面试必备(背)--MySQL 八股文系列！

mp.weixin.qq.com/s/CfGEUdGMlgMDorkekF44Rw

1. 三大范式

- 第一范式：确保每列保持原子性，数据表中的所有字段值都是不可分解的原子值。
- 第二范式：确保表中的每列都和主键相关。
- 第三范式：确保每列都和主键列直接相关而不是间接相关。

1.1 反范式化

我们应从业务角度出发，设计出符合范式准则要求的表结构。

- 反范式化指的是通过增加冗余或重复的数据来换时间增加效率,违反第二第三范式。
- 反范式化可以减少关联查询时，join表的次数。
- 在一些场景下，可以通过 JSON 数据类型进行反范式设计，提升存储效率。

2. mysql的几种引擎，有什么区别

特性	InnoDB	MyISAM	MEMORY
事物安全	支持	不支持	不支持
对外建的支持	支持	不支持	不支持
存储限制	64TB	有	有
空间使用	高	低	低
内存使用	高	低	高
插入数据的速度	低	高	高

2.1 InnoDB、MyISAM 对比

- InnoDB支持事务，MyISAM不支持。
- InnoDB 支持外键，而 MyISAM 不支持。
- InnoDB是聚集索引，数据文件是和索引绑在一起的，必须要有主键。MyISAM是非聚集索引，索引和数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。
- InnoDB 不保存表的具体行数。MyISAM 用一个变量保存了整个表的行数。
- Innodb 有 redolog 日志文件，MyISAM 没有。
- Innodb存储文件有frm、ibd，而Myisam是frm、MYD、MYI。
- Innodb：frm是表定义文件，ibd是数据文件。
- Myisam：frm是表定义文件，myd是数据文件，myi是索引文件。
- InnoDB 支持表、行锁，而 MyISAM 支持表级锁。
- InnoDB 必须有唯一索引(主键),如果没有指定的话 InnoDB 会自己生成一个隐藏列 Row_id 来充当默认主键，MyISAM 可以没有。

3. 为什么要使用自增主键

- １.普通索引的 B+ 树上存放的是主键索引的值，如果该值较大，会「导致普通索引的存储空间较大」
- ２.使用自增 id 做主键索引新插入数据只要放在该页的最尾端就可以，直接「按照顺序插入」，不用刻意维护
- 3.页分裂容易维护，当插入数据的当前页快满时，会发生页分裂的现象，如果主键索引不为自增 id，那么数据就可能从页的中间插入，页的数据会频繁的变动，「导致页分裂维护成本较高」

4. 什么是索引？

> “
> 百度百科的解释：索引是对数据库表的一列或者多列的值进行排序一种结构，使用索引可以快速访问数据表中的特定信息。
>
> ”

索引就一本书的目录，可以极大的提高我们在数据库的查询效率。

4.1 索引的优缺点？

优点：

- 大大加快数据检索的速度。
- 将随机I/O变成顺序I/O(因为B+树的叶子节点是连接在一起的)
- 加速表与表之间的连接

缺点：

- 从空间角度考虑，建立索引需要占用物理空间
- 从时间角度 考虑，创建和维护索引都需要花费时间，例如对数据进行增删改的时候都需要维护索引。

4.2 索引的数据结构？

索引的数据结构主要有 B+ 树和哈希表，对应的索引分别为 B+ 树索引和哈希索引。InnoDB 默认的索引类型为 B+ 树索引。

4.3 索引的类型有哪些?

MySQL 主要的索引类型主要有 FULLTEXT，HASH，BTREE，RTREE。

- FULLTEXT

  FULLTEXT 即全文索引，MyISAM存储引擎和InnoDB存储引擎在MySQL5.6.4以上版本支持全文索引，一般用于查找文本中的关键字，多在CHAR，VARCHAR，TAXT 等数据类型上创建全文索引。全文索引主要是用来解决 `WHERE name LIKE "%wekenw%"` 等针对文本的模糊查询效率低的问题。
- HASH

  HASH 即哈希索引，哈希索引多用于等值查询，时间复杂夫为o(1)，效率非常高，但不支持排序、范围查询及模糊查询等。
- BTREE

  BTREE 即 B+ 树索引，INnoDB存储引擎默认的索引，支持排序、分组、范围查询、模糊查询等，并且性能稳定。
- RTREE

- RTREE 即空间数据索引，多用于地理数据的存储，相比于其他索引，空间数据索引的优势在于范围查找。

4.4 索引的种类有哪些？

- 主键索引：数据列不允许重复，不能为NULL，一个表只能有一个主键索引
- 组合索引：由多个列值组成的索引。
- 唯一索引：数据列不允许重复，可以为NULL，索引列的值必须唯一的，如果是组合索引，则列值的组合必须唯一。
- 全文索引：对文本的内容进行搜索。
- 普通索引：基本的索引类型，可以为NULL

4.5 什么是聚簇索引，什么是非聚簇索引？

- 聚簇索引：将数据和索引放到一起存储，索引结构的叶子节点保留了数据行。
- 非聚簇索引：将数据进和索引分开存储，索引叶子节点存储的是指向数据行的地址。

4.6 索引的设计原则？

- 最适合创建索引的列是出现在 WHERE 或 ON 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。
- 对于字符串进行索引，应该制定一个前缀长度，可以节省大量的索引空间。
- 索引列的基数越大、索引列的区分度越高，索引的效果越好。
- 尽量使用短索引，因为较小的索引涉及到的磁盘I/O较少，并且索引高速缓存中的块可以容纳更多的键值，会使得查询速度更快。
- 尽量利用最左前缀。
- 不要过度索引，每个索引都需要额外的物理空间，维护也需要花费时间，所以索引不是越多越好。

4.7 索引失效的场景有哪些?

- 不要在索引上做任何操作（计算、函数、自动/手动类型转换），不然会导致索引失效而转向全表扫描。
- 不能继续使用索引中范围条件（bettween、<、>、in等）右边的列 。

- 索引字段上使用（！= 或者 < >）判断时，会导致索引失效而转向全表扫描。
- 索引字段上使用 is null / is not null 判断时，会导致索引失效而转向全表扫描。
- 索引字段使用like以通配符开头（‘%字符串’）时，会导致索引失效而转向全表扫描，也是最左前缀原则。
- 索引字段是字符串，但查询时不加单引号，会导致索引失效而转向全表扫描。
- 索引字段使用 or 时，会导致索引失效而转向全表扫描。

4.8 创建索引的语法：

首先创建一个表：`create table t1 (id int primary key,username varchar(20),password varchar(20));`

创建单个索引的语法：`CREATE INDEX 索引名 on 表名（字段名）`

索引名一般是：`表名_字段名`

给id创建索引：`CREATE INDEX t1_id on t1(id);`

创建联合索引的语法：`CREATE INDEX 索引名 on 表名（字段名1，字段名2）`

给 username 和 password 创建联合索引：`CREATE index t1_username_password ON t1(username,password)`

其中index还可以替换成 unique，primary key，分别代表唯一索引和主键索引

删除索引：`DROP INDEX t1_username_password ON t1`

5.数据库的事务

5.1 什么是事务?其特性是什么?

事务是指是程序中一系列操作必须全部成功完成，有一个失败则全部失败

特性：

- 1.「原子性（Atomicity）」：要么全部执行成功，要么全部不执行。
- 2.「一致性（Consistency）」：事务前后数据的完整性必须保持一致。
- 3.「隔离性（Isolation）」：隔离性是当多个事务同事触发时，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。
- 4.「持久性（Durability）」：事务完成之后的改变是永久的。

5.2 事务的隔离级别?

- 1.「读已提交」：即能够「读取到那些已经提交」的数据。
- 2.「读未提交」：即能够「读取到没有被提交」的数据。
- 3.「可重复读」：可重复读指的是在一个事务内，最开始读到的数据和事务结束前的「任意时刻读到的同一批数据都是一致的」。
- 4.「可串行化」：最高事务隔离级别，不管多少事务，都是「依次按序一个一个执行」。

5.3 隔离性实现原理：

隔离性的实现原理比较特殊，是通过数据库锁的机制实现的。

隔离性分四个级别：

- 读未提交：一个事务可以读到另外一个事务未提交的数据。脏读

  实现：事务在读数据的时候并未对数据进行加锁。

  事务在发生更新数据的瞬间，必须先对其加 行级共享锁，直到事务结束才释放。

  举例：事务A读取某行记录时(没有加锁)，事务2也能对这行记录进行读取、更新。当事务B对该记录进行更新时，事务A读取该记录，能读到事务B对该记录的修改版本，即使该修改尚未被提交。

  事务A更新某行记录时，事务B不能对这行记录做更新，直到事务A结束。

- 读已提交：一个事务可以读到另外一个事务提交的数据。不可重复读

  实现：事务对当前被读取的数据加 `行级共享锁` （当读到时才加锁），一旦读完该行，立即释放该行级共享锁；

  事务在更新某数据的瞬间（就是发生更新的瞬间），必须先对其加 `行级排他锁` ，直到事务结束才释放。

  原理：事务A读取某行记录时，事务B也能对这行记录进行读取、更新；当事务B对该记录进行更新时，事务A再次读取该记录，读到的只能是事务B对其更新前的版本，或者事务B提交后的版本。事务A更新某行记录时，事务B不能对这行记录做更新，直到事务1结束。

  流程描述：事务A读操作会加上 `共享锁` ，事务B写操作时会加上 `排他锁` ，当事务B正在写操作时，事务A要读操作，发现有排他锁，事务A就会阻塞，等待排他锁释放(事务B写操作提交才会释放)，才能进行读操作。

- 可重复读

  实现：事务在读取某数据的瞬间（就是开始读取的瞬间），必须先对其加 `行级共享锁` ，直到事务结束才释放；

  事务在更新某数据的瞬间（就是发生更新的瞬间），必须先对其加 `行级排他锁` ，直到事务结束才释放。

  举例：事务A读取某行记录时，事务B也能对这行记录进行读取、更新；当事务B对该记录进行更新时，事务A再次读取该记录，读到的仍然是第一次读取的那个版本。事务A更新某行记录时，事务B不能对这行记录做更新，直到事务1结束。

- 可串行化(Serializable) 写操作串联执行

  实现：事务在读取数据时，必须先对其加 `表级共享锁` ，直到事务结束才释放；

  事务在更新数据时，必须先对其加 `表级排他锁` ，直到事务结束才释放。

  举例：事务A正在读取A表中的记录时，则事务B也能读取A表，但不能对A表做更新、新增、删除，直到事务A结束。事务A正在更新A表中的记录时，则事务B不能读取A表的任意记录，更不可能对A表做更新、新增、删除，直到事务A结束。

  原理：在读操作时，加 `表级共享锁` ，事务结束时释放；写操作时候，加 `表级独占锁` ，事务结束时释放。

「MySQL的默认隔离级别是可重复读。」数据库的隔离级别分别可以解决数据库的脏读、不可重复读、幻读等问题。

隔离级别	脏读	不可重复读	幻读
未提交读	允许	允许	允许
提交读	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

- 1.「脏读」

  脏读指的是「读到了其他事务未提交的数据」，未提交意味着这些数据可能会回滚，也就是可能最终不会存到数据库中，也就是不存在的数据。读到了并一定最终存在的数据，这就是脏读。

- 2.「不可重复读」

  对比可重复读，不可重复读指的是在同一事务内，「不同的时刻读到的同一批数据可能是不一样的」。

- 3.「幻读」

  幻读是针对数据插入（INSERT）操作来说的。假设事务A对某些行的内容作了更改，但是还未提交，此时事务B插入了与事务A更改前的记录相同的记录行，并且在事务A提交之前先提交了，而这时，在事务A中查询，会发现「好像刚刚的更改对于某些数据未起作用」，但其实是事务B刚插入进来的这就叫幻读。

5.4 隔离级别是如何实现的？

事务的隔离机制主要是依靠锁机制和MVCC(多版本并发控制)实现的，提交读和可重复读可以通过MVCC实现，串行化可以通过锁机制实现。

6. 什么是MVCC，有什么作用?

MVCC:多版本并发控制，主要用来提高数据库的并发性能。

MVCC的作用就是在不加锁的情况下，解决数据库读写冲突问题，并且解决脏读、幻读、不可重复读等问题，但是不能解决丢失修改问题。

7. 数据库的锁

7.1 什么是数据库的锁？

当数据库有并发事务的时候，保证数据访问顺序的机制称为锁机制。

数据库的锁与隔离级别的关系？

隔离级别	实现方式
未提交读	总是读取最新的数据，无需加锁
提交读	读取数据时加共享锁，读取数据后释放共享锁
可重复读	读取数据时加共享锁，事务结束后释放共享锁
串行化	锁定整个范围的键，一直持有锁直到事务结束

7.2 数据库锁的类型有哪些？

MySQL锁类别	资源开销	加锁速度	是否会出现死锁	锁的粒度	并发度
表级锁	小	快	不会	大	低
行级锁	大	慢	会	小	高
页面锁	一般	一般	不会	一般	一般

MyISAM 默认采用表级锁，InnoDB 默认采用行级锁。

从锁的类别上区别可以分为共享锁和排他锁

- 共享锁：共享锁又称读锁，简写为S锁，一个事务对一个数据对象加了S锁，可以对这个数据对象进行读取操作，但不能进行更新操作。并且在加锁期间其他事务只能对这个数据对象加S锁，不能加X锁。
- 排他锁：排他锁又称为写锁，简写为X锁，一个事务对一个数据对象加了X锁，可以对这个对象进行读取和更新操作，加锁期间，其他事务不能对该数据对象进行加X锁或S锁。

7.3 什么是数据库的乐观锁和悲观锁，如何实现？

乐观锁：系统假设数据的更新在大多数时候是不会产生冲突的，所以数据库只在更新操作提交的时候对数据检测冲突，如果存在冲突，则数据更新失败。

乐观锁实现方式：一般通过版本号和CAS算法实现。

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。通俗讲就是每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁。

悲观锁的实现方式：通过数据库的锁机制实现，对查询语句添加for updata。

7.4 什么是死锁？如何避免？

死锁是指两个或者两个以上进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象。在 MySQL 中，MyISAM 是一次获得所需的全部锁，要么全部满足，要么等待，所以不会出现死锁。在 InnoDB 存储引擎中，除了单个 SQL 组成的事务外，锁都是逐步获得的，所以存在死锁问题。

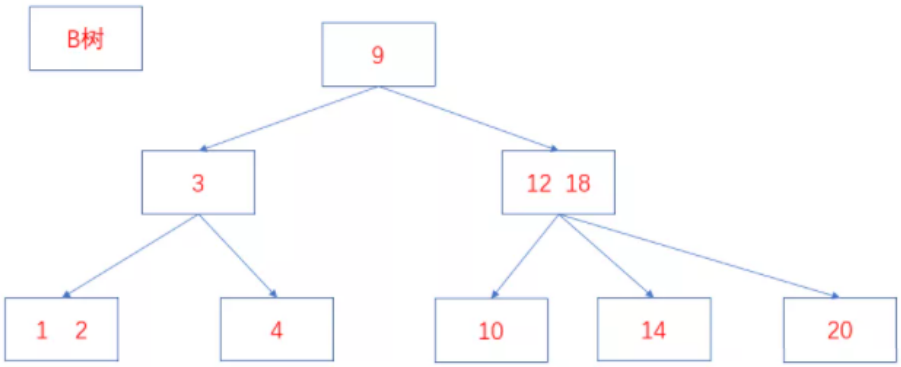
如何避免MySQL发生死锁或锁冲突：

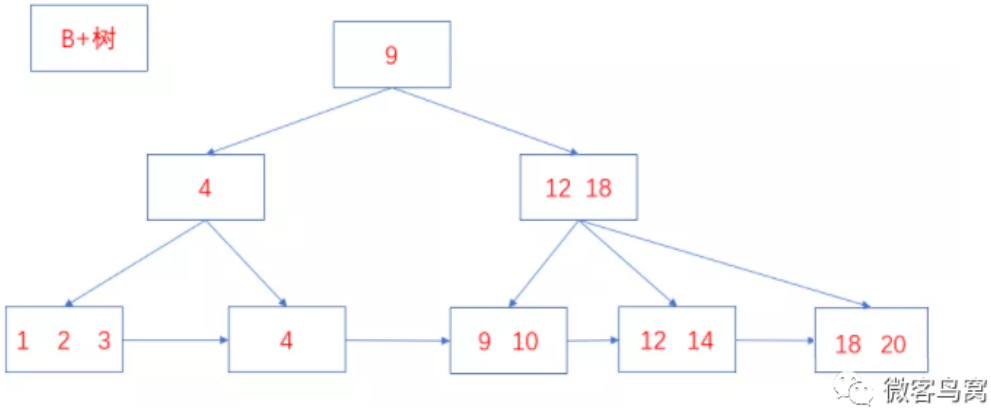
1. 如果不同的程序并发存取多个表，尽量以相同的顺序访问表。
2. 在程序以批量方式处理数据的时候，如果已经对数据排序，尽量保证每个线程按照固定的顺序来处理记录。
3. 在事务中，如果需要更新记录，应直接申请足够级别的排他锁，而不应该先申请共享锁，更新时在申请排他锁，因为在当前用户申请排他锁时，其他事务可能已经获得了相同记录的共享锁，从而造成锁冲突或者死锁。
4. 尽量使用较低的隔离级别。

5. 尽量使用索引访问数据，使加锁更加准确，从而减少锁冲突的机会。
6. 合理选择事务的大小，小事务发生锁冲突的概率更低。
7. 尽量用相等的条件访问数据，可以避免Next-Key锁对并发插入的影响。
8. 不要申请超过实际需要的锁级别，查询时尽量不要显示加锁。
9. 对于一些特定的事务，可以表锁来提高处理速度或减少死锁的概率。

8. B 树和 B+ 树的区别？

- B 树中的内部节点和叶子节点均存放键和值，而 B+ 树的内部节点只有键没有值，叶子节点存放所有的键和值。
- B + 树的叶子节点是通过相连在一起的，方便顺序检索。





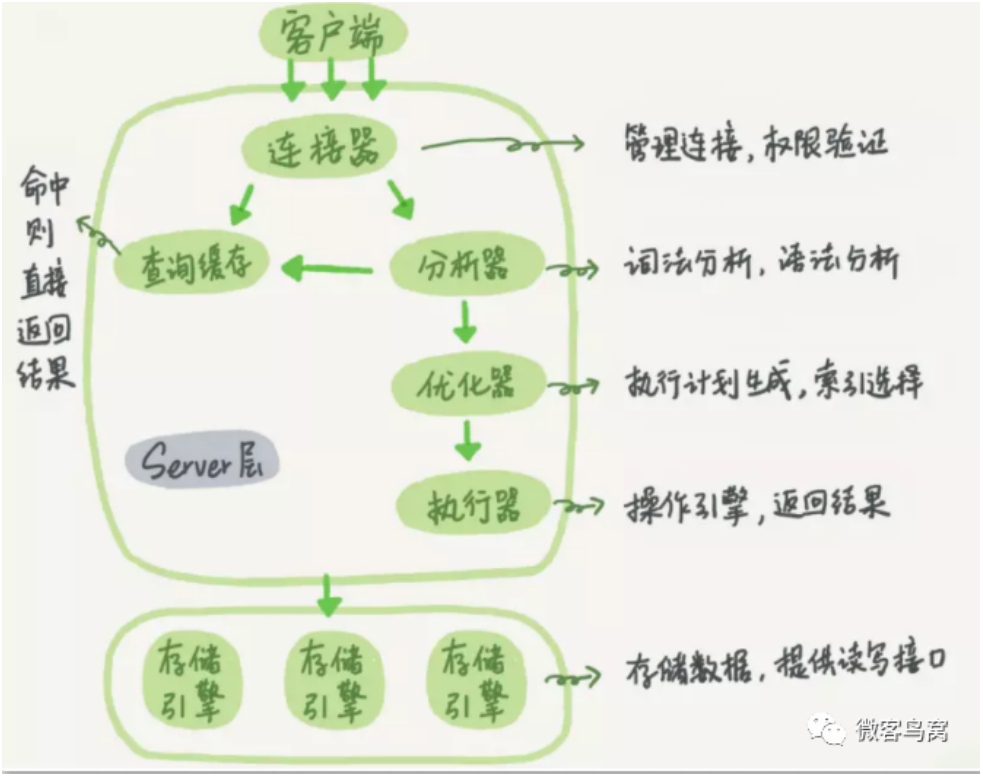
9. 数据库为什么使用 B+ 树而不是 B 树？

- B 树适用于随机检索，而 B+ 树适用于随机检索和顺序检索。
- B+ 树的空间利用率更高，因为 B 树每个节点要存储键和值，而 B+ 树的内部节点只存储键，这样 B+ 树的一个节点就可以存储更多的索引，从而使树的高度变低，减少了 I/O 次数，使得数据检索速度更快。
- B+ 树的叶子节点都是连接在一起的，所以范围查找，顺序查找更加方便。
- B+ 树的性能更加稳定，因为在 B+ 树中，每次查询都是从根节点到叶子节点，而在 B 树中，要查询的值可能不在叶子节点，在内部节点就已经找到。

9.1 什么情况适合使用 B 树呢？

因为 B 树的内部节点也可以存储值，所以可以把一些频繁访问的值放在距离根节点比较近的地方，这样就可以提高查询效率。

10. MySQL执行SQL语句的的流程？



- 1.通过连接器跟客户端「建立连接」。
- 2.通过查询「缓存查询」之前是否有查询过该 sql。
  - 有则直接返回结果
  - 没有则执行第3步
- 3.通过分析器「分析该 sql 的语义」是否正确，包括格式，表等等。
- 4.通过优化器「优化该语句」，比如选择索引，join 表的连接顺序。
- 5.「验证权限」，验证是否有该表的查询权限。
  - 没有则返回无权限的错误
  - 有则执行第6步
- 6.通过执行器调用存储引擎执行该 sql，然后返回「执行结果」。

11. binlog、undolog、relaylog、redolog？

1. binlog 是归档日志，属于 Server 层的日志，是一个二进制格式的文件，用于「记录用户对数据库更新的SQL语句信息」。

主要作用：主从复制、数据恢复。

2. undolog 是 InnoDB 存储引擎的日志，用于保证数据的原子性，「保存了事务发生之前的数据的一个版本，也就是说记录的是数据是修改之前的数据，可以用于回滚」，同时可以提供多版本并发控制下的读（MVCC）。

主要作用：事务回滚、实现多版本控制(MVCC)。

3. relaylog 是中继日志，「在主从同步的时候使用到」，它是一个中介临时的日志文件，用于存储从master节点同步过来的binlog日志内容。

4. redolog 是 「InnoDB 存储引擎所特有的一种日志」，用于记录事务操作的变化，记录的是数据修改之后的值，不管事务是否提交都会记录下来。

可以做「数据恢复并且提供 crash-safe 能力」。当有增删改相关的操作时，会先记录到 Innodb 中，并修改缓存页中的数据，「等到 mysql 闲下来的时候才会真正的将 redolog 中的数据写入到磁盘当中」。

12. 说说两阶段提交。

两阶段提交分为 prepare 和 commit 阶段：

1. 准备阶段：事物 SQL 先写入 redo log buffer，然后做一个事物准备标记，在将log buffer 中的数据刷新到 redo log。

   提交阶段：将事物产生的 binlog 写入文件，刷新磁盘。

2. 再在 redo log 中做一个事物提交的标记，并把 binlog 写成功的标记也一并写入 redo log 文件。

场景分析两阶段提交如何保证数据库的一致性。

1. 准备阶段，redo log 刷新到磁盘了，但是 binlog 写磁盘前发生了 mysql实例 crash，这时会发生怎样的操作呢？

   即使 redo log 写盘成功了，但由于 binlog 未写入成功，需要执行回滚操作来保证数据库的一致性。

2. 提交阶段，binlog 写盘成功了，这时 mysql 实例 crash了。这时 binlog 已经确保写成功了，我们在重启实例进行恢复的时候，只需要让 redo log 重做一次就可以了。

13. 分库分表相关

13.1 分库分表方案:

水平分库：以字段为依据，按照一定策略（hash、range等），将一个库中的数据拆分到多个库中。水平分表：以字段为依据，按照一定策略（hash、range等），将一个表中的数据拆分到多个表中。垂直分库：以表为依据，按照业务归属不同，将不同的表拆分到不同的库中。垂直分表：以字段为依据，按照字段的活跃性，将表中字段拆到不同的表（主表和扩展表）中。

13.2 常用的分库分表中间件：

- sharding-jdbc
- Mycat

13.3 分库分表可能遇到的问题

- 事务问题：需要用分布式事务。
- 跨节点Join的问题：解决这一问题可以分两次查询实现。
- 跨节点的count,order by,group by以及聚合函数问题：分别在各个节点上得到结果后在应用程序端进行合并。
- 数据迁移，容量规划，扩容等问题。
- ID问题：数据库被切分后，不能再依赖数据库自身的主键生成机制啦，最简单可以考虑UUID。
- 跨分片的排序分页问题。

13.4 数据库如何进行垂直拆分以及水平拆分的原理是什么？

垂直拆分

> 专库专用 一个数据库由很多表的构成，每个表对应着不同的业务，垂直切分是指按照业务将表进行分类，分布到不同的数据库上面，这样也就将数据或者说压力分担到不同的库上面

优点：

1. 拆分后业务清晰，拆分规则明确。
2. 系统之间整合或扩展容易。
3. 数据维护简单。

缺点：

1. 部分业务表无法join，只能通过接口方式解决，提高了系统复杂度。
2. 受每种业务不同的限制存在单库性能瓶颈，不易数据扩展跟性能提高。
3. 事务处理复杂。

水平拆分

垂直拆分后遇到单机瓶颈，可以使用水平拆分。相对于垂直拆分的区别是：垂直拆分是把不同的表拆到不同的数据库中，而水平拆分是把同一个表拆到不同的数据库中。

相对于垂直拆分，水平拆分不是将表的数据做分类，而是按照某个字段的某种规则来分散到多个库之中，每个表中包含一部分数据。简单来说，我们可以将数据的水平切分理解为是按照数据行的切分，就是将表中 的某些行切分到一个数据库，而另外的某些行又切分到其他的数据库中，主要有分表，分库两种模式，

优点：

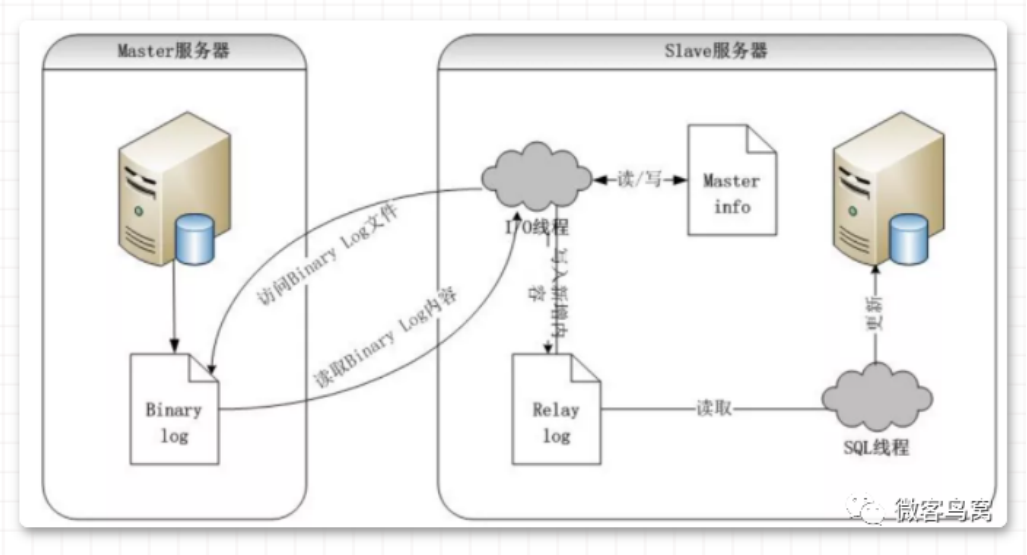
1. 不存在单库大数据，高并发的性能瓶颈。
2. 对应用透明，应用端改造较少。
3. 按照合理拆分规则拆分，join操作基本避免跨库。
4. 提高了系统的稳定性跟负载能力。

缺点：

1. 拆分规则难以抽象。
2. 分片事务一致性难以解决。
3. 数据多次扩展难度跟维护量极大。
4. 跨库join性能较差。

14. Mysql 主从之间是怎么同步数据的?

1.master 主库将此次更新的事件类型「写入到主库的 binlog 文件」中。2.master 「创建 log dump 线程通知 slave」 需要更新数据。3.「slave」 向 master 节点发送请求，「将该 binlog 文件内容存到本地的 relaylog 中」。4.「slave 开启 sql 线程」读取 relaylog 中的内容，「将其中的内容在本地重新执行一遍」，完成主从数据同步。

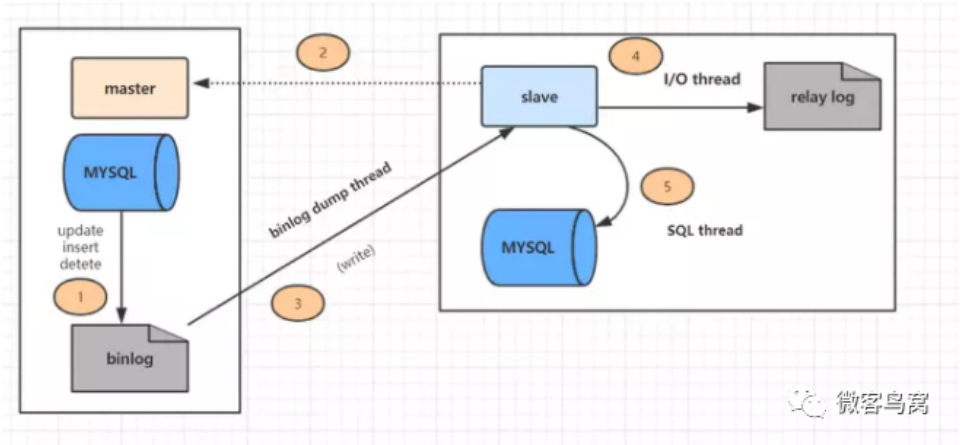


14.1 同步策略

1.「全同步复制」：主库强制同步日志到从库，等全部从库执行完才返回客户端，性能差。2.「半同步复制」：主库收到至少一个从库确认就认为操作成功，从库写入日志成功返回ack确认。

14.2 主从延迟要怎么解决?

主从复制分了五个步骤进行：



1. 主库的更新事件(update、insert、delete)被写到binlog。
2. 从库发起连接，连接到主库。
3. 此时主库创建一个binlog dump thread，把binlog的内容发送到从库。
4. 从库启动之后，创建一个I/O线程，读取主库传过来的binlog内容并写入到relay log。
5. 还会创建一个SQL线程，从relay log里面读取内容，从Exec_Master_Log_Pos位置开始执行读取到的更新事件，将更新内容写入到slave的db。

主从同步延迟的原因:

一个服务器开放N个链接给客户端来连接的，这样有会有大并发的更新操作, 但是从服务器的里面读取binlog的线程仅有一个，当某个SQL在从服务器上执行的时间稍长 或者由于某个SQL要进行锁表就会导致，主服务器的SQL大量积压，未被同步到从服务器里。这就导致了主从不一致， 也就是主从延迟。

主从同步延迟的解决办法

1.MySQL 5.6 版本以后，提供了一种「并行复制」的方式，通过将 SQL 线程转换为多个 work 线程来进行重放。2.「提高机器配置」增加从服务器，目的分散读的压力，从而降低服务器负载。3.在业务初期就选择合适的分库、分表策略，「避免单表单库过大」带来额外的复制压力 4.「避免长事务」。5.「避免让数据库进行各种大量运算」。6.对于一些对延迟很敏感的业务「直接使用主库读」。

15. 如何优化 SQL，说说你的 Sql 调优思路吧

- 「表结构优化」
  - 拆分字段
  - 字段类型的选择
  - 字段类型大小的限制
  - 合理的增加冗余字段
  - 新建字段一定要有默认值
- 「索引方面」
  - 索引字段的选择
  - 利用好mysql支持的索引下推，覆盖索引等功能
  - 唯一索引和普通索引的选择
- 「查询语句方面」
  - 避免索引失效
  - 合理的书写where条件字段顺序
  - 小表驱动大表
  - 可以使用force index()防止优化器选错索引
- 「分库分表」

16. 了解慢日志查询吗？统计过慢查询吗？对慢查询如何优化？

慢查询一般用于记录执行时间超过某个临界值的SQL语句的日志。

相关参数：

slow_query_log：是否开启慢日志查询，1表示开启，0表示关闭。slow_query_log_file：MySQL数据库慢查询日志存储路径。long_query_time：慢查询阈值，当SQL语句查询时间大于阈值，会被记录在日志上。log_queries_not_using_indexes：未使用索引的查询会被记录到慢查询日志中。log_output：日志存储方式。“FILE”表示将日志存入文件。“TABLE”表示将日志存入数据库。如何对慢查询进行优化？

分析语句的执行计划，查看SQL语句的索引是否命中 优化数据库的结构，将字段很多的表分解成多个表，或者考虑建立中间表。优化LIMIT分页。

17. 字段为什么要设置成 not null?

首先说一点，NULL和空值是不一样的，空值是不占用空间的，而NULL是占用空间的，所以字段设为NOT NULL后仍然可以插入空值。

字段设置成not null主要有以下几点原因：

NULL值会影响一些函数的统计，如count，遇到NULL值，这条记录不会统计在内。

B树不存储NULL，所以索引用不到NULL，会造成第一点中说的统计不到的问题。

NOT IN子查询在有NULL值的情况下返回的结果都是空值。

18. varchar和char的区别？

- varchar表示变长，char表示长度固定。
- 存储容量不同，对于 char 来说，最多能存放的字符个数为255。对于 varchar，最多能存放的字符个数是 65532。
- 存储速度不同，char 长度固定，存储速度会比 varchar 快一些，但在空间上会占用额外的空间，属于一种空间换时间的策略。而 varchar 空间利用率会高些，但存储速度慢，属于一种时间换空间的策略。

18.1 为什么 VarChar 建议不要超过255?

- 当定义varchar长度小于等于255时，长度标识位需要一个字节(utf-8编码)。
- 当大于255时，长度标识位需要两个字节，并且建立的索引也会失效。

18.2 varchar(10)和int(10)代表什么含义?

- varchar 的10代表了申请的空间长度，也是可以存储的数据的最大长度。
- int 的10只是代表了展示的长度，不足10位以0填充。
- int(1)和int(10)所能存储的数字大小以及占用的空间都是相同的，只是在展示时按照长度展示。

19. drop、delete和truncate的区别？

	drop	delete	truncate
速度	快	逐行删除，慢	较快
类型	DDL	DML	DDL
回滚	不可回滚	可回滚	不可回滚
删除内容	删除整个表，数据行、索引都会被删除	表结构还在，删除表的一部分或全部数据	表结构还在，删除表的全部数据

总结：删除整个表，使用drop，删除表的部分数据使用delete，保留表结构删除表的全部数据使用truncate。

20 对慢查询如何优化？

慢查询一般用于记录执行时间超过某个临界值的SQL语句的日志。

20.1 如何查找查询速度慢的原因？

1. 记录慢查询日志，分析查询日志，可以使用pt-query-digest工具进行分析。

相关参数：

- slow_query_log：是否开启慢日志查询，1表示开启，0表示关闭。
- slow_query_log_file：MySQL数据库慢查询日志存储路径。
- long_query_time：慢查询阈值，当SQL语句查询时间大于阈值，会被记录在日志上。
- log_queries_not_using_indexes：未使用索引的查询会被记录到慢查询日志中。
- log_output：日志存储方式。“FILE”表示将日志存入文件。“TABLE”表示将日志存入数据库。

2. show profile

```
set profiling=1; //开启，服务器上所有执行语句会记录执行时间，存到临时表中
show profiles
show profile for query 临时表ID
```

3. show status

show status 会返回一些计数器，show global status 会查看所有服务器级别的所有计数。有时根据这些计数，可以推测出哪些操作代价较高或者消耗时间多。

4. show processlist

观察是否有大量线程处于不正常的状态或特征：

```
mysql> show processlist;
+-----+------+-----------+------+---------+------+-------+------------------+
| Id  | User | Host      | db   | Command | Time | State | Info             |
+-----+------+-----------+------+---------+------+-------+------------------+
| 430 | root | localhost | test | Query   |    0 | NULL  | show processlist |
+-----+------+-----------+------+---------+------+-------+------------------+
1 row in set (0.00 sec)
```


5. 使用 explain 分析语句

分析慢语句是否命中索引：

```
mysql> explain select * from a;
+----+-------------+-------+------+---------------+------+---------+------+------+-------
+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra
|
+----+-------------+-------+------+---------------+------+---------+------+------+-------
+
|  1 | SIMPLE      | a     | ALL  | NULL          | NULL | NULL    | NULL |    3 |
|
+----+-------------+-------+------+---------------+------+---------+------+------+-------
+
1 row in set (0.00 sec)
```


20.2 如何对慢查询进行优化？

- 分析语句的执行计划，查看SQL语句的索引是否命中。
- 优化数据库的结构，将字段很多的表分解成多个表，或者考虑建立中间表。
- 优化LIMIT分页。

往期好文推荐：

又谈mysql，面试官问表结构设计要注意啥?

深入浅出，一文吃透mysql索引

面试必备(背)--Go语言八股文系列！

图片及部分相关技术知识点来源于网络搜索，侵权删！

参考资料：

https://leetcode-cn.com/circle/discuss/aX6VxT/

https://blog.csdn.net/weixin_40524659/article/details/104412329

https://mp.weixin.qq.com/s/7q0thZ7Dh99WyKyTUKlJmA

https://mp.weixin.qq.com/s/REzOiTTNKstR1JHlu3thzQ

https://mp.weixin.qq.com/s?__biz=MzAwMjg1NjY3Nw==&mid=2247503280&idx=2&sn=98275dadc94de06fb0a1fbb29d3e2ccf&scene=21#wechat_redirect

https://mp.weixin.qq.com/s?__biz=MzAwMjg1NjY3Nw==&mid=2247500407&idx=2&sn=db4a6c9bfa7859e5904209e4ad1f1353&scene=21#wechat_redirect

NEW////ARRIVAL

微信公众号

gophpython

我的微信

wucs_dd