些？**

在最频繁使用的、用以缩小查询范围的字段上建立索引。

在频繁使用的、需要排序的字段上建立索引

**（4）什么情况下不宜建立索引？**

对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引。

对于一些特殊的数据类型，不宜建立索引，比如文本字段（text）等

**37、解释 MySQL 外连接、内连接与自连接的区别**

先说什么是交叉连接: 交叉连接又叫笛卡尔积，它是指不使用任何条件，直接将一个表的所有记录和另一个表中的所有记录一一匹配。

内连接 则是只有条件的交叉连接，根据某个条件筛选出符合条件的记录，不符合条件的记录不会出现在结果集中，即内连接只连接匹配的行。

外连接 其结果集中不仅包含符合连接条件的行，而且还会包括左表、右表或两个表中的所有数据行，这三种情况依次称之为左外连接，右外连接，和全外连接。

左外连接，也称左连接，左表为主表，左表中的所有记录都会出现在结果集中，对于那些在右表中并没有匹配的记录，仍然要显示，右边对应的那些字段值以NULL 来填充。右外连接，也称右连接，右表为主表，右表中的所有记录都会出现在结果集中。左连接和右连接可以互换，MySQL 目前还不支持全外连接。

**38、Myql 中的事务回滚机制概述**

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位，事务回滚是指将该事务已经完成的对数据库的更新操作撤销。

要同时修改数据库中两个不同表时，如果它们不是一个事务的话，当第一个表修改完，可能第二个表修改过程中出现了异常而没能修改，此时就只有第二个表依旧是未修改之前的状态，而第一个表已经被修改完毕。而当你把它们设定为一个事务的时候，当第一个表修改完，第二表修改出现异常而没能修改，第一个表和第二个表都要回到未修改的状态，这就是所谓的事务回滚

**39、SQL 语言包括哪几部分？每部分都有哪些操作关键字？**

SQL 语言包括数据定义(DDL)、数据操纵(DML),数据控制(DCL)和数据查询（DQL） 四个部分。

数据定义：Create Table,Alter Table,Drop Table, Craete/Drop Index 等

数据操纵：Select ,insert,update,delete,

数据控制：grant,revoke

数据查询：select

**40、完整性约束包括哪些？**

数据完整性(Data Integrity)是指数据的精确(Accuracy)和可靠性(Reliability)。

分为以下四类：

（1）实体完整性：规定表的每一行在表中是惟一的实体。

（2）域完整性：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。

（3）参照完整性：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。

（4）用户定义的完整性：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。

与表有关的约束：包括列约束(NOT NULL（非空约束）)和表约束(PRIMARY KEY、foreign key、check、UNIQUE) 。

**41、什么是锁？**

数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

加锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。

基本锁类型：锁包括行级锁和表级锁

**42、什么叫视图？游标是什么？**

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

**43、什么是存储过程？用什么来调用？**

存储过程是一个预编译的 SQL 语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次 SQL，使用存储过程比单纯 SQL 语句执行要快。可以用一个命令对象来调用存储过程。

**44、如何通俗地理解三个范式？**

第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解；

第二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；

第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。。

**范式化设计优缺点:**

优点:可以尽量得减少数据冗余，使得更新快，体积小

缺点:对于查询需要多个表进行关联，减少写得效率增加读得效率，更难进行索引优化

**反范式化:**

优点:可以减少表得关联，可以更好得进行索引优化

缺点:数据冗余以及数据异常，数据得修改需要更多的成本

**45、什么是基本表？什么是视图？**

基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表

**46、试述视图的优点？**

(1) 视图能够简化用户的操作

(2) 视图使用户能以多种角度看待同一数据；

(3) 视图为数据库提供了一定程度的逻辑独立性；

(4)视图能够对机密数据提供安全保护。

**47、 NULL 是什么意思**

NULL 这个值表示 UNKNOWN(未知):它不表示“”(空字符串)。对 NULL 这个值的任何比较都会生产一个 NULL 值。您不能把任何值与一个 NULL 值进行比较，并在逻辑上希望获得一个答案。

使用 IS NULL 来进行 NULL 判断

**48、主键、外键和索引的区别？**

主键、外键和索引的区别

**定义：**

主键——唯一标识一条记录，不能有重复的，不允许为空

外键——表的外键是另一表的主键, 外键可以有重复的, 可以是空值

索引——该字段没有重复值，但可以有一个空值

**作用：**

主键——用来保证数据完整性

外键——用来和其他表建立联系用的

索引——是提高查询排序的速度

**个数：**

主键—— 主键只能有一个

外键—— 一个表可以有多个外键

索引—— 一个表可以有多个唯一索引

**49、你可以用什么来确保表格里的字段只接受特定范围里的值?**

Check 限制，它在数据库表格里被定义，用来限制输入该列的值。

触发器也可以被用来限制数据库表格里的字段能够接受的值，但是这种办法要求触发器在表格里被定义，这可能会在某些情况下影响到性能。

**50、说说对 SQL 语句优化有哪些方法？（选择几条）**

（1）Where 子句中：where 表之间的连接必须写在其他 Where 条件之前，那些可以过滤掉最大数量记录的条件必须写在 Where 子句的末尾.HAVING 最后。

（2）用 EXISTS 替代 IN、用 NOT EXISTS 替代 NOT IN。

（3） 避免在索引列上使用计算

（4）避免在索引列上使用 IS NULL 和 IS NOT NULL

（5）对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

（6）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

（7）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

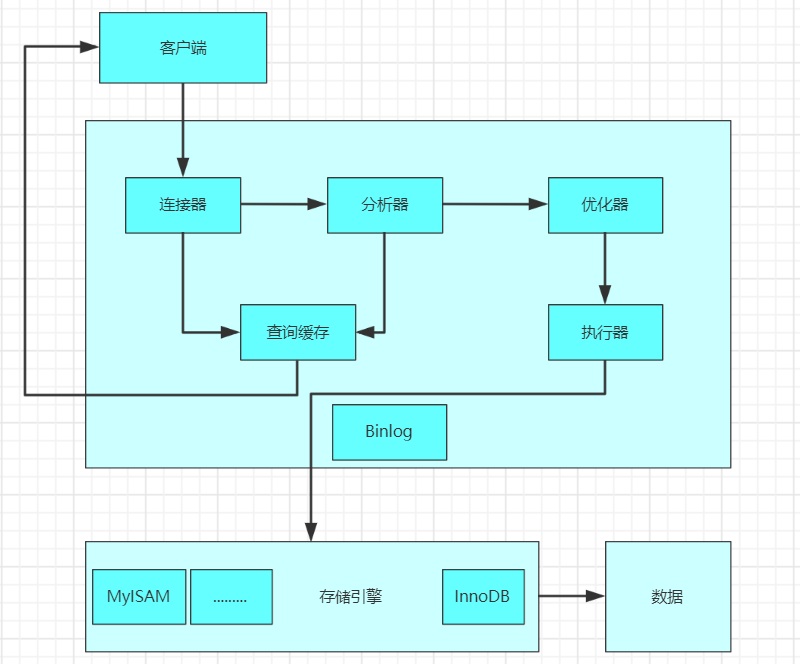
**51 面试官：听说你sql写的挺溜的，你说一说查询sql的执行过程**

当希望Mysql能够高效的执行的时候，最好的办法就是清楚的了解Mysql是如何执行查询的，只有更加全面的了解SQL执行的每一个过程，才能更好的进行SQl的优化。

当执行一条查询的SQl的时候大概发生了一下的步骤：

1. 客户端发送查询语句给服务器。
2. 服务器首先检查缓存中是否存在该查询，若存在，返回缓存中存在的结果。若是不存在就进行下一步。
3. 服务器进行SQl的解析、语法检测和预处理，再由优化器生成对应的执行计划。
4. Mysql的执行器根据优化器生成的执行计划执行，调用存储引擎的接口进行查询。
5. 服务器将查询的结果返回客户端。

**Mysql的执行的流程**

Mysql的执行的流程图如下图所示：  
[](https://img-blog.csdnimg.cn/20200330211824877.png)

这里以一个实例进行说明Mysql的的执行过程，新建一个User表，如下：

// 新建一个表

DROP TABLE IF EXISTS User;

CREATE TABLE `User` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(10) DEFAULT NULL,

`age` int DEFAULT 0,

`address` varchar(255) DEFAULT NULL,

`phone` varchar(255) DEFAULT NULL,

`dept` int,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=40 DEFAULT CHARSET=utf8;

// 并初始化数据，如下

INSERT INTO User(name,age,address,phone,dept)VALUES('张三',24,'北京','13265543552',2);

INSERT INTO User(name,age,address,phone,dept)VALUES('张三三',20,'北京','13265543557',2);

INSERT INTO User(name,age,address,phone,dept)VALUES('李四',23,'上海','13265543553',2);

INSERT INTO User(name,age,address,phone,dept)VALUES('李四四',21,'上海','13265543556',2);

INSERT INTO User(name,age,address,phone,dept)VALUES('王五',27,'广州','13265543558',3);

INSERT INTO User(name,age,address,phone,dept)VALUES('王五五',26,'广州','13265543559',3);

INSERT INTO User(name,age,address,phone,dept)VALUES('赵六',25,'深圳','13265543550',3);

INSERT INTO User(name,age,address,phone,dept)VALUES('赵六六',28,'广州','13265543561',3);

INSERT INTO User(name,age,address,phone,dept)VALUES('七七',29,'广州','13265543562',4);

INSERT INTO User(name,age,address,phone,dept)VALUES('八八',23,'广州','13265543563',4);

INSERT INTO User(name,age,address,phone,dept)VALUES('九九',24,'广州','13265543564',4);

现在针对这个表发出一条SQl查询：**查询每个部门中25岁以下的员工个数大于3的员工个数和部门编号，并按照人工个数降序排序和部门编号升序排序的前两个部门。**

SELECT dept,COUNT(phone) AS num FROM User WHERE age< 25 GROUP BY dept HAVING num >= 3 ORDER BY num DESC,dept ASC LIMIT 0,2;

**执行连接器**

开始执行这条sql时，会检查该语句是否有权限，若是没有权限就直接返回错误信息，有权限会进行下一步，校验权限的这一步是在图一的连接器进行的，对连接用户权限的校验。

**执行检索内存**

相连建立之后，履行查询语句的时候，会先行检索内存，Mysql会先行冗余这个sql与否履行过，以此**Key-Value**的形式平缓适用内存中，Key是**检索预定**，Value是**结果集**。

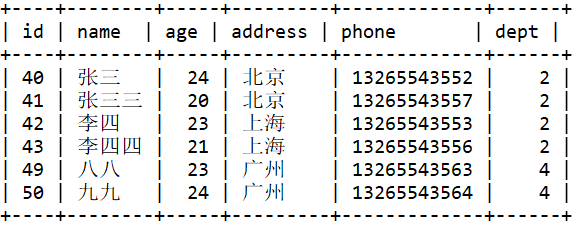
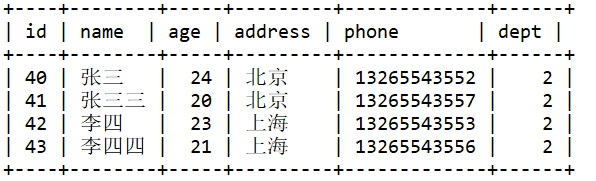
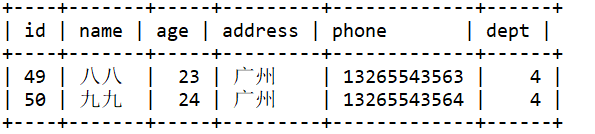
假如内存key遭击中，便会间接回到给客户端，假如没命中，便会履行后续的操作，完工之后亦会将结果内存上去，当下一次进行查询的时候也是如此的循环操作。

**执行分析器**

分析器主要有两步：（1）词法分析（2）语法分析

词法分析主要执行**提炼关键性字**，比如select，**提交检索的表**，**提交字段名**，**提交检索条件**。

语法分析主要执行辨别你**输出的sql与否准确**，是否**合乎mysql的语法**。

当Mysql没有命中内存的时候，接着执行的是 FROM student 负责把数据库的表文件加载到内存中去，**WHERE age< 60**，会把所示表中的数据进行过滤，取出符合条件的记录行，生成一张临时表，如下图所示。  
[](https://img-blog.csdnimg.cn/20200330213710187.png)  
**GROUP BY dept** 会把上图的临时表分成若干临时表，切分的过程如下图所示：  
[](https://img-blog.csdnimg.cn/20200330214252626.png)  
[](https://img-blog.csdnimg.cn/20200330214316713.png)

查询的结果只有部门2和部门3才有符合条件的值，生成如上两图的临时表。接着执行**SELECT后面的字段**，SELECT后面可以是**表字段**也可以是**聚合函数**。

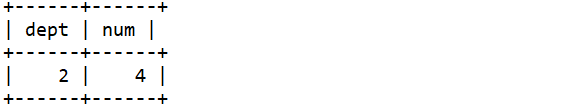
这里SELECT的情况与是否存在**GROUP BY**有关，若是不存在Mysql直接按照上图内存中整列读取。若是存在分别SELECT临时表的数据。

最后生成的临时表如下图所示：  
[](https://img-blog.csdnimg.cn/20200330215715536.png)

紧接着执行**HAVING num>2**过滤员工数小于等于2的部门，对于**WHERE**和**HAVING**都是进行过滤，那么这两者有什么不同呢？

第一点是WHERE后面只能对表字段进行过滤，不能使用聚合函数，而HAVING可以过滤表字段也可以使用聚合函数进行过滤。

第二点是WHERE是对执行from USer操作后，加载表数据到内存后，WHERE是对**原生表的字段**进行过滤，而HAVING是对**SELECT后的字段进行过滤**，也就是WHERE**不能使用别名进行过滤**。

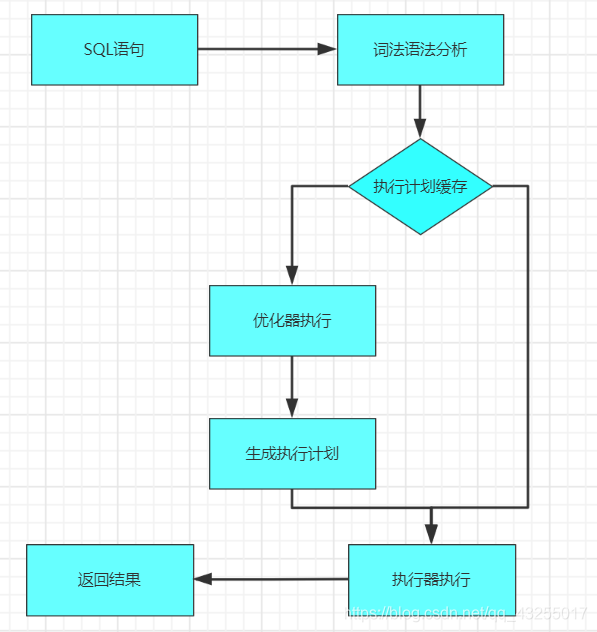
因为执行WHERE的时候，还没有SELECT，还没有给字段赋予别名。接着生成的临时表如下图所示：  
[](https://img-blog.csdnimg.cn/2020033022062543.png)  
最后在执行**ORDER BY后面的排序**以及**limit0,2**取得前两个数据，因为这里数据比较少，没有体现出来。最后生成得结果也是如上图所示。接着判断这个sql语句**是否有语法错误**，**关键性词与否准确**等等。

**执行优化器**

查询优化器会将解析树转化成执行计划。一条查询可以有多种执行方法，最后都是返回相同结果。优化器的作用就是找到这其中**最好的执行计划**。

生成执行计划的过程会消耗较多的时间，特别是存在许多可选的执行计划时。如果在一条SQL语句执行的过程中将该语句对应的最终执行计划进行缓存。

当**相似的语句**再次被输入服务器时，就可以直接**使用已缓存的执行计划**，从而跳过SQL语句生成执行计划的整个过程，进而可以提高语句的执行速度。

[](https://img-blog.csdnimg.cn/2020033022230581.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3FxXzQzMjU1MDE3,size_16,color_FFFFFF,t_70)  
MySQL使用基于成本的查询优化器。它会尝试预测一个查询使用某种执行计划时的成本，并选择其中成本最少的一个。

**执行执行器**

由优化器生成得执行计划，交由执行器进行执行，执行器调用存储引擎得接口，存储引擎获取数据并返回，结束整个查询得过程。

这里之讲解了select的过程，对于update这些修改数据或者删除数据的操作，会涉及到事务，会使用两个日志模块，redo log和binlog日志。具体对这两个日志的介绍请看着一篇文章。

以前的Mysql的默认存储引擎MyISAM引擎是没redo log的，而现在的默认存储引擎InnoDB引擎便是透过redo 复杂度来拥护事务的，保证事务能够准确的回滚或者提交，保证事务的ACID。

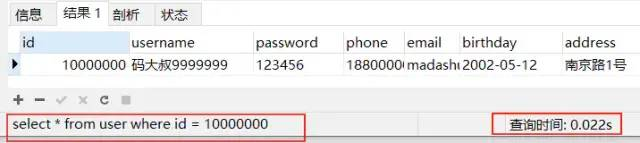
**52：必备：从千万级数据查询来聊一聊索引结构和数据库原理**

在日常工作中我们不可避免地会遇到慢SQL问题，比如笔者在之前的公司时会定期收到DBA彪哥发来的Oracle AWR报告，并特别提示我某条sql近阶段执行明显很慢，可能要优化一下等。对于这样的问题通常大家的第一反应就是看看sql是不是写的不合理啊诸如：*“避免使用in和not in，否则可能会导致全表扫描”“ 避免在where子句中对字段进行函数操作”*等等，还有一种常见的反应就是这个表有没有加索引？绝大部分情况下，加了个索引基本上就搞定了。

既然题目是《从千万级数据查询来聊一聊索引结构和数据库原理》，首先就来构造一个千万级的表直观感受下。我们创建了一张user表，然后插入了1000万条数据，查询一下：

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2V5OWlhd0NhUXNFZ0VacHpobm1tNm5KNk1tbHM3MWFjdGljN0RqUTRIa3lWQkE5ZUpNWmliMmliV3ZnLzY0MA?x-oss-process=image/format,png)

用了近30秒的时间，这还是单表查询，关联查询明显会更让人无法忍受。接下来，我们只是对id增加一个索引，再来验证一把：

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VvMndJbE5lWERlUDVRdGpLbkJXc2ZpY2Q0aWE3cTBySmlhUGlhazdBYldTQ01pYmljMjRuTTZPT2NpYW1BLzY0MA?x-oss-process=image/format,png)

从30s到0.02s，提升了足足1500倍。为什么加了索引之后，速度嗖地一下子就上去了呢？我们从【索引数据结构】、【Mysql原理】两个方面入手。

**一、索引数据结构**

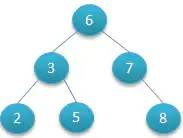
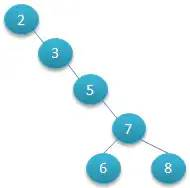
我们先来看下 MySQL官方对索引的定义：

索引（Index）是帮助MySQL高效获取数据的数据结构。

这里面有2个关键词：高效查找、数据结构。对于数据库来说，查询是我们最主要的使用功能，查询速度肯定是越快越好。最基本的查找是顺序查找，更高效的查找我们很自然会想到二叉树、红黑树、Hash表、BTree等等。

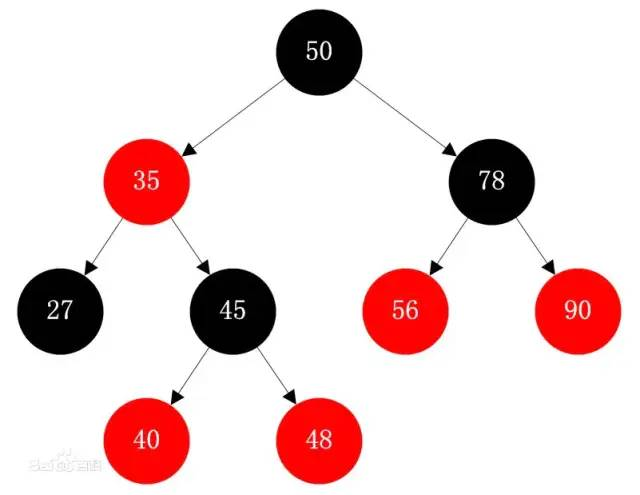
**1.1 二叉树**

这个大家很熟悉了，他有一个很重要的特点：左边节点的键值小于根的键值，右边节点的键值大于根的键值。比如图1，它确实能明显提高我们的搜索性能。但如果用来作为数据库的索引，明显存在很大的缺陷，但对于图2这种递增的id，存储后索引近似于变成了单边的链表，肯定是不合适的。

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2UzaWJpYTI3aWMwRmxjYWR4VUdsQTFGMVJmVnNTMXM4bkRUQ3BtQjNDbGljWWlhQTVkeUVWM1ZiS0ZlUS82NDA?x-oss-process=image/format,png)[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VUaWJ4bFRZbG5FRlRkYWRWZTBNUWliVmQwdlRGbTFMcTNtYUEweWlhUVRQR1c0RklzcFI2R1owbGcvNjQw?x-oss-process=image/format,png)

**1.2 红黑树**

也称之为平衡二叉树。在JDK1.8后，HashMap对底层的链表也优化成了红黑树（后续文章我们可以讲讲Hashmap1.8之后的调整）。平衡二叉树的结构使树的结构较好，明显提高查找运算的速度。但是缺陷也同样很明显，插入和删除运算变得复杂化，从而降低了他们的运算速度。对大数据量的支撑很不好，当数据量很大时，树的高度太高，如果查找的数据是叶子节点，依然会超级慢。

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VPemw1d0IybGE1cnA0MjFPcmxlV09nODRKRVJYNklSMTBqOU1CaWFiZ2NJSlh4bTZDMkpEeUxRLzY0MA?x-oss-process=image/format,png)

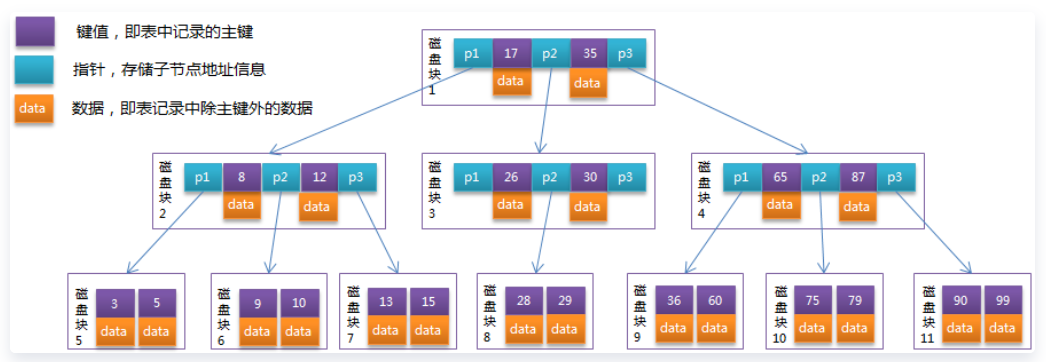
**1.3 BTree**

B-Tree是为磁盘等外存储设备设计的一种平衡查找树。系统从磁盘读取数据到内存时是以磁盘块（block）为基本单位的，位于同一个磁盘块中的数据会被一次性读取到内存中。在Mysql存储引擎中有页（Page）的概念，页是其磁盘管理的最小单位。Mysql存储引擎中默认每个页的大小为16KB，查看方式：

mysql> show variables like 'innodb\_page\_size';

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VaYkJZaWFDbnR1T29INU1zaWNuNHNSY2ZRbGsyV3Z2a0hEcUpuY1MwWURpYzBaM250c1VVcFZpYkZnLzY0MA?x-oss-process=image/format,png)

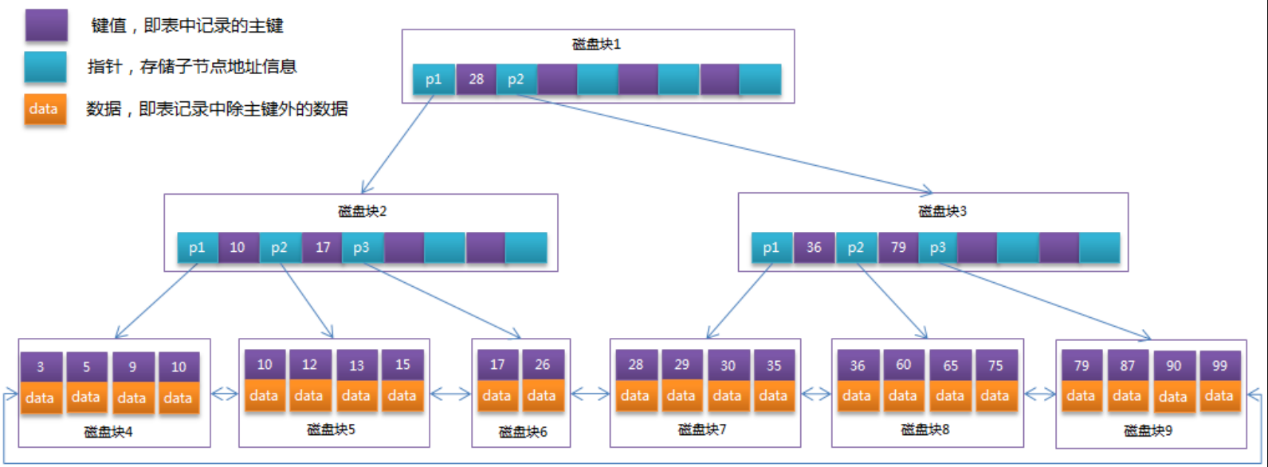
我们也可以将它修改为4K、8K、16K。系统一个磁盘块的存储空间往往没有16K，因此Mysql每次申请磁盘空间时都会将若干地址连续磁盘块来达到页的大小16KB。Mysql在把磁盘数据读入到磁盘时会以页为基本单位，在查询数据时如果一个页中的每条数据都能有助于定位数据记录的位置，这将会减少磁盘I/O次数，提高查询效率。

[](https://img-blog.csdnimg.cn/20210428150732814.png)

如上图所示，一棵B树包含有键值、存储子节点的指针信息、及除主键外的数据。相对于普通的树BTree将横向节点的容量变大，从而存储更多的索引。

**1.4 B+Tree**

在B-Tree的基础上大牛们又研究出了许多变种，其中最常见的是B+Tree，MySQL就普遍使用B+Tree实现其索引结构。

[](https://img-blog.csdnimg.cn/20210428150817317.png)

与B-Tree相比，B+Tree做了以下一些改进：  
1、非叶子节点，只存储键值信息，这样极大增加了存放索引的数据量。  
2、 所有叶子节点之间都有一个链指针。对于区间查询时，不需要再从根节点开始，可直接定位到数据。  
3、 数据记录都存放在叶子节点中。根据二叉树的特点，这个是顺序访问指针，提升了区间访问的性能。  
通过这样的设计，一张千万级的表最多只需要3次磁盘交互就可以找出数据。

**二、Mysql部分原理说明**

这一部分我们选举几个日常面试过程中或者使用过程中比较常见的问题通过问答的形式来进行讲解。

**2.1、数据库引擎MyISAM和InnoDB有什么区别**

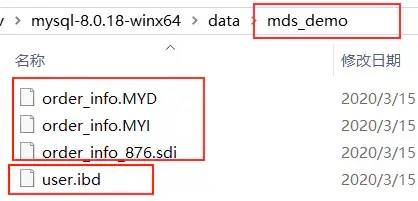
* MyISAM：  
  在Mysql8之前，默认引擎是MyISAM，其目标是快速读取。  
  特点：  
  1、读取非常快，如果频繁插入和更新的话，因为涉及到数据全表锁，效率并不高  
  2、保存了数据库行数，执行count时，不需要扫描全表；  
  3、不支持数据库事务；  
  4、不支持行级锁和外键；  
  5、不支持故障恢复。  
  6、支持全文检索FullText，压缩索引。  
  建议使用场景：  
  1、做很多count计算的，（如果count计算后面有where还是会全表扫描）  
  2、插入和更新较少，查询比较频繁的
* InnoDB：  
  在Mysql8里，默认存储引擎改成了InnoDB。  
  特点  
  1、支持事务处理、ACID事务特性  
  2、实现了SQL标准的四种隔离级别  
  3、支持行级锁和外键约束  
  4、可以利用事务日志进行数据恢复  
  5、不支持FullText类型的索引，没有保存数据库行数，计算count(\*)需要全局扫描  
  6、支持自动增加列属性auto\_increment  
  7、最后也是非常重要的一点：InnerDB是为了处理大量数据时的最大性能设计，其CPU效率可能是其他基于磁盘的关系型数据库所不能匹敌的。  
  建议使用场景  
  1、可靠性高或者必须要求事务处理  
  2、表更新和查询相当的频繁，并且表锁定的机会比较大的情况下，指定InnerDB存储引擎。

**2.2 表和数据等在Mysql中是如何存储的**

我们新建一个数据库mds\_demo，里面有两张表：order\_info,user

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2U5dmlja0xkNnVVcEZpYnN5aWFCb2lhMEVEdGVtTUNId3k4Q0lheWdkNHFrNHd0WU9rcExIbWF0ZTZ3LzY0MA?x-oss-process=image/format,png)

我们找到mysql存放数据的data目录，存在一个mds\_demo的文件夹，同时我们也找到了order\_info和user的文件。

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VjWlRtZUVZR3RuSVlUc3dzNXNHUlJVa1NmcVNaRmZaMU90OXZkYWpCUG9wSENSS3hXMlRTN2cvNjQw?x-oss-process=image/format,png)

为什么两张表产生了不同的文件呢？原因很简单，因为创建这两张表时使用了不同的引擎

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2V1cEs2MWtnbmw4OVU1S0FMT2lhekpXQUVRblkxTWljRmRvZWlhd08xcmowbnNzTGF3a20yemNYZkEvNjQw?x-oss-process=image/format,png)

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VNY01VS1gwMGlhOG5JdmhzWGxBeWZJZjV4UnZzUEUxMmJUcEhHZXhVT3lDYklBcWV6dkRqanVRLzY0MA?x-oss-process=image/format,png)

* MyISAM引擎在创建表的时候，会创建三个文件  
  .MYD文件：存放表里的数据  
  .MYI文件：存放索引数据  
  .sdi文件： Serialized Dictionary Information的缩写。在Mysql5里没有sdi文件，但会有一个FRM文件，用户存放表结构信息。在MySQL8.0中重新设计了数据字典，改为sdi。  
  MyISAM的索引和数据是分开的，并且索引是有压缩的，所以存储文件就会小很多，MyISAM应对错误码导致的数据恢复的速度很快。
* InnerDB引擎在创建表的时候，只有1个文件.ibd，即存放了索引又存放了文件，参见B+Tree。所以它也被称之为聚集索引，即叶子节点包含完整的索引和数据，对应的MyISAM为非聚集索引。  
  补充说明一下：存储引擎是针对表的，而不是针对数据库，同一个库的不同的表可以使用不同的引擎。

**2.3 为什么InnoDB必须要有主键，并且推荐使用整型的自增主键？**

通过上面的讲解这个问题其实已经很清楚了，为了满足MySQL的索引数据结构B+树的特性，必须要有索引作为主键，可以有效提高查询效率。有的童鞋可能会说我创建表的时候可以没有主键啊，这个其实和Oracle的rownum一样，如果不指定主键，InnoDB会从插入的数据中找出不重复的一列作为主键索引，如果没找到不重复的一列，InnoDB会在后台增加一列rowId做为主键索引。所以不如我们自己创建一个主键。

将索引的数据类型是设置为整型，一来占有的磁盘空间或内存空间更少，另一方面整型相对于字符串比较更快速，而字符串需要先转换为ASCII码然后再一个个进行比较的。

参见B+树的图它本质上是多路多叉树，如果主键索引不是自增的，那么后续插入的索引就会引起B+树的其他节点的分裂和重新平衡，影响数据插入的效率，如果是自增主键，只用在尾节点做增加就可以。

最后特别强调一点：不管当前是否有性能要求或者数据量多大，千万不要使用UUID作为索引。

**2.4 为什么Mysql存储引擎中默认每个页的大小为16KB？**

假设我们一行数据大小为1K，那么一页就能存16条数据，包含指针+数据+索引。假设一行数据大小为1K，那么一页（1个叶子节点）就能存16条数据；对于非叶子节点，假设ID为bigint类型那么长度为8B，指针大小在Innodb源码中为6B，一共就是14B，那么一页里就可以存储16K/14=1170个(主键+指针)，这样一颗高度为3的B+树能存储的数据为：1170*1170*16=2千万级别。所以我们前面1000万的数据只有0.02s。

**2.5 HASH算法的使用场景**

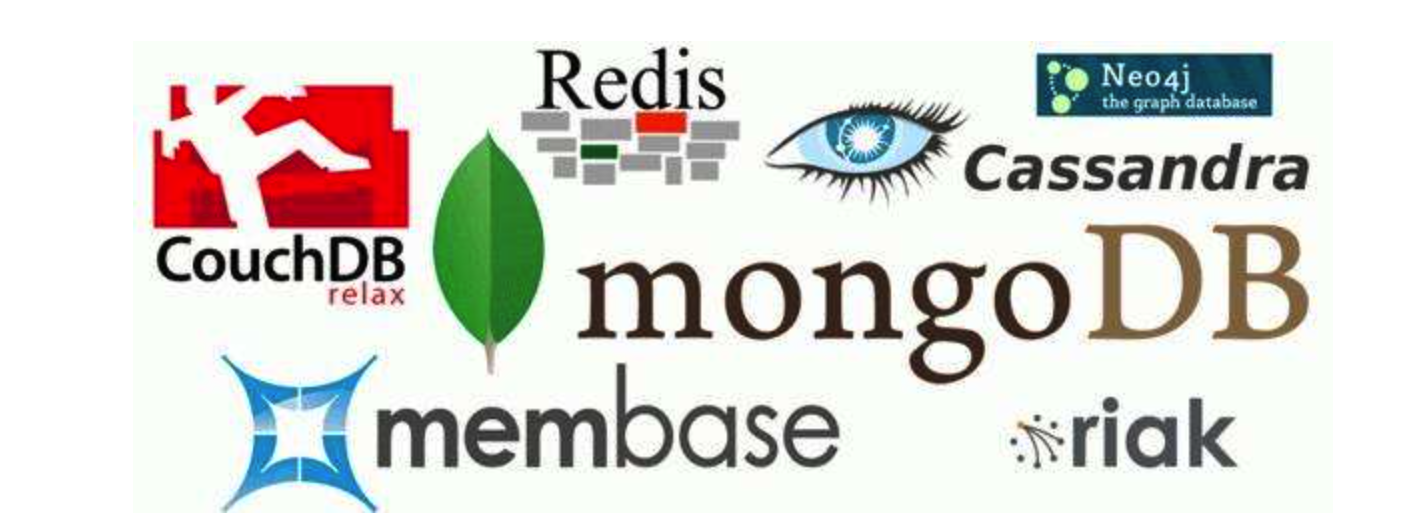
[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9tbWJpei5xcGljLmNuL21tYml6L0RrQTBWWXJPSHZ2c2xHNmljbHpGaWNYUVFtY3M2N1ZCc2VleVJyeGVyVFVZVGtiemRnOEoxaWN1dWJvdEVsRkVZNWxJaWNGY1E3OWp4MEFwRzhTMWRpYmJuc2cvNjQw?x-oss-process=image/format,png)

Hash算法是一种散列算法，就是计算出某个字段的hash，然后存放在对应的地址中，查找数据时只需要1次定位而不像BTree那样从根节点找到叶子节点经过多次IO操作，所以查询效率非常地高。但同样也有很多的弊端，讲一下最重要的两条。

1、很明显hash只支持=、IN等查询，而不支持范围查询  
2、 Hash 索引在任何时候都不能避免表扫描。

所以使用时务必注意。

**53: 非关系型数据库和关系型数据库区别，优势比较**

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5MzgzNzYyNy0xNTczOTI4NDcwLnBuZw?x-oss-process=image/format,png)

非关系型数据库（感觉翻译不是很准确）称为 **NoSQL**，也就是 Not Only SQL，不仅仅是 SQL。非关系型数据库不需要写一些复杂的 SQL 语句，其内部存储方式是以 **key-value** 的形式存在可以把它想象成电话本的形式，每个人名（key）对应电话（value）。常见的非关系型数据库主要有 **Hbase、Redis、MongoDB** 等。非关系型数据库不需要经过 SQL 的重重解析，所以性能很高；非关系型数据库的可扩展性比较强，数据之间没有耦合性，遇见需要新加字段的需求，就直接增加一个 key-value 键值对即可。

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5Mzg0NzM4Ny0zNTgwMTgzOTQucG5n?x-oss-process=image/format,png)

关系型数据库以**表格**的形式存在，以**行和列**的形式存取数据，关系型数据库这一系列的行和列被称为表，无数张表组成了**数据库**，常见的关系型数据库有 **Oracle、DB2、Microsoft SQL Server、MySQL**等。关系型数据库能够支持复杂的 SQL 查询，能够体现出数据之间、表之间的关联关系；关系型数据库也支持事务，便于提交或者回滚。

它们之间的劣势都是基于对方的优势来满足的。

**54: MySQL 事务四大特性**

一说到 MySQL 事务，你肯定能想起来四大特性：**原子性**、**一致性**、**隔离性**、**持久性**，下面再对这事务的四大特性做一个描述

* **原子性(Atomicity)**: 原子性指的就是 MySQL 中的包含事务的操作要么**全部成功**、要么全部**失败回滚**，因此事务的操作如果成功就必须要全部应用到数据库，如果操作失败则不能对数据库有任何影响。

这里涉及到一个概念，什么是 MySQL 中的事务？

事务是一组操作，组成这组操作的各个单元，要不全都成功要不全都失败，这个特性就是事务。

在 MySQL 中，事务是在引擎层实现的，只有使用 **innodb** 引擎的数据库或表才支持事务。

* **一致性(Consistency)**：一致性指的是一个事务在执行前后其状态一致。比如 A 和 B 加起来的钱一共是 1000 元，那么不管 A 和 B 之间如何转账，转多少次，事务结束后两个用户的钱加起来还得是 1000，这就是事务的一致性。
* **持久性(Durability)**: 持久性指的是一旦事务提交，那么发生的改变就是永久性的，即使数据库遇到特殊情况比如故障的时候也不会产生干扰。
* **隔离性(Isolation)**：隔离性需要重点说一下，当多个事务同时进行时，就有可能出现**脏读(dirty read)**、**不可重复读(non-repeatable read)**、**幻读(phantom read)** 的情况，为了解决这些并发问题，提出了隔离性的概念。

脏读：事务 A 读取了事务 B 更新后的数据，但是事务 B 没有提交，然后事务 B 执行回滚操作，那么事务 A 读到的数据就是脏数据

不可重复读：事务 A 进行多次读取操作，事务 B 在事务 A 多次读取的过程中执行更新操作并提交，提交后事务 A 读到的数据不一致。

幻读：事务 A 将数据库中所有学生的成绩由 A -> B，此时事务 B 手动插入了一条成绩为 A 的记录，在事务 A 更改完毕后，发现还有一条记录没有修改，那么这种情况就叫做出现了幻读。

SQL的隔离级别有四种，它们分别是**读未提交(read uncommitted)**、**读已提交(read committed)**、**可重复读(repetable read)** 和 **串行化(serializable)**。下面分别来解释一下。

读未提交：读未提交指的是一个事务在提交之前，它所做的修改就能够被其他事务所看到。

读已提交：读已提交指的是一个事务在提交之后，它所做的变更才能够让其他事务看到。

可重复读：可重复读指的是一个事务在执行的过程中，看到的数据是和启动时看到的数据是一致的。未提交的变更对其他事务不可见。

串行化：顾名思义是对于同一行记录，**写**会加**写锁**，**读**会加**读锁**。当出现读写锁冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行。

这四个隔离级别可以解决脏读、不可重复读、幻象读这三类问题。总结如下

| **事务隔离级别** | **脏读** | **不可重复读** | **幻读** |
| --- | --- | --- | --- |
| 读未提交 | 允许 | 允许 | 允许 |
| 读已提交 | 不允许 | 允许 | 允许 |
| 可重复读 | 不允许 | 不允许 | 允许 |
| 串行化 | 不允许 | 不允许 | 不允许 |

其中隔离级别由低到高是：读未提交 < 读已提交 < 可重复读 < 串行化

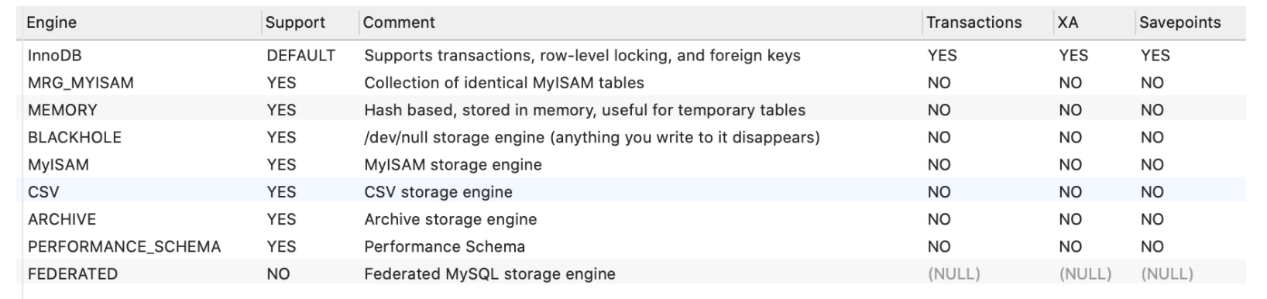
隔离级别越高，越能够保证数据的完整性和一致性，但是对并发的性能影响越大。大多数数据库的默认级别是**读已提交(Read committed)**，比如 Sql Server、Oracle ，但是 MySQL 的默认隔离级别是 **可重复读(repeatable-read)**。

**55: MySQL 常见存储引擎的区别**

MySQL 常见的存储引擎，可以使用

SHOW ENGINES

命令，来列出所有的存储引擎

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5MzkwNzIwNi0xMjA4ODU2OTk3LnBuZw?x-oss-process=image/format,png)

可以看到，InnoDB 是 MySQL 默认支持的存储引擎，支持**事务、行级锁定和外键**。

**MyISAM 存储引擎的特点**

在 5.1 版本之前，MyISAM 是 MySQL 的默认存储引擎，MyISAM 并发性比较差，使用的场景比较少，主要特点是

* 不支持**事务**操作，ACID 的特性也就不存在了，这一设计是为了性能和效率考虑的。
* 不支持**外键**操作，如果强行增加外键，MySQL 不会报错，只不过外键不起作用。
* MyISAM 默认的锁粒度是**表级锁**，所以并发性能比较差，加锁比较快，锁冲突比较少，不太容易发生死锁的情况。
* MyISAM 会在磁盘上存储三个文件，文件名和表名相同，扩展名分别是 **.frm(存储表定义)**、**.MYD(MYData,存储数据)**、**MYI(MyIndex,存储索引)**。这里需要特别注意的是 MyISAM 只缓存**索引文件**，并不缓存数据文件。
* MyISAM 支持的索引类型有 **全局索引(Full-Text)**、**B-Tree 索引**、**R-Tree 索引**

Full-Text 索引：它的出现是为了解决针对文本的模糊查询效率较低的问题。

B-Tree 索引：所有的索引节点都按照平衡树的数据结构来存储，所有的索引数据节点都在叶节点

R-Tree索引：它的存储方式和 B-Tree 索引有一些区别，主要设计用于存储空间和多维数据的字段做索引,目前的 MySQL 版本仅支持 geometry 类型的字段作索引，相对于 BTREE，RTREE 的优势在于范围查找。

* 数据库所在主机如果宕机，MyISAM 的数据文件容易损坏，而且难以恢复。
* 增删改查性能方面：SELECT 性能较高，适用于查询较多的情况

**InnoDB 存储引擎的特点**

自从 MySQL 5.1 之后，默认的存储引擎变成了 InnoDB 存储引擎，相对于 MyISAM，InnoDB 存储引擎有了较大的改变，它的主要特点是

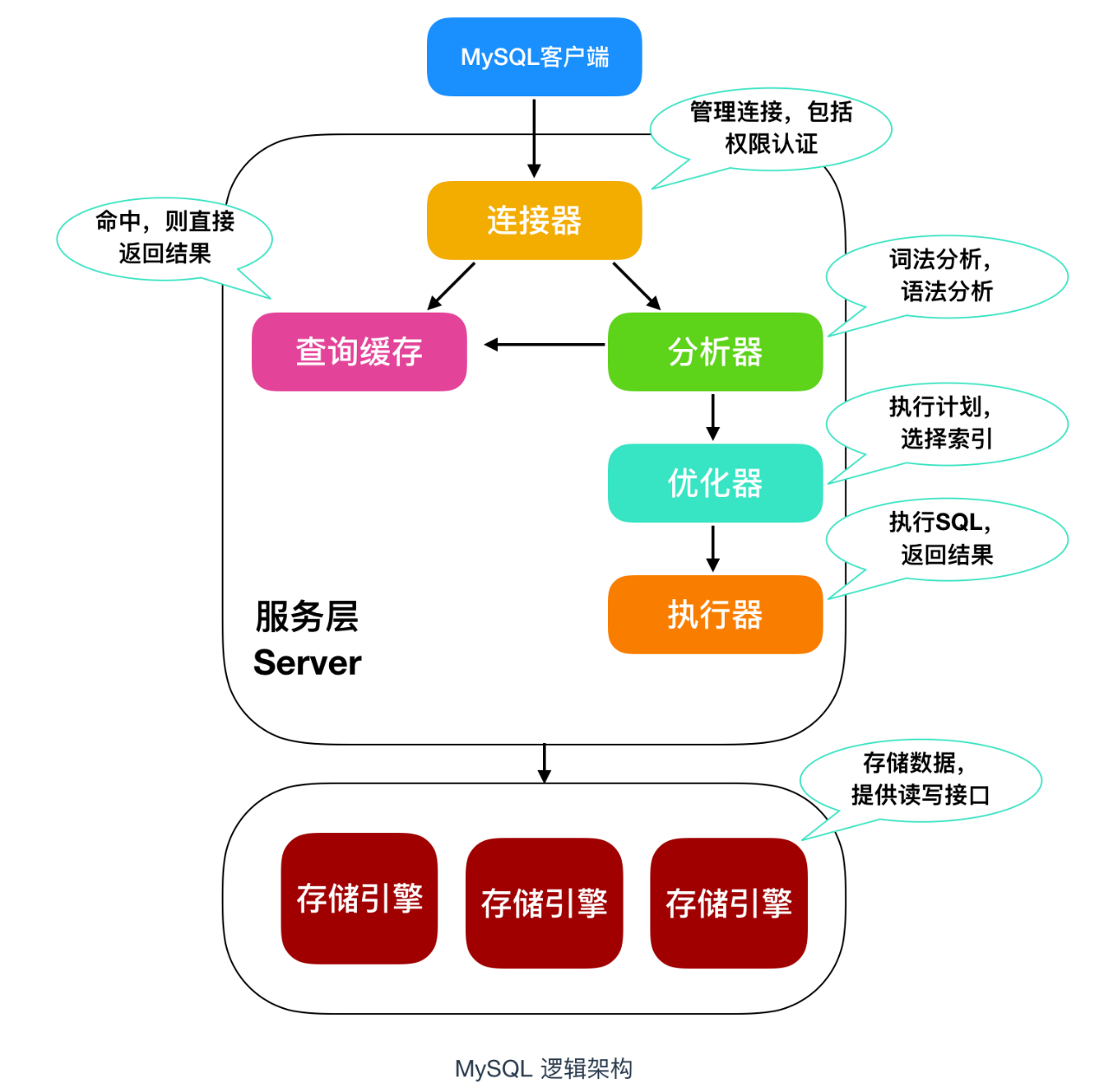
* 支持事务操作，具有事务 ACID 隔离特性，默认的隔离级别是**可重复读(repetable-read)**、通过MVCC（并发版本控制）来实现的。能够解决**脏读**和**不可重复读**的问题。
* InnoDB 支持外键操作。
* InnoDB 默认的锁粒度**行级锁**，并发性能比较好，会发生死锁的情况。
* 和 MyISAM 一样的是，InnoDB 存储引擎也有 **.frm文件存储表结构** 定义，但是不同的是，InnoDB 的表数据与索引数据是存储在一起的，都位于 B+ 数的叶子节点上，而 MyISAM 的表数据和索引数据是分开的。
* InnoDB 有安全的日志文件，这个日志文件用于恢复因数据库崩溃或其他情况导致的数据丢失问题，保证数据的一致性。
* InnoDB 和 MyISAM 支持的索引类型相同，但具体实现因为文件结构的不同有很大差异。
* 增删改查性能方面，果执行大量的增删改操作，推荐使用 InnoDB 存储引擎，它在删除操作时是对行删除，不会重建表。

**MyISAM 和 InnoDB 存储引擎的对比**

* **锁粒度方面**：由于锁粒度不同，InnoDB 比 MyISAM 支持更高的并发；InnoDB 的锁粒度为行锁、MyISAM 的锁粒度为表锁、行锁需要对每一行进行加锁，所以锁的开销更大，但是能解决脏读和不可重复读的问题，相对来说也更容易发生死锁
* **可恢复性上**：由于 InnoDB 是有事务日志的，所以在产生由于数据库崩溃等条件后，可以根据日志文件进行恢复。而 MyISAM 则没有事务日志。
* **查询性能上**：MyISAM 要优于 InnoDB，因为 InnoDB 在查询过程中，是需要维护数据缓存，而且查询过程是先定位到行所在的数据块，然后在从数据块中定位到要查找的行；而 MyISAM 可以直接定位到数据所在的内存地址，可以直接找到数据。
* **表结构文件上**： MyISAM 的表结构文件包括：.frm(表结构定义),.MYI(索引),.MYD(数据)；而 InnoDB 的表数据文件为:.ibd和.frm(表结构定义)；

**56:MySQL 基础架构**

这道题应该从 MySQL 架构来理解，我们可以把 MySQL 拆解成几个零件，如下图所示

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDQyMDQ4Ny0xNzI5ODk0OTQyLnBuZw?x-oss-process=image/format,png)

大致上来说，MySQL 可以分为 **Server**层和 **存储引擎**层。

Server 层包括连接器、查询缓存、分析器、优化器、执行器，包括大多数 MySQL 中的核心功能，所有跨存储引擎的功能也在这一层实现，包括 **存储过程、触发器、视图等**。

存储引擎层包括 MySQL 常见的存储引擎，包括 **MyISAM、InnoDB 和 Memory** 等，最常用的是 InnoDB，也是现在 MySQL 的默认存储引擎。存储引擎也可以在创建表的时候手动指定，比如下面

CREATE TABLE t (i INT) ENGINE = <Storage Engine>;

然后我们就可以探讨 MySQL 的执行过程了

**连接器**

首先需要在 MySQL 客户端登陆才能使用，所以需要一个**连接器**来连接用户和 MySQL 数据库，我们一般是使用

mysql -u 用户名 -p 密码

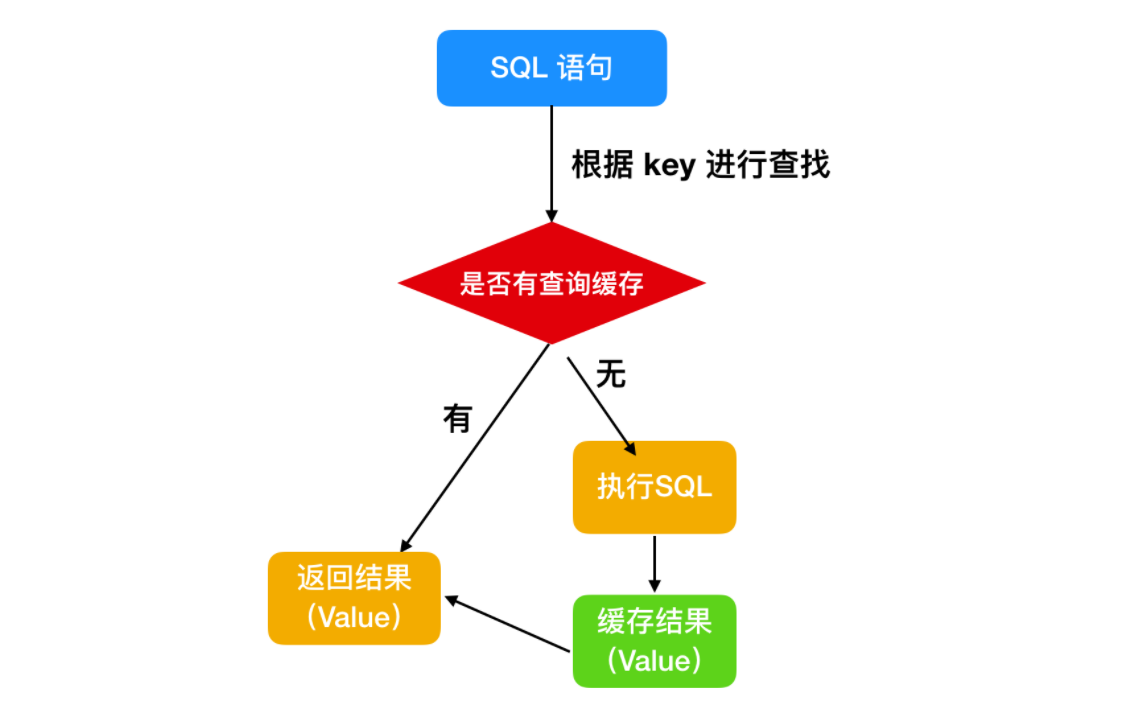
来进行 MySQL 登陆，和服务端建立连接。在完成 **TCP 握手** 后，连接器会根据你输入的用户名和密码验证你的登录身份。如果用户名或者密码错误，MySQL 就会提示 **Access denied for user**，来结束执行。如果登录成功后，MySQL 会根据权限表中的记录来判定你的权限。

**查询缓存**

连接完成后，你就可以执行 SQL 语句了，这行逻辑就会来到第二步：查询缓存。

MySQL 在得到一个执行请求后，会首先去 **查询缓存** 中查找，是否执行过这条 SQL 语句，之前执行过的语句以及结果会以 **key-value** 对的形式，被直接放在内存中。key 是查询语句，value 是查询的结果。如果通过 key 能够查找到这条 SQL 语句，就直接返回 SQL 的执行结果。

如果语句不在查询缓存中，就会继续后面的执行阶段。执行完成后，执行结果就会被放入查询缓存中。可以看到，如果查询命中缓存，MySQL 不需要执行后面的复杂操作，就可以直接返回结果，效率会很高。

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDE1Mjk0MC0yMDgzMDkyODk1LnBuZw?x-oss-process=image/format,png)

**但是查询缓存不建议使用**

为什么呢？因为只要在 MySQL 中对某一张表执行了更新操作，那么所有的查询缓存就会失效，对于更新频繁的数据库来说，查询缓存的命中率很低。

**分析器**

如果没有命中查询，就开始执行真正的 SQL 语句。

* 首先，MySQL 会根据你写的 SQL 语句进行解析，分析器会先做 **词法分析**，你写的 SQL 就是由多个字符串和空格组成的一条 SQL 语句，MySQL 需要识别出里面的字符串是什么，代表什么。
* 然后进行 **语法分析**，根据词法分析的结果， 语法分析器会根据语法规则，判断你输入的这个 SQL 语句是否满足 MySQL 语法。如果 SQL 语句不正确，就会提示 **You have an error in your SQL syntax**

**优化器**

经过分析器的词法分析和语法分析后，你这条 SQL 就**合法**了，MySQL 就知道你要做什么了。但是在执行前，还需要进行优化器的处理，优化器会判断你使用了哪种索引，使用了何种连接，优化器的作用就是确定效率最高的执行方案。

**执行器**

MySQL 通过分析器知道了你的 SQL 语句是否合法，你想要做什么操作，通过优化器知道了该怎么做效率最高，然后就进入了执行阶段，开始执行这条 SQL 语句

在执行阶段，MySQL 首先会判断你有没有执行这条语句的权限，没有权限的话，就会返回没有权限的错误。如果有权限，就打开表继续执行。打开表的时候，执行器就会根据表的引擎定义，去使用这个引擎提供的接口。对于有索引的表，执行的逻辑也差不多。

至此，MySQL 对于一条语句的执行过程也就完成了。

**57: SQL 的执行顺序**

我们在编写一个查询语句的时候

SELECT DISTINCT

< select\_list >

FROM

< left\_table > < join\_type >

JOIN < right\_table > ON < join\_condition >

WHERE

< where\_condition >

GROUP BY

< group\_by\_list >

HAVING

< having\_condition >

ORDER BY

< order\_by\_condition >

LIMIT < limit\_number >

它的执行顺序你知道吗？这道题就给你一个回答。

**FROM 连接**

首先，对 SELECT 语句执行查询时，对**FROM** 关键字两边的表执行连接，会形成**笛卡尔积**，这时候会产生一个**虚表VT1(virtual table)**

首先先来解释一下什么是**笛卡尔积**

现在我们有两个集合 A = {0,1} , B = {2,3,4}

那么，集合 A \* B 得到的结果就是

A \* B = {(0,2)、(1,2)、(0,3)、(1,3)、(0,4)、(1,4)};

B \* A = {(2,0)、{2,1}、{3,0}、{3,1}、{4,0}、(4,1)};

上面 A \* B 和 B \* A 的结果就可以称为两个集合相乘的 **笛卡尔积**

我们可以得出结论，A 集合和 B 集合相乘，包含了集合 A 中的元素和集合 B 中元素之和，也就是 A 元素的个数 \* B 元素的个数

再来解释一下什么是虚表

在 MySQL 中，有三种类型的表

一种是**永久表**，永久表就是创建以后用来长期保存数据的表

一种是**临时表**，临时表也有两类，一种是和永久表一样，只保存临时数据，但是能够长久存在的；还有一种是临时创建的，SQL 语句执行完成就会删除。

一种是**虚表**，虚表其实就是**视图**，数据可能会来自多张表的执行结果。

**ON 过滤**

然后对 FROM 连接的结果进行 ON 筛选，创建 VT2，把符合记录的条件存在 VT2 中。

**JOIN 连接**

第三步，如果是 **OUTER JOIN(left join、right join)** ，那么这一步就将添加外部行，如果是 left join 就把 ON 过滤条件的左表添加进来，如果是 right join ，就把右表添加进来，从而生成新的虚拟表 VT3。

**WHERE 过滤**

第四步，是执行 WHERE 过滤器，对上一步生产的虚拟表引用 WHERE 筛选，生成虚拟表 VT4。

WHERE 和 ON 的区别

* 如果有外部列，ON 针对过滤的是关联表，主表(保留表)会返回所有的列;
* 如果没有添加外部列，两者的效果是一样的;

应用

* 对主表的过滤应该使用 WHERE;
* 对于关联表，先条件查询后连接则用 ON，先连接后条件查询则用 WHERE;

**GROUP BY**

根据 group by 字句中的列，会对 VT4 中的记录进行分组操作，产生虚拟机表 VT5。果应用了group by，那么后面的所有步骤都只能得到的 VT5 的列或者是聚合函数（count、sum、avg等）。

**HAVING**

紧跟着 GROUP BY 字句后面的是 HAVING，使用 HAVING 过滤，会把符合条件的放在 VT6

**SELECT**

第七步才会执行 SELECT 语句，将 VT6 中的结果按照 SELECT 进行刷选，生成 VT7

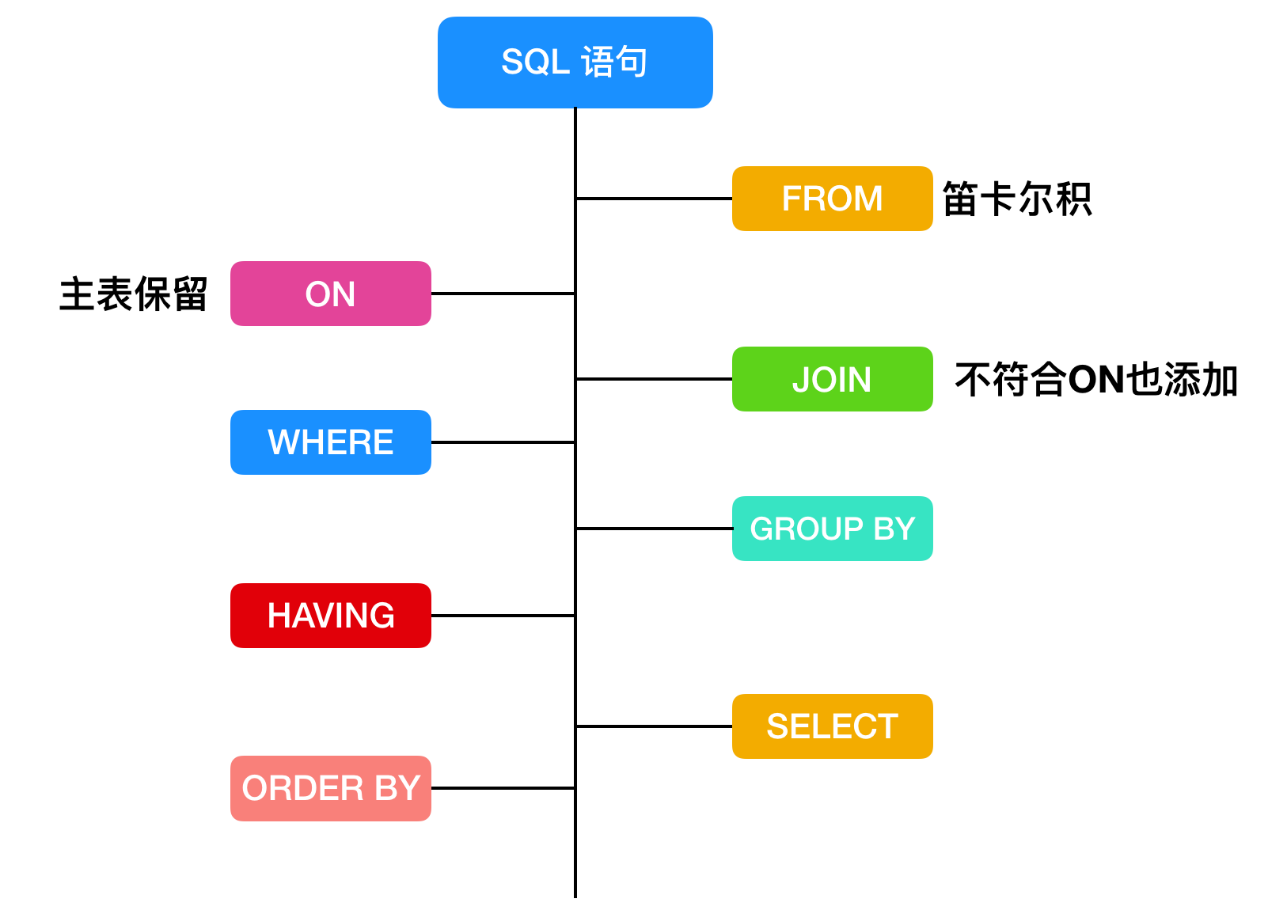
**DISTINCT**

在第八步中，会对 TV7 生成的记录进行去重操作，生成 VT8。事实上如果应用了 group by 子句那么 distinct 是多余的，原因同样在于，分组的时候是将列中唯一的值分成一组，同时只为每一组返回一行记录，那么所以的记录都将是不相同的。

**ORDER BY**

应用 order by 子句。按照 order\_by\_condition 排序 VT8，此时返回的一个游标，而不是虚拟表。sql 是基于集合的理论的，集合不会预先对他的行排序，它只是成员的逻辑集合，成员的顺序是无关紧要的。

SQL 语句执行的过程如下

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDIyNDI1OS01ODk2MTg5OTQucG5n?x-oss-process=image/format,png)

**58:什么是临时表，何时删除临时表**

什么是临时表？MySQL 在执行 SQL 语句的过程中，通常会临时创建一些**存储中间结果集**的表，临时表只对当前连接可见，在连接关闭时，临时表会被删除并释放所有表空间。

临时表分为两种：一种是**内存临时表**，一种是**磁盘临时表**，什么区别呢？内存临时表使用的是 MEMORY 存储引擎，而临时表采用的是 MyISAM 存储引擎。

MEMORY 存储引擎：**memory** 是 MySQL 中一类特殊的存储引擎，它使用存储在内容中的内容来创建表，而且**数据全部放在内存中**。每个基于 MEMORY 存储引擎的表实际对应一个磁盘文件。该文件的文件名与表名相同，类型为 **frm** 类型。而其数据文件，都是存储在内存中，这样有利于数据的快速处理，提高整个表的效率。MEMORY 用到的很少，因为它是把数据存到内存中，如果内存出现异常就会影响数据。如果重启或者关机，所有数据都会消失。因此，基于 MEMORY 的表的生命周期很短，一般是一次性的。

MySQL 会在下面这几种情况产生临时表

* 使用 UNION 查询：UNION 有两种，一种是**UNION** ，一种是 **UNION ALL** ，它们都用于联合查询；区别是 使用 UNION 会去掉两个表中的重复数据，相当于对结果集做了一下**去重(distinct)**。使用 UNION ALL，则不会排重，返回所有的行。使用 UNION 查询会产生临时表。
* 使用 **TEMPTABLE 算法**或者是 UNION 查询中的视图。TEMPTABLE 算法是一种创建临时表的算法，它是将结果放置到临时表中，意味这要 MySQL 要先创建好一个临时表，然后将结果放到临时表中去，然后再使用这个临时表进行相应的查询。
* ORDER BY 和 GROUP BY 的子句不一样时也会产生临时表。
* DISTINCT 查询并且加上 ORDER BY 时；
* SQL中用到 SQL\_SMALL\_RESULT 选项时；如果查询结果比较小的时候，可以加上 SQL\_SMALL\_RESULT 来优化，产生临时表
* FROM 中的子查询；
* EXPLAIN 查看执行计划结果的 Extra 列中，如果使用 **Using Temporary** 就表示会用到临时表。

**59:MySQL 常见索引类型**

索引是存储在一张表中特定列上的**数据结构**，索引是在列上创建的。并且，索引是一种数据结构。

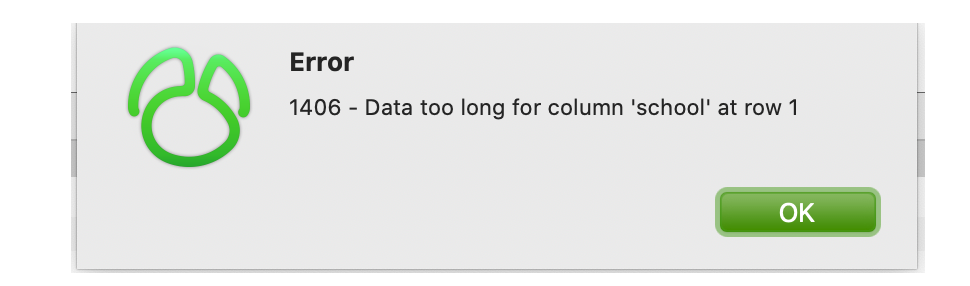
在 MySQL 中，主要有下面这几种索引

* **全局索引(FULLTEXT)**：全局索引，目前只有 MyISAM 引擎支持全局索引，它的出现是为了解决针对文本的模糊查询效率较低的问题。
* **哈希索引(HASH)**：哈希索引是 MySQL 中用到的唯一 key-value 键值对的数据结构，很适合作为索引。HASH 索引具有一次定位的好处，不需要像树那样逐个节点查找，但是这种查找适合应用于查找单个键的情况，对于范围查找，HASH 索引的性能就会很低。
* **B-Tree 索引**：B 就是 Balance 的意思，BTree 是一种平衡树，它有很多变种，最常见的就是 B+ Tree，它被 MySQL 广泛使用。
* **R-Tree 索引**：R-Tree 在 MySQL 很少使用，仅支持 geometry 数据类型，支持该类型的存储引擎只有MyISAM、BDb、InnoDb、NDb、Archive几种，相对于 B-Tree 来说，R-Tree 的优势在于范围查找。

**60:varchar 和 char 的区别和使用场景**

MySQL 中没有 nvarchar 数据类型，所以直接比较的是 varchar 和 char 的区别

**char** ：表示的是**定长**的字符串，当你输入小于指定的数目，比如你指定的数目是 **char(6)**，当你输入小于 6 个字符的时候，char 会在你最后一个字符后面补空值。当你输入超过指定允许最大长度后，MySQL 会报错

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDIzNTM3OS04OTY3MTEyMzkucG5n?x-oss-process=image/format,png)

**varchar**： varchar 指的是长度为 n 个字节的可变长度，并且是**非Unicode**的字符数据。n 的值是介于 1 - 8000 之间的数值。存储大小为实际大小。

Unicode 是一种字符编码方案，它为每种语言中的每个字符都设定了统一唯一的二进制编码，以实现跨语言、跨平台进行文本转换、处理的要求

使用 char 存储定长的数据非常方便、char 检索效率高，无论你存储的数据是否到了 10 个字节，都要去占用 10 字节的空间

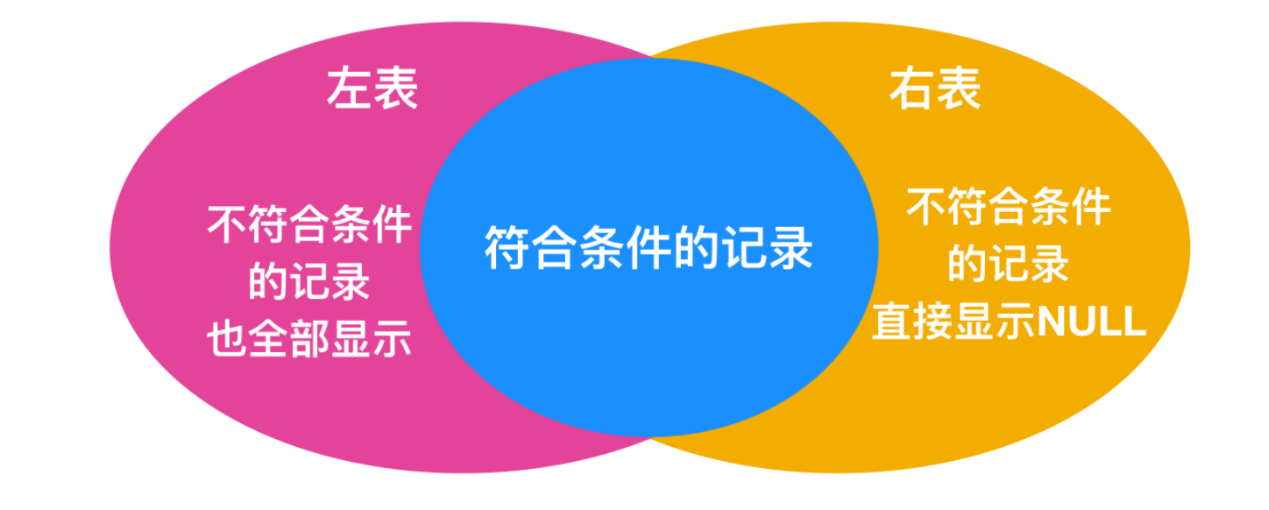
使用 varchar 可以存储变长的数据，但存储效率没有 char 高。

**60:什么是 内连接、外连接、交叉连接、笛卡尔积**

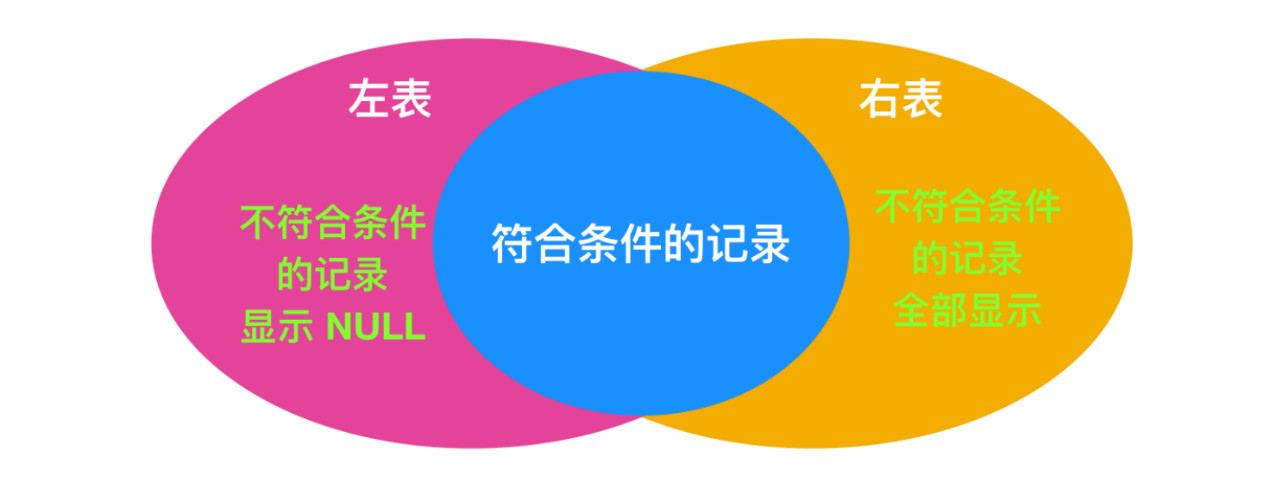
连接的方式主要有三种：**外连接、内链接、交叉连接**

* **外连接(OUTER JOIN)**：外连接分为三种，分别是**左外连接(LEFT OUTER JOIN 或 LEFT JOIN)** 、**右外连接(RIGHT OUTER JOIN 或 RIGHT JOIN)** 、**全外连接(FULL OUTER JOIN 或 FULL JOIN)**

左外连接：又称为左连接，这种连接方式会显示左表不符合条件的数据行，右边不符合条件的数据行直接显示 NULL

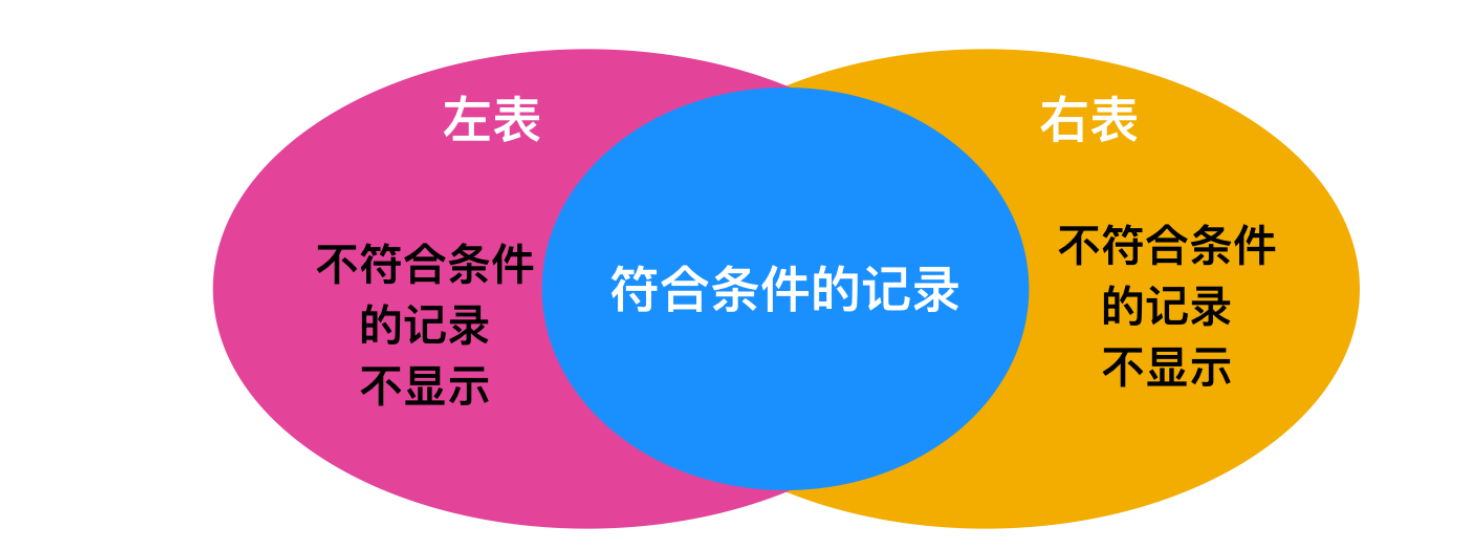
[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDI0NDYyNC04MzAxMTM1NTEucG5n?x-oss-process=image/format,png)

右外连接：也被称为右连接，他与左连接相对，这种连接方式会显示右表不符合条件的数据行，左表不符合条件的数据行直接显示 NULL

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDI1MjE3Mi0xODYzMzYzODE5LnBuZw?x-oss-process=image/format,png)

**MySQL 暂不支持全外连接**

* **内连接(INNER JOIN)**：结合两个表中相同的字段，返回关联字段相符的记录。

[](https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWcyMDIwLmNuYmxvZ3MuY29tL2Jsb2cvMTUxNTExMS8yMDIwMDQvMTUxNTExMS0yMDIwMDQxODA5NDMwMDI1Ny0xMTM0Mjk2MDgxLnBuZw?x-oss-process=image/format,png)

* **笛卡尔积(Cartesian product)**： 我在上面提到了笛卡尔积，为了方便，下面再列出来一下。

现在我们有两个集合 A = {0,1} , B = {2,3,4}

那么，集合 A \* B 得到的结果就是

A \* B = {(0,2)、(1,2)、(0,3)、(1,3)、(0,4)、(1,4)};

B \* A = {(2,0)、{2,1}、{3,0}、{3,1}、{4,0}、(4,1)};

上面 A \* B 和 B \* A 的结果就可以称为两个集合相乘的 **笛卡尔积**

我们可以得出结论，A 集合和 B 集合相乘，包含了集合 A 中的元素和集合 B 中元素之和，也就是 A 元素的个数 \* B 元素的个数

* 交叉连接的原文是**Cross join** ，就是笛卡尔积在 SQL 中的实现，SQL中使用关键字**CROSS JOIN**来表示交叉连接，在交叉连接中，随便增加一个表的字段，都会对结果造成很大的影响。
* SELECT \* FROM t\_Class a CROSS JOIN t\_Student b WHERE a.classid=b.classid

或者不用 CROSS JOIN，直接用 FROM 也能表示交叉连接的效果

SELECT \* FROM t\_Class a ,t\_Student b WHERE a.classid=b.classid

如果表中字段比较多，不适宜用交叉连接，交叉连接的效率比较差。

* 全连接：全连接也就是 **full join**，MySQL 中不支持全连接，但是可以使用其他连接查询来模拟全连接，可以使用 **UNION** 和 **UNION ALL** 进行模拟。例如
* (select colum1,colum2...columN from tableA ) union (select colum1,colum2...columN from tableB )
* 或
* (select colum1,colum2...columN from tableA ) union all (select colum1,colum2...columN from tableB )；

使用 UNION 和 UNION ALL 的注意事项

通过 union 连接的 SQL 分别单独取出的列数必须相同

使用 union 时，多个相等的行将会被合并，由于合并比较耗时，一般不直接使用 union 进行合并，而是通常采用 union all 进行合并

**61:谈谈 SQL 优化的经验**

* 查询语句无论是使用哪种判断条件 **等于、小于、大于**， **WHERE** 左侧的条件查询字段不要使用函数或者表达式
* 使用 **EXPLAIN** 命令优化你的 SELECT 查询，对于复杂、效率低的 sql 语句，我们通常是使用 explain sql 来分析这条 sql 语句，这样方便我们分析，进行优化。
* 当你的 SELECT 查询语句只需要使用一条记录时，要使用 **LIMIT 1**
* 不要直接使用 **SELECT \***，而应该使用具体需要查询的表字段，因为使用 EXPLAIN 进行分析时，SELECT \* 使用的是全表扫描，也就是 **type = all**。
* 为每一张表设置一个 ID 属性
* 避免在 **WHERE** 字句中对字段进行 **NULL** 判断
* 避免在 **WHERE** 中使用 **!=** 或 **<>** 操作符
* 使用 **BETWEEN AND** 替代 **IN**
* 为搜索字段创建索引
* 选择正确的存储引擎，InnoDB 、MyISAM 、MEMORY 等
* 使用 **LIKE %abc%** 不会走索引，而使用 **LIKE abc%** 会走索引
* 对于枚举类型的字段(即有固定罗列值的字段)，建议使用**ENUM**而不是**VARCHAR**，如性别、星期、类型、类别等
* 拆分大的 DELETE 或 INSERT 语句
* 选择合适的字段类型，选择标准是 **尽可能小、尽可能定长、尽可能使用整数**。
* 字段设计尽可能使用 **NOT NULL**
* 进行水平切割或者垂直分割

水平分割：通过建立结构相同的几张表分别存储数据

垂直分割：将经常一起使用的字段放在一个单独的表中，分割后的表记录之间是一一对应关系。

**63：说几个常见的 影响MYSQL性能 的案例**

参考文献 ：<https://www.cnblogs.com/zhiqian-ali/p/6336521.html>

**大规模数据导出功能**

相信很多业务都遇到过数据导出，明细展示这方面的需求，sql基本上都是先求一个数据的总和然后，limit n,m分页查询，这样的问题就在于，在扫描前面的数据时是不会有性能问题的，当n值越大，偏移量越多，扫描的数据就越多，这个时候就会产生问题，一个本来不的sql就会变成慢sql，导致DB性能下降。针对这种问题DBA都会建议开发将limit n,m改为id范围的查询，或者进行业务改造对于一些不必要的场景只展示前几百条，只需要进行一次分页即可。

类似sql模式：

select count(\*) from table\_name\_1;

select \* from table\_name\_1 limit n,m;(n值越大性能越差)

建议改造成：

select \* from table\_name\_1 where id>? and id<?

**ERP类系统使用聚合函数或者分组排序**

类似仓库内管理系统会需要展示很多统计信息，很多开发会选择在DB端计算出结果直接展示，问题在于sum，max，min类的聚合函数在DB端执行会消耗到CPU资源，如果这个时候还遇到索引不合理的情况，往往会带来灾难性的后果。这种情况DB端除了增加索引，对CPU的消耗是无法优化的，所以DB性能必然下降。一般这种情况DBA会建议能在程序端计算的就不要放在DB端，或者直接接搜索引擎。

类似sql模式：

select sum(column\_name) as column\_1 from table\_name\_1;

or

select distinct cloumn\_name from table\_name\_1 group by column\_name\_1 order by column\_name\_1;

**错误使用子查询**

在DB端执行去重，**join以及子查询等操作的时候，mysql会自动创建临时表**。

DB自动创建临时表的情况有如下几种

1. Evaluation of UNION statements.

2. Evaluation of some views, such those that use the TEMPTABLE algorithm, UNION, or aggregation.

3. Evaluation of derived tables (subqueries in the FROM clause).（这个是本节关注的重点）

4. Tables created for subquery or semi-join materialization (see Section 8.2.1.18, “Subquery Optimization”).

5. Evaluation of statements that contain an ORDER BY clause and a different GROUP BY clause, or for which the ORDER BY or GROUP BY contains columns from tables other than the first table in the join queue.

6. Evaluation of DISTINCT combined with ORDER BY may require a temporary table.

7. For queries that use the SQL\_SMALL\_RESULT option, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.

8. Evaluation of multiple-table UPDATE statements.

9. Evaluation of GROUP\_CONCAT() or COUNT(DISTINCT) expressions.

在mysql中，对于子查询，外层每执行一次，内层子查询要重复执行一次，所以一般建议用join代替子查询。

**下面举一个子查询引起DB性能问题的例子**

Query1：select count(\*) from wd\_order\_late\_reason\_send wrs left join wd\_order\_detail\_late\_send wds on wrs.store\_code = wds.store\_code;

下面是执行计划：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong> 1. row </strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong>

id: 1

select\_type: SIMPLE

table: wrs

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 836846

Extra: NULL

</strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong> 2. row </strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: wds

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 670612

Extra: Using where; Using join buffer (Block Nested Loop)

Query2：select count(\*) from (select wrs.store\_code from wd\_order\_late\_reason\_send wrs left join wd\_order\_detail\_late\_send wds on wrs.store\_code = wds.store\_code) tb；

执行计划如下

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong> 1. row </strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong>

id: 1

select\_type: PRIMARY

table: <derived2>

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 561198969752

Extra: NULL

</strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong> 2. row </strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong>

id: 2

select\_type: DERIVED

table: wrs

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 836846

Extra: NULL

</strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<strong> 3. row </strong>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 2

select\_type: DERIVED

table: wds

type: ALL

possible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 670612

Extra: Using where; Using join buffer (Block Nested Loop)

这两个sql结果相同，**唯一不同的是第二条sql使用了子查询。通过执行计划可以看出（排除没有索引部分）两个sql最大的差别就是第二个sql有derived table并且rows是561198969752，出现这个数值是因为在select count（\*）\*\*\*\*每次计数的时候子查询的sql都会执行一遍，所以最后是子查询join的笛卡尔积**。因为内存中用于进行join操作的空间有限，这个时候就会使用磁盘空间来创建临时表，所以当第二种sql频繁执行的时候会有磁盘被撑爆的风险。 想要了解更多关于子查询的优化可以参考下面这个链接link

**慢sql**

这里我们所说的慢sql主要指那些由于索引使用不正确或没有使用索引产生的，一般可以通过增加索引。一个合理的索引对一条sql性能的影响是非常巨大的。索引的主要目的是为了减少读取的数据块，也就是我们常说的逻辑读，读取的数据块越少，sql效率越高。另外索引在一定程度上也可以减少CPU的消耗，例如排序，分组，因为索引本来就是有序的。

说到逻辑读，对应的就会有物理读，在mysql服务端是有buffer pool来缓存硬盘中的数据，但是这个buffer pool的大小跟磁盘中数据文件的大小是不等的，往往buffer pool会远远小于磁盘中数据的大小。buffer pool会有一个LRU链表，当从磁盘中加载数据块到内存中（这个就是物理读）发现没有空间的时候会优先覆盖LRU链表中的数据块。当一条sql没有合理的索引需要扫描大量的数据的时候，不光要扫描内存中的许多数据块，还可能需要从磁盘中加载不同不存在的数据块到内存中进行判断，当这种情况频繁发生的时候，sql性能就会急剧下降，因而也影响了DB实例的性能。

以下表格是访问不同存储设备的rt，由此可见一个合理的索引的重要性。

| **类别** | **吞吐量** | **响应时间** |
| --- | --- | --- |
| 访问L1 | Cache | 0.5ns |
| 访问L2 | Cache | 7ns |
| 内存访问 | 800M/s | 100ns |
| 机械盘 | 300M/s | 10ms |
| SSD | 300M/s | 0.1~0.2ms |

**日志刷盘策略不合理**

目前集团mysql大部分使用的都是innodb存储引擎，因此在每条DML语句执行时不光会记如binlog还有记录innodb特有的redo log和undo log。这些日志文件都是先写入内存中然后在刷新到磁盘中。**在server端有两个参数分别控制他们的写入速度。innodb\_flush\_log\_at\_trx\_commit控制redo log写入模式，sync\_binlog控制binlog写入模式。**

[](https://images2015.cnblogs.com/blog/549413/201701/549413-20170121100909828-688547130.png)

通过以上表格可以了解到，在使用线上默认配置的情况下每次commit都会刷redo log到磁盘，也就是说每次写入都会伴随着日志刷盘的操作，需要消耗磁盘IO，所以在高TPS或者类似业务大促情况下，DBA可以调整这个参数，来提升DB支撑TPS的能力。

**BP设置过小**

前面已经提到sql在读写数据的时候不会直接跟磁盘交互，而是先读写内存数据，因为这样最快。但是考虑到成本问题BP（buffer pool）大小是有限的，不可能跟数据文件同等大小，所以如果BP设置不合理就会导致DB的QPS TPS始终上不去。下面我们具体分析一下。

mysql buffer pool中包含undo page，insert buffer page，adaptive hash index，index page，lock info，data dictionary等等DB相关信息，但是这些page都可以归为三类free page,clean page,dirty page.buffer pool中维护了三个链表：free list,dirty list,lru list

* free page:此page未被使用，此种类型page位于free链表中
* clean page:此page被使用，对应数据文件中的一个页面，但是页面没有被修改，此种类型page位于lru链表中
* dirty page:此page被使用，对应数据文件中的一个页面，但是页面被修改过，此种类型page位于lru链表和flush链表中

当BP设置过小的时候，**比如BP 10g 数据文件有200g 这个时候有大量的select或者dml语句，mysql就会频繁的刷新lru list或者dirty list 到磁盘，大部分时间消耗在刷磁盘上，而不是业务sql处理上，这个时候就会导致业务TPS QPS始终上不去，伴随着DB内存命中率降低**。通常这个时候的解决办法是需要DBA调整一下实例BP的大小。

**硬件问题**

就像生活中会有意外一样，在排除了之前那些因素之后，还会存在因为硬件故障或者参数设置不合理导致DB性能抖动的情况，如果不能立即修复，DBA一般只能通过迁移实例的方式来消除影响。