MySQL 有哪些数据类型?

MySQL 支持多种类型，大致可以分为三类：数值、日期/时间和字符串(字符)类型。具体可以看看 《MySQL 数据类型》 文档。

- 正确的使用数据类型，对数据库的优化是非常重要的。

MySQL 中 varchar 与 char 的区别？varchar(50) 中的 50 代表的涵义?

- 1、varchar 与 char 的区别，char 是一种固定长度的类型，varchar 则是一种可变长度的类型。
- 2、varchar(50) 中 50 的涵义最多存放 50 个字符。varchar(50) 和 (200) 存储 hello 所占空间一样，**但后者在排序时会消耗更多内存，因为 `ORDER BY col` 采用 fixed_length 计算 col 长度(memory引擎也一样)**。

所以，实际场景下，选择合适的 varchar 长度还是有必要的。

int(11) 中的 11 代表什么涵义?

int(11) 中的 11 ，不影响字段存储的范围，只影响展示效果。具体可以看看 《MySQL 中 int 长度的意义》 文章。

金额(金钱)相关的数据，选择什么数据类型?

- 方式一，使用 int 或者 bigint 类型。如果需要存储到分的维度，需要 *100 进行放大。
- 方式二，使用 decimal 类型，避免精度丢失。如果使用 Java 语言时，需要使用 BigDecimal 进行对应。

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?

- 一般情况下，我们创建的表的类型是 InnoDB ，如果新增一条记录（不重启 MySQL 的情况下），这条记录的 ID 是18；但是如果重启 MySQL 的话，这条记录的 ID 是 15 。因为 InnoDB 表只把自增主键的最大 ID 记录到内存中，所以重启数据库或者对表 OPTIMIZE 操作，都会使最大 ID 丢失。
- 但是，如果我们使用表的类型是 MyISAM ，那么这条记录的 ID 就是 18 。因为 MyISAM 表会把自增主键的最大 ID 记录到数据文件里面，重启 MYSQL 后，自增主键的最大 ID 也不会丢失。

最后，还可以跟面试官装个 x ，生产数据，不建议进行物理删除记录。

表中有大字段 X(例如：text 类型)，且字段 X 不会经常更新，以读为为主，请问您是选择拆成子表，还是继续放一起？写出您这样选择的理由

- 拆带来的问题：连接消耗 + 存储拆分空间。

如果能容忍拆分带来的空间问题，拆的话最好和经常要查询的表的主键在物理结构上放置在一起(分区) 顺序 IO ，减少连接消耗，最后这是一个文本列再加上一个全文索引来尽量抵消连接消耗。

- 不拆可能带来的问题：查询性能。

如果能容忍不拆分带来的查询性能损失的话，上面的方案在某个极致条件下肯定会出现问题，那么不拆就是最好的选择。

实际场景下，例如说商品表数据量比较大的情况下，会将商品描述单独存储到一个表中。即，使用拆的方案。

MySQL 有哪些存储引擎?

MySQL 提供了多种的存储引擎：

- InnoDB
- MyISAM
- MRG_MYISAM
- MEMORY
- CSV
- ARCHIVE
- BLACKHOLE
- PERFORMANCE_SCHEMA
- FEDERATED
- ...

具体每种存储引擎的介绍，可以看看 《数据库存储引擎》 。

如何选择合适的存储引擎?

提供几个选择标准，然后按照标准，选择对应的存储引擎即可，也可以根据 常用引擎对比 来选择你使用的存储引擎。使用哪种引擎需要根据需求灵活选择，一个数据库中多个表可以使用不同的引擎以满足各种性能和实际需求。使用合适的存储引擎，将会提高整个数据库的性能。

1. 是否需要支持事务。
2. 对索引和缓存的支持。
3. 是否需要使用热备。
4. 崩溃恢复，能否接受崩溃。

5. 存储的限制。
6. 是否需要外键支持。

芳芳：目前开发已经不考虑外键，主要原因是性能。具体可以看看 《从 MySQL 物理外键开始的思考》 文章。

目前，MySQL 默认的存储引擎是 InnoDB ，并且也是最主流的选择。主要原因如下：

- 【最重要】支持事务。
- 支持行级锁和表级锁，能支持更多的并发量。
- 查询不加锁，完全不影响查询。
- 支持崩溃后恢复。

在 MySQL5.1 以及之前的版本，默认的存储引擎是 MyISAM ，但是目前已经不再更新，且它有几个比较关键的缺点：

- 不支持事务。
- 使用表级锁，如果数据量大，一个插入操作锁定表后，其他请求都将阻塞。

芳芳：也就是说，我们不需要花太多力气在 MyISAM 的学习上。

请说明 InnoDB 和 MyISAM 的区别

	InnoDB	MyISAM
事务	支持	不支持
存储限制	64TB	无
锁粒度	行锁	表锁
崩溃后的恢复	支持	不支持
外键	支持	不支持
全文检索	5.7 版本后支持	支持

更完整的对比，可以看看 《数据库存储引擎》 的 「常用引擎对比」 小节。

请说说 InnoDB 的 4 大特性?

芳芳：貌似我面试没被问过...反正，我是没弄懂过~~

- 插入缓冲(insert buffer)
- 二次写(double write)
- 自适应哈希索引(ahi)
- 预读(read ahead)

*为什么 SELECT COUNT() FROM table 在 InnoDB 比 MyISAM 慢? **

对于 `SELECT COUNT(*) FROM table` 语句，在没有 `WHERE` 条件的情况下，InnoDB 比 MyISAM 可能会慢很多，尤其在大表的情况下。因为，InnoDB 是去实时统计结果，会全表扫描；而 MyISAM 内部维持了一个计数器，预存了结果，所以直接返回即可。

详细的原因，胖友可以看看 《高性能 MySQL 之 Count 统计查询》 博客。

各种不同 MySQL 版本的 Innodb 的改进?

芳芳：这是一个选择了解的问题。

MySQL5.6 下 Innodb 引擎的主要改进：

1. online DDL
2. memcached NoSQL 接口
3. transportable tablespace（ alter table discard/import tablespace）
4. MySQL 正常关闭时，可以 dump 出 buffer pool 的（ space， page_no），重启时 reload，加快预热速度
5. 索引和表的统计信息持久化到 mysql.innodb_table_stats 和 mysql.innodb_index_stats，可提供稳定的执行计划
6. Compressed row format 支持压缩表

MySQL5.7 下 Innodb 引擎的主要改进：

- 1、修改 varchar 字段长度有时可以使用

这里的“有时”，指的是也有些限制。可见 《MySQL 5.7 online ddl 的一些改进》 。

- 2、Buffer pool 支持在线改变大小
- 3、Buffer pool 支持导出部分比例
- 4、支持新建 innodb tablespace，并可以在其中创建多张表
- 5、磁盘临时表采用 innodb 存储，并且存储在 innodb temp tablespace 里面，以前是 MyISAM 存储

- 6、透明表空间压缩功能

【重点】什么是索引?

索引，类似于书籍的目录，想找到一本书的某个特定的主题，需要先找到书的目录，定位对应的页码。

MySQL 中存储引擎使用类似的方式进行查询，先去索引中查找对应的值，然后根据匹配的索引找到对应的数据行。

索引有什么好处?

1. 提高数据的检索速度，降低数据库IO成本：使用索引的意义就是通过缩小表中需要查询的记录的数目从而加快搜索的速度。
2. 降低数据排序的成本，降低CPU消耗：索引之所以查的快，是因为先将数据排好序，若该字段正好需要排序，则正好降低了排序的成本。

索引有什么坏处?

1. 占用存储空间：索引实际上也是一张表，记录了主键与索引字段，一般以索引文件的形式存储在磁盘上。
2. 降低更新表的速度：表的数据发生了变化，对应的索引也需要一起变更，从而减低的更新速度。否则索引指向的物理数据可能不对，这也是索引失效的原因之一。

索引的使用场景?

- 1、对非常小的表，大部分情况下全表扫描效率更高。
- 2、对中大型表，索引非常有效。
- 3、特大型的表，建立和使用索引的代价随着增长，可以使用分区技术来解决。

实际场景下，MySQL 分区表很少使用，原因可以看看 《互联网公司为啥不使用 MySQL 分区表？》 文章。

对于特大型的表，更常用的是“分库分表”，目前解决方案有 Sharding Sphere、MyCAT 等等。

索引的类型?

索引，都是实现在存储引擎层的。主要有六种类型：

- 1、普通索引：最基本的索引，没有任何约束。
- 2、唯一索引：与普通索引类似，但具有唯一性约束。
- 3、主键索引：特殊的唯一索引，不允许有空值。
- 4、复合索引：将多个列组合在一起创建索引，可以覆盖多个列。
- 5、外键索引：只有InnoDB类型的表才可以使用外键索引，保证数据的一致性、完整性和实现级联操作。
- 6、全文索引：MySQL 自带的全文索引只能用于 InnoDB、MyISAM ，并且只能对英文进行全文检索，一般使用全文索引引擎。

常用的全文索引引擎的解决方案有 Elasticsearch、Solr 等等。最为常用的是 Elasticsearch 。

具体的使用，可以看看 《服务端指南 数据存储篇 | MySQL（03） 如何设计索引》 。

MySQL 索引的“创建”原则?

注意，是“创建”噢。

- 1、最适合索引的列是出现在 `WHERE` 子句中的列，或连接子句中的列，而不是出现在 `SELECT` 关键字后的列。
- 2、索引列的基数越大，索引效果越好。

具体为什么，可以看看如下两篇文章：

  - 《MySQL 索引基数》 理解相对简单
  - 《低基数索引为什么会对性能产生负面影响》 写的更原理，所以较为难懂。

- 3、根据情况创建复合索引，复合索引可以提高查询效率。

因为复合索引的基数会更大。

- 4、避免创建过多的索引，索引会额外占用磁盘空间，降低写操作效率。
- 5、主键尽可能选择较短的数据类型，可以有效减少索引的磁盘占用提高查询效率。
- 6、对字符串进行索引，应该定制一个前缀长度，可以节省大量的索引空间。

MySQL 索引的“使用”注意事项?

注意，是“使用”噢。

- 1、应尽量避免在 `WHERE` 子句中使用 `!=` 或 `<>` 操作符，否则将引擎放弃使用索引而进行全表扫描。优化器将无法通过索引来确定将要命中的行数,因此需要搜索该表的所有行。

注意，`column IS NULL` 也是不可以使用索引的。

- 2、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，如：`SELECT id FROM t WHERE num = 10 OR num = 20` 。
- 3、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。
- 4、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。
- 5、不要在 `WHERE` 子句中的 `=` 左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。
- 6、复合索引遵循前缀原则。
- 7、如果 MySQL 评估使用索引比全表扫描更慢，会放弃使用索引。如果此时想要索引，可以在语句中添加强制索引。

- 8、列类型是字符串类型，查询时一定要给值加引号，否则索引失效。
- 9、`LIKE` 查询，`%` 不能在前，因为无法使用索引。如果需要模糊匹配，可以使用全文索引。

关于这块，可以看看 《服务端指南 数据存储篇 | MySQL（04） 索引使用的注意事项》 文章，写的更加细致。

以下三条 SQL 如何建索引，只建一条怎么建?

```
WHERE a = 1 AND b = 1
WHERE b = 1
WHERE b = 1 ORDER BY time DESC
```

- 以顺序 b , a, time 建立复合索引，`CREATE INDEX table1_b_a_time ON index_test01(b, a, time)`。
- 对于第一条 SQL ，因为最新 MySQL 版本会优化 `WHERE` 子句后面的列顺序，以匹配复合索引顺序。

想知道一个查询用到了哪个索引，如何查看?

`EXPLAIN` 显示了 MYSQL 如何使用索引来处理 SELECT 语句以及连接表,可以帮助选择更好的索引和写出更优化的查询语句。

使用方法，在 `SELECT` 语句前加上 `EXPLAIN` 就可以了。感兴趣的胖友，可以详细看看 《MySQL explain 执行计划详细解释》 。

【重点】MySQL 索引的原理?

解释 MySQL 索引的原理，篇幅会比较长，并且网络上已经有靠谱的资料可以看，所以艿艿这里整理了几篇，胖友可以对照着看。

- 《MySQL 索引原理》
- 《深入理解 MySQL 索引原理和实现 —— 为什么索引可以加速查询? 》

下面，艿艿对关键知识做下整理，方便胖友回顾。

MySQL 有哪些索引方法?

艿艿：这个问题是索引方法 Index Method ， 上面的索引类型 Index Type 。

在 MySQL 中，我们可以看到两种索引方式：

- B-Tree 索引。
- Hash 索引。

实际场景下，我们基本仅仅使用 B-Tree 索引。详细的对比可以看看 《MySQL BTree 索引和 hash 索引的区别》 。

对于 Hash 索引，我们了解即可，面试重点是掌握 B-Tree 索引的原理。

什么是 B-Tree 索引?

B-Tree 是为磁盘等外存储设备设计的一种平衡查找树。因此在讲 B-Tree 之前先了解下磁盘的相关知识。

- 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出来，而不是需要什么取什么。
- InnoDB存储引擎中有页（Page）的概念，页是其磁盘管理的最小单位。InnoDB 存储引擎中默认每个页的大小为 16 KB，可通过参数 innodb_page_size 将页的大小设置为 4K、8K、16K ，在 MySQL 中可通过如下命令查看页的大小：

```
mysql> show variables like 'innodb_page_size';
```

- 而系统一个磁盘块的存储空间往往没有这么大，因此 InnoDB 每次申请磁盘空间时都会是若干地址连续磁盘块来达到页的大小 16KB 。InnoDB 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。

B-Tree 结构的数据可以让系统高效的找到数据所在的磁盘块。为了描述B-Tree，首先定义一条记录为一个二元组 [key, data] ，key 为记录的键值，对应表中的主键值，data 为一行记录中除主键外的数据。对于不同的记录，key值互不相同。

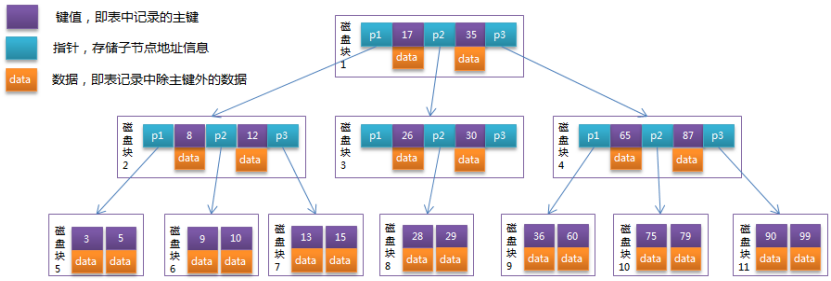
一棵 m 阶的 B-Tree 有如下特性：

艿艿：如果胖友对算法了解不对，可能对下面的各种数字关系不太能理解。最起码，要弄懂层级关系，以及每种节点存储的数据。

1. 每个节点最多有 m 个孩子。
    - 除了根节点和叶子节点外，其它每个节点至少有 Ceil(m/2) 个孩子。
    - 若根节点不是叶子节点，则至少有 2 个孩子。
2. 所有叶子节点都在同一层，且不包含其它关键字信息。
3. 每个非叶子节点包含 n 个关键字信息（P0,P1,...Pn, k1,...kn）
    - 关键字的个数 n 满足：ceil(m/2)-1 <= n <= m-1

- ki(i=1,...n) 为关键字，且关键字升序排序。
- Pi(i=0,...n) 为指向子树根节点的指针。P(i-1) 指向的子树的所有节点关键字均小于 ki，但都大于 k(i-1)。

B-Tree 中的每个节点根据实际情况可以包含大量的关键字信息和分支，如下图所示为一个 3 阶的 B-Tree：



- 每个节点占用一个盘块的磁盘空间，一个节点上有两个升序排序的 key 和三个指向子树根节点的 point，point 存储的是子节点所在磁盘块的地址。两个 key 划分成的三个范围域，对应三个 point 指向的子树的数据的范围域。
- 以根节点为例，key 为 17 和 35，P1 指针指向的子树的数据范围为小于 17，P2 指针指向的子树的数据范围为 [17~35]，P3 指针指向的子树的数据范围为大于 35。

模拟查找 key 为 29 的过程：

- 1、根据根节点找到磁盘块 1，读入内存。【磁盘I/O操作第1次】
- 2、比较 key 29 在区间（17,35），找到磁盘块 1 的指针 P2。
- 3、根据 P2 指针找到磁盘块 3，读入内存。【磁盘I/O操作第2次】
- 4、比较 key 29 在区间（26,30），找到磁盘块3的指针P2。
- 5、根据 P2 指针找到磁盘块 8，读入内存。【磁盘I/O操作第3次】
- 6、在磁盘块 8 中的 key 列表中找到 eky 29。

分析上面过程，发现需要 3 次磁盘 I/O 操作，和 3 次内存查找操作。由于内存中的 key 是一个有序表结构，可以利用二分法查找提高效率。而 3 次磁盘 I/O 操作是影响整个 B-Tree 查找效率的决定因素。B-Tree 相对于 AVLTree 缩减了节点个数，使每次磁盘 I/O 取到内存的数据都发挥了作用，从而提高了查询效率。

什么是 B+Tree 索引?

B+Tree 是在 B-Tree 基础上的一种优化，使其更适合实现外存储索引结构，InnoDB存储引擎就是用 B+Tree 实现其索引结构。

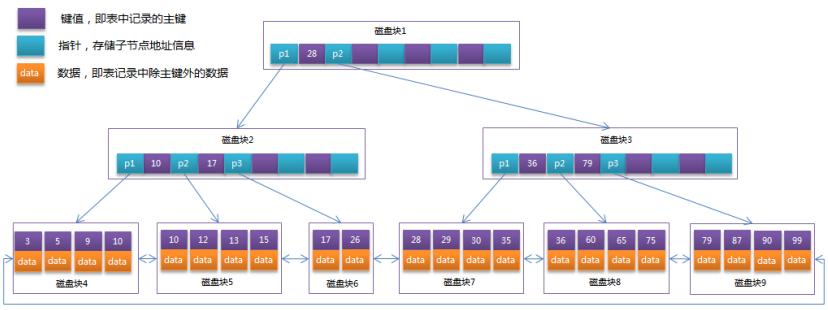
下面这一段，面试非常关键。

从上一节中的 B-Tree 结构图中可以看到，每个节点中不仅包含数据的 key 值，还有 data 值。而每一个页的存储空间是有限的，如果 data 数据较大时将会导致每个节点（即一个页）能存储的 key 的数量很小，当存储的数据量很大时同样会导致 B-Tree 的深度较大，增大查询时的磁盘 I/O 次数，进而影响查询效率。在 B+Tree 中，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储 key 值信息，这样可以大大加大每个节点存储的 key 值数量，降低 B+Tree 的高度。

B+Tree 相对于 B-Tree 有几点不同：

- 非叶子节点只存储键值信息。
- 所有叶子节点之间都有一个链指针。
- 数据记录都存放在叶子节点中。

将上一节中的 B-Tree 优化，由于 B+Tree 的非叶子节点只存储键值信息，假设每个磁盘块能存储 4 个键值及指针信息，则变成 B+Tree 后其结构如下图所示：



- 通常在 B+Tree 上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子节点，而且所有叶子节点（即数据节点）之间是一种链式环结构。因此可以对 B+Tree 进行两种查找运算：一种是对于主键的范围查找和分页查找，另一种是从根节点开始，进行随机查找。

可能上面例子中只有 22 条数据记录，看不出 B+Tree 的优点，下面做一个推算：

- InnoDB 存储引擎中页的大小为 16KB，一般表的主键类型为 INT（占用4个字节） 或 BIGINT（占用8个字节），指针类型也一般为 4 或 8 个字节，也就是说一个页（B+Tree 中的一个节点）中大概存储 16KB/(8B+8B)=1K 个键值（因为是估值，为方便计算，这里的 K 取值为〖10〗^3）。也就是说一个深度为 3 的 B+Tree 索引可以维护10^3 * 10^3 * 10^3 = 10亿 条记录。
- 实际情况中每个节点可能不能填充满，因此在数据库中，B+Tree 的高度一般都在 2~4 层。MySQL 的 InnoDB 存储引擎在设计时是将根节点常驻内存的，也就是说查找某一键值的行记录时最多只需要 1~3 次磁盘 I/O 操作。

B-Tree 有哪些索引类型?

在 B+Tree 中，根据叶子节点的内容，索引类型分为**主键索引**和**非主键索引**。

注意，这里的索引类型，和上面的索引类型，还是对的上的噢。

- 主键索引的叶子节点存的数据是整行数据(即具体数据)。在 InnoDB 里，主键索引也被称为**聚集索引**（clustered index）。
- 非主键索引的叶子节点存的数据是整行数据的主键，键值是索引。在 InnoDB 里，非主键索引也被称为**辅助索引**（secondary index）。

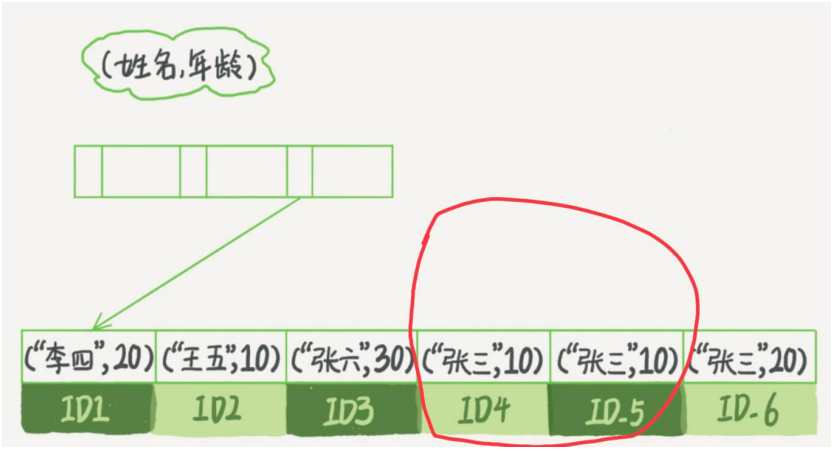
二级索引的叶节点存储的是主键值，而不是行指针，这是为了减少当出现行移动或数据页分裂时二级索引的维护工作，但会让二级索引占用更多的空间。

辅助索引与聚集索引的区别在于辅助索引的叶子节点并不包含行记录的全部数据，而是存储相应行数据的聚集索引键，即主键。当通过辅助索引来查询数据时，需要进过两步：

- 首先，InnoDB 存储引擎会遍历辅助索引找到主键。
- 然后，再通过主键在聚集索引中找到完整的行记录数据。

另外，InnoDB 通过主键聚簇数据，如果没有定义主键，会选择一个唯一的非空索引代替，如果没有这样的索引，会隐式定义个主键作为聚簇索引。

再另外，可能有胖友有和芳芳的一样疑惑，在**辅助索引**如果相同的索引怎么存储？最终存储到 B+Tree 非子节点中时，它们对应的主键 ID 是不同的，所以妥妥的。如下图所示：



聚簇索引的注意点有哪些?

聚簇索引表最大限度地提高了 I/O 密集型应用的性能，但它也有以下几个限制：

- 1、插入速度严重依赖于插入顺序，按照主键的顺序插入是最快的方式，否则将会出现页分裂，严重影响性能。因此，对于 InnoDB 表，我们一般都会定义一个自增的 ID 列为主键。

关于这一点，可能面试官会换一个问法。例如，为什么主键需要是自增 ID ，又或者为什么主键需要带有时间性关联。

- 2、更新主键的代价很高，因为将会导致被更新的行移动。因此，对于InnoDB 表，我们一般定义主键为不可更新。

MySQL 默认情况下，主键是允许更新的。对于 MongoDB ，其 主键是不允许更新的。

- 3、二级索引访问需要两次索引查找，第一次找到主键值，第二次根据主键值找到行数据。

当然，有一种情况可以无需二次查找，基于非主键索引查询，但是查询字段只有主键 ID ，那么在二级索引中就可以查找到。

- 4、主键 ID 建议使用整型。因为，每个主键索引的 B+Tree 节点的键值可以存储更多主键 ID ，每个非主键索引的 B+Tree 节点的数据可以存储更多主键 ID 。

什么是索引的最左匹配特性?

当 B+Tree 的数据项是复合的数据结构，比如索引 `(name, age, sex)` 的时候，B+Tree 是按照从左到右的顺序来建立搜索树

的。

- 比如当 `(张三, 20, F)` 这样的数据来检索的时候，B+Tree 会优先比较 name 来确定下一步的所搜方向，如果 name 相同再依次比较 age 和 sex ，最后得到检索的数据。
- 但当 `(20, F)` 这样的没有 name 的数据来的时候，B+Tree 就不知道下一步该查哪个节点，因为建立搜索树的时候 name 就是第一个比较因子，必须要先根据 name 来搜索才能知道下一步去哪里查询。
- 比如当 `(张三, F)` 这样的数据来检索时，B+Tree 可以用 name 来指定搜索方向，但下一个字段 age 的缺失，所以只能把名字等于张三的数据都找到，然后再匹配性别是 F 的数据了。

这个是非常重要的性质，即索引的最左匹配特性。

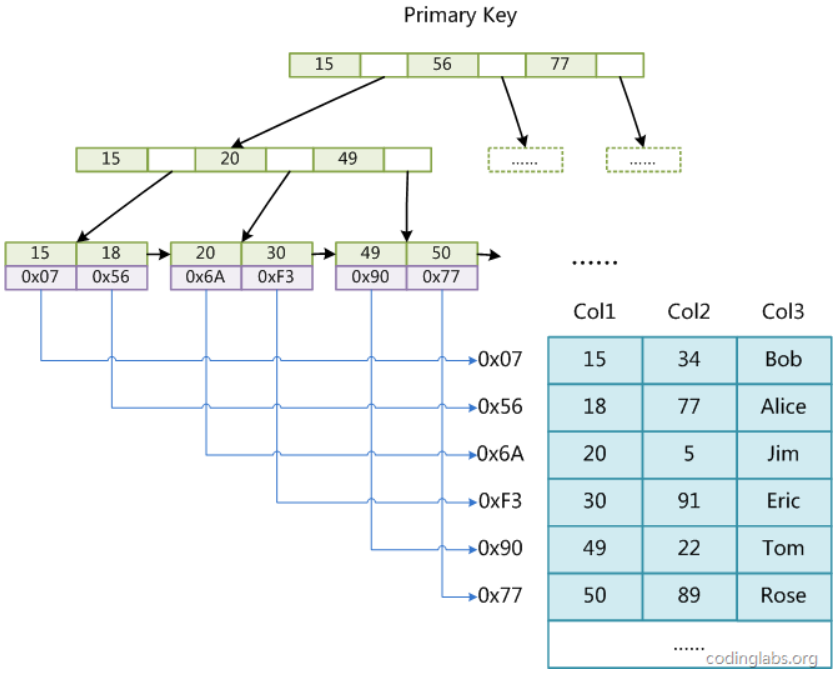
MyISAM 索引实现?

芳芳：注意，我们上面看到的都是 InnoDB 存储引擎下的索引实现。

MyISAM 索引的实现，和 InnoDB 索引的实现是一样使用 B+Tree ，**差别在于 MyISAM 索引文件和数据文件是分离的，索引文件仅保存数据记录的地址**。

1）主键索引：

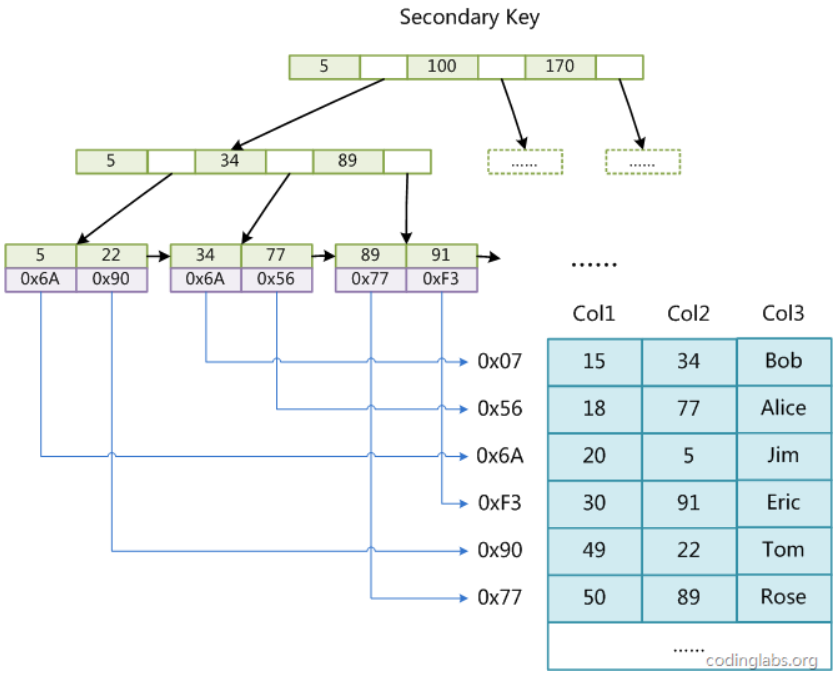
MyISAM引 擎使用B+Tree作为索引结构，**叶节点的data域存放的是数据记录的地址**。下图是MyISAM主键索引的原理图：



- 这里设表一共有三列，假设我们以 Col1 为主键，上图是一个 MyISAM 表的主索引（Primary key）示意。可以看出 MyISAM 的索引文件仅仅保存数据记录的地址。

2）辅助索引：

在 MyISAM 中，主索引和辅助索引在结构上没有任何区别，只是主索引要求 key 是唯一的，而辅助索引的 key 可以重复。如果我们在 Col2 上建立一个辅助索引，则此索引的结构如下图所示：



- 同样也是一颗 B+Tree，data 域保存数据记录的地址。因此，**MyISAM 中索引检索的算法为首先按照 B+Tree 搜索算法搜索索引，如果指定的 Key 存在，则取出其 data 域的值，然后以 data 域的值为地址，读取相应数据记录。**

MyISAM 的索引方式也叫做“**非聚集**”的，之所以这么称呼是为了与InnoDB 的聚集索引区分。

MyISAM 索引与 InnoDB 索引的区别?

- InnoDB 索引是聚簇索引，MyISAM 索引是非聚簇索引。
- InnoDB 的主键索引的叶子节点存储着行数据，因此主键索引非常高效。
- MyISAM 索引的叶子节点存储的是行数据地址，需要再寻址一次才能得到数据。
- InnoDB 非主键索引的叶子节点存储的是主键和其他带索引的列数据，因此查询时做到覆盖索引会非常高效。

覆盖索引，指的是基于非主键索引查询，但是查询字段只有主键 ID，那么在二级索引中就可以查找到。

【重点】请说说 MySQL 的四种事务隔离级别?

芳芳：这是面试中最常见的问题。如果回答不对，可能就被抬走了。

事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。

这样可以防止出现脏数据，防止数据库数据出现问题。

事务的特性指的是?

指的是 **ACID**，如下图所示：

1. 原子性 Atomicity：一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。即，事务不可分割、不可约简。

2. 一致性 Consistency： 在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设约束、触发器、级联回滚等。

3. 隔离性 Isolation： 数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。

4. 持久性 Durability： 事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

🦅 事务的并发问题?

实际场景下，事务并不是串行的，所以会带来如下三个问题：

- 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。

🦅 MySQL 事务隔离级别会产生的并发问题?

事务定义了四种事务隔离级别，不同数据库在实现时，产生的并发问题是不同的。

不同的隔离级别有不同的现象，并有不同的锁定/并发机制，隔离级别越高，数据库的并发性就越差。

- READ UNCOMMITTED（未提交读）： 事务中的修改，即使没有提交，对其他事务也都是可见的。

会导致脏读。

- READ COMMITTED（提交读）： 事务从开始直到提交之前，所做的任何修改对其他事务都是不可见的。

会导致不可重复读。

这个隔离级别，也可以叫做“不可重复读”。

- REPEATABLE READ（可重复读）： 一个事务按相同的查询条件读取以前检索过的数据，其他事务插入了满足其查询条件的新数据。产生幻行。

会导致幻读。

- SERIALIZABLE（可串行化）： 强制事务串行执行。

MySQL InnoDB 采用 **MVCC** 来支持高并发，实现结果如下表所示：

关于 Oracle 和 PostgreSQL ，需要胖友自己去搜索资料。

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
读已提交（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是（x）
串行化（serializable）	否	否	否

- MySQL 默认的事务隔离级别为可重复读（repeatable-read） 。
- 上图的 `<X>` 处，MySQL 因为其间隙锁的特性，导致其在可重复读（repeatable-read）的隔离级别下，不存在幻读问题。也就是说，上图 `<X>` 处，需要改成“否”！！！！
- 😈 记住这个表的方式，我们会发现它是自左上向右下是一个对角线。当然，最好是去理解。
- 具体的实验，胖友可以看看 《MySQL 的四种事务隔离级别》 。
- 有些资料说可重复读解决了幻读，实际是存在的，可以通过 `SELECT xxx FROM t WHERE id = ? FOR UPDATE` 的方式，获得到悲观锁，禁止其它事务操作对应的数据，从而解决幻读问题。感兴趣的胖友，可以看看如下文章：
  - 必读 《MySQL 幻读的详解、实例及解决办法》 案例性更强，易懂。

其实 RR 也是可以避免幻读的，通过对 select 操作手动加 行X锁（SELECT ... FOR UPDATE 这也正是 SERIALIZABLE 隔离级别下会隐式为你做的事情），同时还需要知道，即便当前记录不存在，比如 id = 1 是不存在的，当前事务也会获得一把记录锁（因为InnoDB的行锁锁定的是索引，故记录实体存在与否没关系，存在就加 行X锁，不存在就加 next-key lock间隙X锁），其他事务则无法插入此索引的记录，故杜绝了幻读。

  - 选读 《MySQL 的 InnoDB 的幻读问题》 原理性更强，读懂会很爽。
  - 随意 《Innodb 中 RR 隔离级别能否防止幻读？》 一个简单的讨论。

【重点】请说说 MySQL 的锁机制?

表锁是日常开发中的常见问题，因此也是面试当中最常见的考察点，当多个查询同一时刻进行数据修改时，就会产生并发控

制的问题。MySQL 的共享锁和排他锁，就是读锁和写锁。

- 共享锁：不堵塞，多个用户可以同时读一个资源，互不干扰。
- 排他锁：一个写锁会阻塞其他的读锁和写锁，这样可以只允许一个用户进行写入，防止其他用户读取正在写入的资源。

锁的粒度?

- 表锁：系统开销最小，会锁定整张表，MyIsam 使用表锁。
- 行锁：最大程度的支持并发处理，但是也带来了最大的锁开销，InnoDB 使用行锁。

什么是悲观锁？什么是乐观锁?

1）悲观锁

它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

在悲观锁的情况下，为了保证事务的隔离性，就需要一致性锁定读。读取数据时给加锁，其它事务无法修改这些数据。修改删除数据时也要加锁，其它事务无法读取这些数据。

芳芳：悲观锁，就是我们上面看到的共享锁和排他锁。

2）乐观锁

相对悲观锁而言，乐观锁机制采取了更加宽松的加锁机制。悲观锁大多数情况下依靠数据库的锁机制实现，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销，特别是对长事务而言，这样的开销往往无法承受。

而乐观锁机制在一定程度上解决了这个问题。乐观锁，大多是基于数据版本（ Version ）记录机制实现。何谓数据版本？即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个 “version” 字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。

芳芳：乐观锁，实际就是通过版本号，从而实现 CAS 原子性更新。

什么是死锁?

多数情况下，可以认为如果一个资源被锁定，它总会在以后某个时间被释放。而死锁发生在当多个进程访问同一数据库时，其中每个进程拥有的锁都是其他进程所需的，由此造成每个进程都无法继续下去。简单的说，进程 A 等待进程 B 释放他的资源，B 又等待 A 释放他的资源，这样就互相等待就形成死锁。

虽然进程在运行过程中，可能发生死锁，但死锁的发生也必须具备一定的条件，死锁的发生必须具备以下四个必要条件：

- 互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放。
- 请求和保持条件：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。
- 不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。
- 环路等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合 {P0，P1，P2，•••，Pn} 中的 P0 正在等待一个 P1 占用的资源；P1 正在等待 P2 占用的资源，……，Pn 正在等待已被 P0 占用的资源。

下列方法有助于最大限度地降低死锁：

- 设置获得锁的超时时间。

通过超时，至少保证最差最差最差情况下，可以有退出的口子。

- 按同一顺序访问对象。

这个是最重要的方式。

- 避免事务中的用户交互。
- 保持事务简短并在一个批处理中。
- 使用低隔离级别。
- 使用绑定连接。

MySQL 中 InnoDB 引擎的行锁是通过加在什么上完成(或称实现)的？为什么是这样子的？？

InnoDB 是基于索引来完成行锁。例如：`SELECT * FROM tab_with_index WHERE id = 1 FOR UPDATE` 。

- `FOR UPDATE` 可以根据条件来完成**行锁**锁定，并且 id 是有索引键的列,如果 id 不是索引键那么 InnoDB 将完成**表锁**，并发将无从谈起。

关于熟悉 MySQL 的锁机制?

芳芳：这个问题，芳芳也没特别研究，先 mark 在这里。

- gap 锁

- next-key 锁
- Innodb 的行锁是怎么实现的?

Innodb 的锁的策略为 next-key 锁，即 record lock + gap lock ，是通过在 index 上加 lock 实现的。

  - 如果 index 为 unique index ，则降级为 record lock 行锁。
  - 如果是普通 index ，则为 next-key lock 。
  - 如果没有 index ，则直接锁住全表，即表锁。

- MyISAM 的表锁是怎么实现的?

MyISAM 直接使用表锁。

【重要】MySQL 查询执行顺序?

MySQL 查询执行的顺序是：

```
(1)     SELECT
(2)     DISTINCT <select_list>
(3)     FROM <left_table>
(4)     <join_type> JOIN <right_table>
(5)     ON <join_condition>
(6)     WHERE <where_condition>
(7)     GROUP BY <group_by_list>
(8)     HAVING <having_condition>
(9)     ORDER BY <order_by_condition>
(10)    LIMIT <limit_number>
```

具体的，可以看看 《SQL 查询之执行顺序解析》 文章。

【重要】聊聊 MySQL SQL 优化?

可以看看如下几篇文章：

- 《PHP 面试之 MySQL 查询优化》
- 《【面试】【MySQL常见问题总结】【03】》 第 078、095、105 题

另外，除了从 SQL 层面进行优化，也可以从服务器硬件层面，进一步优化 MySQL 。具体可以看看 《MySQL 数据库性能优化之硬件优化》 。

【加分】什么是 MVCC ?

芳芳：这是一个面试的加分题，一些大厂比较喜欢问，例如蚂蚁金服。

多版本并发控制（MVCC），是一种用来**解决读-写冲突**的无锁并发控制，也就是为事务分配单向增长的时间戳，为每个修改保存一个版本，版本与事务时间戳关联，读操作只读该事务开始前的数据库的快照。 这样在读操作不用阻塞写操作，写操作不用阻塞读操作的同时，避免了脏读和不可重复读。

推荐可以看看如下资料：

- 沈询 《在线分布式数据库原理与实践》

一共 1 小时 53 分钟，有趣，牛逼，强烈推荐！！！

- 钟延辉
  - 《分布式数据库 MVCC 技术探秘 (1)》
  - 《分布式数据库 MVCC 技术探秘(2): 混合逻辑时钟》

编写 SQL 查询语句的考题合集

因为考题比较多，芳芳就不一一列举，瞄了一些还不错的文章，如下：

- 《10 道 MySQL 查询语句面试题》
- 《MySQL 开发面试题》
- 《企业面试题丨最常问的 MySQL 面试题集合（二）》

MySQL 数据库 CPU 飙升到 500% 的话，怎么处理?

当 CPU 飙升到 500% 时，先用操作系统命令 top 命令观察是不是 mysqld 占用导致的，如果不是，找出占用高的进程，并进行相关处理。

如果此时是 IO 压力比较大，可以使用 iostat 命令，定位是哪个进程占用了磁盘 IO 。

如果是 mysqld 造成的，使用 `show processlist` 命令，看看里面跑的 Session 情况，是不是有消耗资源的 SQL 在运行。找出消耗高的 SQL ，看看执行计划是否准确， index 是否缺失，或者实在是数据量太大造成。一般来说，肯定要 kill 掉这些线程(同时观察 CPU 使用率是否下降)，等进行相应的调整(比如说加索引、改 SQL 、改内存参数)之后，再重新跑这些 SQL。

也可以查看 MySQL 慢查询日志，看是否有慢 SQL 。

也有可能是每个 SQL 消耗资源并不多，但是突然之间，有大量的 Session 连进来导致 CPU 飙升，这种情况就需要跟应用一起来分析为何连接数会激增，再做出相应的调整，比如说限制连接数等。

在 MySQL 服务器运行缓慢的情况下输入什么命令能缓解服务器压力?

这个回答，和上面的回答思路是差不多的，优秀在更有层次感。

1）检查系统的状态

通过操作系统的一些工具检查系统的状态，比如 CPU、内存、交换、磁盘的利用率，根据经验或与系统正常时的状态相比对，有时系统表面上看起来看空闲，这也可能不是一个正常的状态，因为 CPU 可能正等待IO的完成。除此之外，还应观注那些占用系统资源(CPU、内存)的进程。

- 使用 sar 来检查操作系统是否存在 IO 问题。
- 使用 vmstat 监控内存 CPU 资源。
- 磁盘 IO 问题，处理方式：做 raid10 提高性能 。
- 网络问题，telnet 一下 MySQL 对外开放的端口。如果不通的话，看看防火墙是否正确设置了。另外，看看 MySQ L是不是开启了 skip-networking 的选项，如果开启请关闭。

2）检查 MySQL 参数

- max_connect_errors
- connect_timeout
- skip-name-resolve
- slave-net-timeout=seconds
- master-connect-retry

3）检查 MySQL 相关状态值

- 关注连接数
- 关注下系统锁情况
- 关注慢查询（slow query）日志

运维

理解一遍，即使有蛮多不会，也不要担心太多。

Innodb 的事务与日志的实现方式

有多少种日志?

- redo 日志
- undo 日志

日志的存放形式?

- redo：在页修改的时候，先写到 redo log buffer 里面， 然后写到 redo log 的文件系统缓存里面(fwrite)，然后再同步到磁盘文件（fsync）。
- undo：在 MySQL5.5 之前，undo 只能存放在 ibdata* 文件里面， 5.6 之后，可以通过设置 innodb_undo_tablespaces 参数把 undo log 存放在 ibdata* 之外。

事务是如何通过日志来实现的，说得越深入越好

芳芳：这个流程的理解还是比较简单的，实际思考实现感觉还是蛮复杂的。

基本流程如下：

- 因为事务在修改页时，要先记 undo ，在记 undo 之前要记 undo 的 redo， 然后修改数据页，再记数据页修改的 redo。 redo（里面包括 undo 的修改）一定要比数据页先持久化到磁盘。
- 当事务需要回滚时，因为有 undo，可以把数据页回滚到前镜像的状态。
- 崩溃恢复时，如果 redo log 中事务没有对应的 commit 记录，那么需要用 undo 把该事务的修改回滚到事务开始之前。如果有 commit 记录，就用 redo 前滚到该事务完成时并提交掉。

MySQL binlog 的几种日志录入格式以及区别

各种日志格式的涵义

binlog 有三种格式类型，分别如下：

1）Statement

每一条会修改数据的 SQL 都会记录在 binlog 中。

- 优点：不需要记录每一行的变化，减少了 binlog 日志量，节约了 IO，提高性能。(相比 row 能节约多少性能与日志量，这个取决于应用的 SQL 情况，正常同一条记录修改或者插入 row 格式所产生的日志量还小于 Statement 产生的日志量，但是考虑到如果带条件的 update 操作，以及整表删除，alter 表等操作，ROW 格式会产生大量日志，因此在考虑是否使用 ROW 格式日志时应该跟据应用的实际情况，其所产生的日志量会增加多少，以及带来的 IO 性能问题。)
- 缺点：由于记录的只是执行语句，为了这些语句能在 slave 上正确运行，因此还必须记录每条语句在执行的时候的一些相关信息，以保证所有语句能在 slave 得到和在 master 端执行时候相同 的结果。另外 MySQL 的复制，像一些特定函数功能，slave 可与 master 上

要保持一致会有很多相关问题(如 `sleep()` 函数，`last_insert_id()`，以及 user-defined functions(udf) 会出现问题)。

- 使用以下函数的语句也无法被复制：
  - `LOAD_FILE()`
  - `UUID()`
  - `USER()`
  - `FOUND_ROWS()`
  - `SYSDATE()` (除非启动时启用了 `--sysdate-is-now` 选项)

同时在 INSERT ...SELECT 会产生比 RBR 更多的行级锁 。

2）Row

不记录 SQL 语句上下文相关信息，仅保存哪条记录被修改。

- 优点：binlog 中可以不记录执行的 SQL 语句的上下文相关的信息，仅需要记录那一条记录被修改成什么了。所以 rowlevel 的日志内容会非常清楚的记录下每一行数据修改的细节。而且不会出现某些特定情况下的存储过程，或 function ，以及 trigger 的调用和触发无法被正确复制的问题。
- 缺点：所有的执行的语句当记录到日志中的时候，都将以每行记录的修改来记录，这样可能会产生大量的日志内容,比如一条 Update 语句，修改多条记录，则 binlog 中每一条修改都会有记录，这样造成 binlog 日志量会很大，特别是当执行 alter table 之类的语句的时候，由于表结构修改，每条记录都发生改变，那么该表每一条记录都会记录到日志中。

3）Mixedlevel

是以上两种 level 的混合使用。

- 一般的语句修改使用 Statement 格式保存 binlog 。
- 如一些函数，statement 无法完成主从复制的操作，则采用 Row 格式保存 binlog 。

MySQL 会根据执行的每一条具体的 SQL 语句来区分对待记录的日志形式，也就是在 Statement 和 Row 之间选择 一种。

新版本的 MySQL 中对 row level 模式也被做了优化，并不是所有的修改都会以 row level 来记录。

- 像遇到表结构变更的时候就会以 Statement 模式来记录。
- 至于 Update 或者 Delete 等修改数据的语句，还是会记录所有行的变更，即使用 Row 模式。

🦅 适用场景?

在一条 SQL 操作了多行数据时， Statement 更节省空间，Row 更占用空间。但是， Row 模式更可靠。

因为，互联网公司，使用 MySQL 的功能相对少，基本不使用存储过程、触发器、函数的功能，选择默认的语句模式，Statement Level（默认）即可。

🦅 结合第一个问题，每一种日志格式在复制中的优劣?

- Statement 可能占用空间会相对小一些，传送到 slave 的时间可能也短，但是没有 Row 模式的可靠。
- Row 模式在操作多行数据时更占用空间，但是可靠。

所以，这是在占用空间和可靠之间的选择。

如何在线正确清理 MySQL binlog?

MySQL 中的 binlog 日志记录了数据中的数据变动，便于对数据的基于时间点和基于位置的恢复。但日志文件的大小会越来越大，占用大量的磁盘空间，因此需要定时清理一部分日志信息。

```
# 首先查看主从库正在使用的binlog文件名称
show master(slave) status

# 删除之前一定要备份
purge master logs before'2017-09-01 00:00:00'; # 删除指定时间前的日志
purge master logs to'mysql-bin.000001'; # 删除指定的日志文件

# 自动删除：通过设置binlog的过期时间让系统自动删除日志
show variables like 'expire_logs_days'; # 查看过期时间
set global expire_logs_days = 30; # 设置过期时间
```

MySQL 主从复制的流程是怎么样的?

MySQL 的主从复制是基于如下 3 个线程的交互（多线程复制里面应该是 4 类线程）：

- 1、Master 上面的 binlog dump 线程，该线程负责将 master 的 binlog event 传到 slave 。
- 2、Slave 上面的 IO 线程，该线程负责接收 Master 传过来的 binlog，并写入 relay log 。
- 3、Slave 上面的 SQL 线程，该线程负责读取 relay log 并执行。
- 4、如果是多线程复制，无论是 5.6 库级别的假多线程还是 MariaDB 或者 5.7 的真正的多线程复制， SQL 线程只做 coordinator ，只负责把 relay log 中的 binlog 读出来然后交给 worker 线程， woker 线程负责具体 binlog event 的执行。

🦅 MySQL 如何保证复制过程中数据一致性?

芳芳：这个问题比较难，理解不了也没问题。我自己也没完全理解，主要是网络找到这个答案，后续有精力在研究。

- 1、在 MySQL5.5 以及之前， slave 的 SQL 线程执行的 relay log 的位置只能保存在文件（ relay-log.info）里面，并且该文件默认每执行 10000 次事务做一次同步到磁盘， 这意味着 slave 意外 crash 重启时， SQL 线程执行到的位置和数据库的数据是不一致的，将导致复制报错，如果不重搭复制，则有可能会导致数据不一致。
    - MySQL 5.6 引入参数 relay_log_info_repository，将该参数设置为 TABLE 时， MySQL 将 SQL 线程执行到的位置存到 mysql.slave_relay_log_info 表，这样更新该表的位置和 SQL 线程执行的用户事务绑定成一个事务，这样 slave 意外宕机后，slave 通过 innodb 的崩溃恢复可以把 SQL 线程执行到的位置和用户事务恢复到一致性的状态。
- 2、MySQL 5.6 引入 GTID 复制，每个 GTID 对应的事务在每个实例上面最多执行一次， 这极大地提高了复制的数据一致性。
- 3、MySQL 5.5 引入半同步复制， 用户安装半同步复制插件并且开启参数后，设置超时时间，可保证在超时时间内如果 binlog 不传到 slave 上面，那么用户提交事务时不会返回，直到超时后切成异步复制，但是如果切成异步之前用户线程提交时在 master 上面等待的时候，事务已经提交，该事务对 master 上面的其他 session 是可见的，如果这时 master 宕机，那么到 slave 上面该事务又不可见了，该问题直到 5.7 才解决。
- 4、MySQL 5.7 引入无损半同步复制，引入参 rpl_semi_sync_master_wait_point，该参数默认为 after_sync，指的是在切成半同步之前，事务不提交，而是接收到 slave 的 ACK 确认之后才提交该事务，从此，复制真正可以做到无损的了。
- 5、可以再说一下 5.7 的无损复制情况下， master 意外宕机，重启后发现有 binlog 没传到 slave 上面，这部分 binlog 怎么办？？？分 2 种情况讨论， 1 宕机时已经切成异步了， 2 是宕机时还没切成异步？？？ 这个怎么判断宕机时有没有切成异步呢？？？ 分别怎么处理？？？

MySQL 如何解决主从复制的延时性?

5.5 是单线程复制，5.6 是多库复制（对于单库或者单表的并发操作是没用的），5.7 是真正意义的多线程复制，它的原理是基于 group commit， 只要 master 上面的事务是 group commit 的，那 slave 上面也可以通过多个 worker线程去并发执行。 和 MairaDB10.0.0.5 引入多线程复制的原理基本一样。

工作遇到的复制 bug 的解决方法?

5.6 的多库复制有时候自己会停止，我们写了一个脚本重新 start slave 。

你是否做过主从一致性校验，如果有，怎么做的，如果没有，你打算怎么做?

主从一致性校验有多种工具 例如 checksum、mysqldiff、pt-table-checksum 等。

聊聊 MySQL 备份方式？备份策略是怎么样的?

具体的，胖友可以看看 《MySQL 高级备份策略》 。主要有几个知识点：

- 数据的备份类型
    - 【常用】完全备份

这是大多数人常用的方式，它可以备份整个数据库，包含用户表、系统表、索引、视图和存储过程等所有数据库对象。但它需要花费更多的时间和空间，所以，一般推荐一周做一次完全备份。

    - 增量备份

它是只备份数据库一部分的另一种方法，它不使用事务日志，相反，它使用整个数据库的一种新映象。它比最初的完全备份小，因为它只包含自上次完全备份以来所改变的数据库。它的优点是存储和恢复速度快。推荐每天做一次差异备份。

    - 【常用】事务日志备份

事务日志是一个单独的文件，它记录数据库的改变，备份的时候只需要复制自上次备份以来对数据库所做的改变，所以只需要很少的时间。为了使数据库具有鲁棒性，推荐每小时甚至更频繁的备份事务日志。

    - 文件备份

数据库可以由硬盘上的许多文件构成。如果这个数据库非常大，并且一个晚上也不能将它备份完，那么可以使用文件备份每晚备份数据库的一部分。由于一般情况下数据库不会大到必须使用多个文件存储，所以这种备份不是很常用。

- 备份数据的类型
    - 热备份
    - 温备份
    - 冷备份
- 备份工具
    - cp
    - mysqldump
    - xtrabackup
    - lvm2 快照

MySQL 几种备份方式?

MySQL 一般有 3 种备份方式。

1）逻辑备份

使用 MySQL 自带的 mysqldump 工具进行备份。备份成sql文件形式。

- 优点：最大好处是能够与正在运行的 MySQL 自动协同工作，在运行期间可以确保备份是当时的点，它会自动将对应操作的表锁定，不允许其他用户修改(只能访问)。可能会阻止修改操作。SQL 文件通用方便移植。
- 缺点：备份的速度比较慢。如果是数据量很多的时候，就很耗时间。如果数据库服务器处在提供给用户服务状态，在这段长时间操作过程中，意味着要锁定表(一般是读锁定，只能读不能写入数据)，那么服务就会影响的。

2）物理备份

芳芳：因为现在主流是 InnoDB ，所以基本不再考虑这种方式。

直接拷贝只适用于 MyISAM 类型的表。这种类型的表是与机器独立的。但实际情况是，你设计数据库的时候不可能全部使用 MyISAM 类型表。你也不可能因为 MyISAM 类型表与机器独立，方便移植，于是就选择这种表，这并不是选择它的理由。

- 缺点：你不能去操作正在运行的 MySQL 服务器(在拷贝的过程中有用户通过应用程序访问更新数据，这样就无法备份当时的数据)，可能无法移植到其他机器上去。

3）双机热备份。

当数据量太大的时候备份是一个很大的问题，MySQL 数据库提供了一种主从备份的机制，也就是双机热备。

- 优点：适合数据量大的时候。现在明白了，大的互联网公司对于 MySQL 数据备份，都是采用热机备份。搭建多台数据库服务器，进行主从复制。

数据库不能停机，请问如何备份？如何进行全备份和增量备份？

可以使用逻辑备份和双机热备份。

- 完全备份：完整备份一般一段时间进行一次，且在网站访问量最小的时候，这样常借助批处理文件定时备份。主要是写一个批处理文件在里面写上处理程序的绝对路径然后把要处理的东西写在后面，即完全备份数据库。
- 增量备份：对 ddl 和 dml 语句进行二进制备份。且 5.0 无法增量备份，5.1 后可以。如果要实现增量备份需要在 `my.ini` 文件中配置备份路径即可，重启 MySQL 服务器，增量备份就启动了。

你的备份工具的选择？备份计划是怎么样的？

视库的大小来定，一般来说 100G 内的库，可以考虑使用 mysqldump 来做，因为 mysqldump 更加轻巧灵活，备份时间选在业务低峰期，可以每天进行都进行全量备份(mysqldump 备份出来的文件比较小，压缩之后更小)。

100G 以上的库，可以考虑用 xtrabackup 来做，备份速度明显要比 mysqldump 要快。一般是选择一周一个全备，其余每天进行增量备份，备份时间为业务低峰期。

芳芳：一般情况下，选择每周备份 + 每天增量备份比较靠谱。

备份恢复时间是多长？

物理备份恢复快，逻辑备份恢复慢。

这里跟机器，尤其是硬盘的速率有关系，以下列举几个仅供参考：

- 20G 的 2 分钟（mysqldump）
- 80G 的 30分钟（mysqldump）
- 111G 的 30分钟（mysqldump）
- 288G 的 3 小时（xtrabackup）
- 3T 的 4 小时（xtrabackup）

逻辑导入时间一般是备份时间的 5 倍以上。

备份恢复失败如何处理？

首先在恢复之前就应该做足准备工作，避免恢复的时候出错。比如说备份之后的有效性检查、权限检查、空间检查等。如果万一报错，再根据报错的提示来进行相应的调整。

mysqldump 和 xtrabackup 实现原理？

1）mysqldump

mysqldump 是最简单的逻辑备份方式。

- 在备份 MyISAM 表的时候，如果要得到一致的数据，就需要锁表，简单而粗暴。
- 在备份 InnoDB 表的时候，加上 `-master-data=1 -single-transaction` 选项，在事务开始时刻，记录下 binlog pos 点，然后利用 MVCC 来获取一致的数据，由于是一个长事务，在写入和更新量很大的数据库上，将产生非常多的 undo ，显著影响性能，所以要慎用。
- 优点：简单，可针对单表备份，在全量导出表结构的时候尤其有用。
- 缺点：简单粗暴，单线程，备份慢而且恢复慢，跨 IDC 有可能遇到时区问题

2）xtrabackup

xtrabackup 实际上是物理备份+逻辑备份的组合。

- 在备份 InnoDB 表的时候，它拷贝 ibd 文件，并一刻不停的监视 redo log 的变化，append 到自己的事务日志文件。在拷贝 ibd 文件过程中，ibd文件本身可能被写”花”，这都不是问题，因为在拷贝完成后的第一个 prepare 阶段，xtrabackup 采用类似于 Innodb 崩溃恢复的方法，把数据文件恢复到与日志文件一致的状态，并把未提交的事务回滚。
- 如果同时需要备份 MyISAM 表以及 InnoDB 表结构等文件，那么就需要用 `flush tables with lock` 来获得全局锁，开始拷贝这些不再变化的文件，同时获得 binlog 位置，拷贝结束后释放锁，也停止对 redo log 的监视。

如何从 mysqldump 产生的全库备份中只恢复某一个库、某一张表?

具体可见 《MySQL 全库备份中恢复某个库和某张表以及 mysqldump 参数 --ignore-table 介绍》 文章。

聊聊 MySQL 集群?

芳芳：这块芳芳懂的少，主要找了一些网络上的资料。

- 《五大常见的 MySQL 高可用方案》
- 《高性能、高可用、可扩展的 MySQL 集群如何组建？》

对于简历中写有熟悉 MySQL 高可用方案?

我一般先问他现在管理的数据库架构是什么，如果他只说出了主从，而没有说任何 HA 的方案，那么我就可以判断出他没有实际的 HA 经验。

不过这时候也不能就是断定他不懂 MySQL 高可用，也许是没有实际机会去使用，那么我就要问 MMM 以及 MHA 以及 MM + keepalived 等的原理、实现方式以及它们之间的优势和不足了，一般这种情况下，能说出这个的基本没有。

- MMM 那东西好像不靠谱，据说不稳定，但是有人在用的，和 mysql-router 比较像，都是指定可写的机器和只读机器。
- MHA 的话一句话说不完，可以搜索下相关博客。

使用过其他分支版本的数据库吗？Percona、Mariadb 等。对Percona 的 pxc 集群了解吗?

除了 Oracle 旗下的 MySQL 外，我还使用过 Percona Server 。

Percona 是在原生 MySQL 的基础上，进行了优化和改进，所以 Percona 的性能比 MySQL 更好。

- 目前，我知道 Percona 提供免费的线程池功能，而社区版的 MySQL 没有线程池的功能（当然，企业版的mysql是有线程池的，但是需要收费）
- 另外 Percona 还支持 NUMA 等功能。

我熟悉 pxc ，我曾经在测试环境搭建过 pxc ，但是没有在生产上使用，因为目前使用 pxc 的企业不是很多，目前我知道搜狐在用 pxc 。

- pxc 是摒弃 MySQL 主从的概念，即对于 pxc 来说，每个节点都可以读写，并且写一份数据，其他节点会同时拥有，这是一种同步的复制方案（区别于 MySQL 主从的异步复制）。

聊聊 MySQL 安全?

感兴趣的胖友，可以看看：

- 《保障 MySQL 安全的14个最佳方法》
- 《详解 MySQL 安全配置》

MySQL 有哪些日志?

- 错误日志：记录了当 mysqld 启动和停止时，以及服务器在运行过程中发生任何严重错误时的相关信息。
- 二进制文件：记录了所有的 DDL（数据定义语言）语句和 DML（数据操纵语言）语句，不包括数据查询语句。语句以“事件”的形式保存，它描述了数据的更改过程。（定期删除日志，默认关闭）。

就是我们上面看到的 MySQL binlog 日志。

- 查询日志：记录了客户端的所有语句，格式为纯文本格式，可以直接进行读取。（log 日志中记录了所有数据库的操作，对于访问频繁的系统，此日志对系统性能的影响较大，建议关闭，默认关闭）。
- 慢查询日志：慢查询日志记录了包含所有执行时间超过参数long_query_time（单位：秒）所设置值的 SQL 语句的日志。（纯文本格式）

重要，一定要开启。

另外，错误日志和慢查询日志的详细解释，可以看看 《MySQL 日志文件之错误日志和慢查询日志详解》 文章。

聊聊 MySQL 监控?

你是如何监控你们的数据库的?

监控的工具有很多，例如 Zabbix ， Lepus ， 我这里用的是 Lepus 。

对一个大表做在线 DDL ，怎么进行实施的才能尽可能降低影响?

使用 pt-online-schema-change ，具体可以看看 《MySQL 大表在线 DML 神器--pt-online-schema-change》 文章。

另外，还有一些其它的工具，胖友可以搜索下。

彩蛋

T T 知识点真多，胖友还是重点掌握文初说的几个，不要方，我们能赢。如下是芳芳在网络上找到过的一个 MySQL 问题清

单（研发向）：

- 3-1 数据库架构
- 3-2 优化你的索引-运用二叉查找树
- 3-3 优化你的索引-运用B树
- 3-4 优化你的索引-运用B+树
- 3-5 优化你的索引-运用Hash以及BitMap
- 3-6 密集索引和稀疏索引的区别
- 3-7 索引额外的问题之如何调优Sql
- 3-8 索引额外问题之最左匹配原则的成因
- 3-9 索引额外问题之索引是建立越多越好吗
- 3-10 锁模块之MyISAM与InooDB关于锁方面的区别
- 3-11 锁模块之MyISAM与InooDB关于锁方面的区别_2
- 3-12 锁模块之数据库事务的四大特性
- 3-13 锁模块之事务并发访问产生的问题以及事务隔离机制
- 3-14 锁模块之事务并发访问产生的问题以及事务隔离机制_2
- 3-15 锁模块之当前读和快照读
- 3-16 锁模块之RR如何避免幻读
- 3-17 锁模块小结
- 3-18 关键语法讲解
- 3-19 本章总结
- 3-20 彩蛋之面试的三层架构

参考与推荐如下文章：

- ranjun940726 《PHP 面试指南》
- 紫葡萄0 《MySQL 索引的使用和优化》
- Ddaidai 《【MySQL】20 个经典面试题》
- 瘦瘦鸭 《MySQL 面试知识点总结》
- 立超的专栏 《MyISAM 和 InnoDB 的索引实现》
- 时芥蓝 《MySQL 面试之必会知识点》
- derrantcm 《【面试】【MySQL常见问题总结】【04】》
- mrlapulga 《MySQL 经典面试题》 提供的面试题，难的想哭。
- 小麦苗 《MySQL 笔试面试题集合》 全的想哭。