```
辅助索引是基于表中某个或某些列构建的,与物理顺序无关且不会影响
                                                                                                                                                                                                                          聚集索引和辅助索引区别
                                                                                                                                                                                                                                                       一个表只能有一个聚集索引
                                                                     - 经常用于WHERE查询条件的字段,如果查询条件不止一个,可以建立联合索引
                                                          - 什么时候需要 -
                                                                                                                                                                                                                                                       一个表可以有多个辅助索引 ——— 辅助索引的存在并不影响数据在聚集索引中的组织
                                                                      经常用于GROUP BY,和ORDER BY的字段查询的时候就不需要再去做一次排序了
                                                                                                                                                                                                                                                      聚集索引 ——— 对于按索引顺序的扫描或范围查询,通常提供最快的访问速度,因为数据在磁盘上是连续存储的
                                             索引的使用范围
                                                                       WHERE条件、GROUP BY、ORDER BY里用不到的字段,因为索引会占用空间
                                                                                                                                                                                                                                                                                                 例如,有一个辅助索引基于"分数",而想根据分数查找学生的"姓名",首先需要在辅助索引
                                                                                                                                                                                                                                                                需要两次查找: 查询需要检索不在辅助索引中的列数据
                             什么是索引?
                                                                                                                                                                                                                                                                                                 中找到对应分数的条目,然后通过指针去聚集索引中找到完整的行,最后从中读取"姓名"数据
                                                                       字段中存在大量重复数据
                                                          什么时候不需要
                                                                                                                                                                                                                                                               - 如果查询只需要访问辅助索引中已包含的列,那么只需要一次查找 ———— 例如,若有一个辅助索引包含"分数"和"姓名"列,而查询是要根据分数查找姓名
                                                                                                                                                                                                                                                               - 索引只用于过滤或排序 ——— 例如只用于加速过滤 (WHERE子句) 或排序 (ORDER BY子句) ,并不需要检索具体的行数据
                                                                                                                                                                                                                                                              一 B+ 树索引适合于范围查找和排序操作 ——— 哈希索引适合于等值查询,但不适合范围查询和排序操作
                                                                    使用索引可以大大加快 数据的检索速度 (大大减少检索的数据量)
                                                                                                                                                                                                                                                               - B+ 树在插入和删除数据时需要调整索引结构,索引的维护成本比较高 ———— 哈希索引在插入和删除数据时只需要计算哈希值并插入或删除链表中的记录,因此哈希索引的维护成本相对较低
                                                                    通过创建唯一性索引,可以保证数据库表中每一行数据的唯一性。
                                                                                                                                                                                                                                          +树索引和Hash索引有什么区别
                                                                                                                                                                                                                                                               - B+ 树索引在磁盘上是有序存储的,效率高 ———— 希索引在磁盘上是无序存储的,查询效率低
                                                                    创建索引和维护索引需要耗费许多时间。
                                             为什么要使用索引?
                                                                                                                                                                                                                                                              索引需要使用物理文件存储,也会耗费一定空间。
                                                                                                                                                                                                                                                                       在B+树中,所有的数据都存储在叶子节点,而且这些叶子节点是相互链接的。这意味着无论查找哪个数据,查询的深度都是一样的,使
                                                                                                                                                                                                                                                          查询效率更稳定
                                                             索引查询快于全表查询,但是如果数据库不大,索引提升也不大。
                                                                                                                                                                                                                                                                       得每次查询的效率都很稳定。
                                                                                                                                                                                                                                                                      B+树的非叶子节点不存储数据,只存储键值,这让它可以有更多的分叉(fanout),从而使树的高度更低。低树高意味着查询时需
                                                                                                                                                                                                                                         为什么用B+树不用B树
                                                                                                                         非聚集索引(比如辅助索引)的叶子节点包含索引键和指向数据页的指针,数据页包含实际的数据行
                                                                                                                                                                                                                                                                      要的磁盘读取次数更少,这对于提高数据库的性能非常重要。
                                                                                      B+树只会先找到被查找数据 (即便给定键值) 所在的页
                                                                                                                                                                                                                                                                       由于B树的非叶子节点也存储数据,这限制了每个节点能存储的键值数量,降低了节点的分叉度(fanout)。而B+树通过仅在叶子节
                                                                                                                         聚集索引的叶子节点直接存储整个数据行
                                                                                                                                                                                                                                                                      点存储数据,提高了节点的利用率,使得数据库能更高效地利用磁盘空间。
                                                                                                          把数据页从磁盘读取到数据库的缓冲池
                                                                                      数据库通过把页读入到内存
                                                                                                                                                                                                                                                                   · 红黑树查找次数多 (子结点数量/出度为2) , IO次数多
                                                                                                          (buffer pool - 位于RAM)
                                                                                                                                                                                                                                                                   利用预读特性,减少磁盘IO
                                                                                                                              已经在-cache hit
                                                                                                                                                                                                                                         红黑树也可以实现索引但是为什么B+树?
                                                                                                       - 先检查所需的页是否已经在缓冲池 -
                                                                                      再在内存中进行二分查找 ———
                                                                                                                             - 不在-cache miss ——— 从磁盘读取数据页到缓冲池
                                                                                                                                                                                                                                                                   因为B+树的特点是只有叶子节点存储数据,而非叶子节点不存储数据,并且节点大
                                                                                                                                                                                                                                                                   小固定,还有就是叶子结点之间通过双向链表链接的,所以,使用B+树实现索引有
                                                                                                            可以减少树的高度,从而减少查找、插入或删除操作中所需的磁盘I/O操作
                                                                                                                                                                                                                                                                   很多好处,比如我们前面提到的支持范围查询、有利于磁盘预读、有利于优化排序等
                                                                                    高扇出性(一个节点拥有许多子节点)
                                                                                                                                                                                                                                                                   等。而这些是红黑树和B树做不到的
                                                                                                            · 磁盘I/O操作通常比在内存中进行操作要慢得多,所以可显著提高索引效率
                                                                                                                                                                                                                                                                                       · 在访问表中很少一部分时使用索引才有意义 ——— 对于性别字段、地区字段、类型字段,它们可取值的范围很小,称为低选择性
                                                                                    所有索引的键值和指向实际内容的地址都在叶子结点,且是按照索引键值顺序存放的(从左到右从小到大)
                                                                                                                                                                                                                                                                                       如果某个字段的取值范围很广,几乎没有重复,即属于高选择性,此时使用B+树索引是最适合的
                                                                                                            - B+树是一棵平衡树,每个叶子节点到根节点的路径长度相同,查找效率较高;
                                                                                                                                                                                                                                                       如何判断是否需要添加索引? ——— 高低选择性判断
                                                                                                            - B+树的所有关键字都在叶子节点上,因此范围查询时只需要遍历一遍叶子节点即可;
                                                                                                                                                                                                                                                                                       如何判断? —— 可以通过SHOW INDEX结果中的列Cardinality来观察 -
                                                                                                                                                                                                                                                                                                                                  如果非常小,那么用户需要考虑是否还有必要创建这个索引
                                                                                                            - B+树的叶子节点都按照关键字大小顺序存放,因此可以快速地支持按照关键字大小进行排序;
                                                                                    优点/为什么innoDB选择B+树实现索引
                                                                                                                                                                                                                                                                          - ALTER TABLE
                                                                                                            - B+树的非叶子节点不存储实际数据,因此可以存储更多的索引数据;
                                                                                                                                                                                                                                                       创建和删除可以通过两种方法
                                                                                                                                                                                                                                                                          - CREATE/DROP INDEX
                                                                                                             B+树的非叶子节点使用指针连接子节点,因此可以快速地支持范围查询和倒序查询。
                                                                                                                                                                                                                                                       查看表中索引的信息 ——— SHOW INDEX
                                                                                                             B+树的叶子节点之间通过双向链表链接,方便进行范围查询。
                                                                                                                                                                                                                                                                                      对于辅助索引的创建,InnoDB存储引擎会对创建 ——— 共享锁允许多个事务读取同一数据资源
                                                                                                                    由于实际的数据页只能按照一棵B+树进行排序,因此每张表只能拥有一个聚集索引 ———— 多数情况下,查询优化器倾向于用聚集索引,因为能在B+树索引的叶子节点上直接找到数据
                                                                                                                                                                                                                                                                                       索引的表加上一个共享锁 (Shared Lock)
                                                                                                                   一 由于定义了数据的逻辑顺序,聚集索引能够特别快地访问针对范围值的查询 ——— 查询优化器能够快速发现某一段范围的数据页需要扫描
                                                                                                                                                                                                                                                                                       在创建的过程中,不需要重建表,因此速度较之前提高很多,并且数据库的可用性也得到了提高
                                                                                                                                                     页通过双向链表链接,页按照主键的顺序排序
                                                                                                                                                                                                                                                                                       - 由于在创建的过程中对表加上了S锁,因此在创建的过程中只能对该表进行读操作
                                                                                                                    聚集索引的存储并不是物理上连续的,而是逻辑上连续的
                                                                                                                                                                                                                                                       Fast Index Creation (快速索引创建)
                                                                                                                                                     一 每个页中的记录也是通过双向链表进行维护的,物理存储上可以同样不按照主键存储
                                                                                                                                                                                                                                                                                      - Mysql的5.6之后InnoDB允许在线DDL操作,这样在很多情况下可以避免长时间的表锁定
                                                                                                                                                                                                                                                       InnoDB1.0.x之后,仅限于辅助索引
                                                                                    聚簇索引/聚集索引 (clustered inex)
                                                                                                                   ─ 按照每张表的主键构造一棵B+树,同时叶子节点中存放的为整张表的行记录数据,也将聚集索引的叶子节点称为数据页
                                                                                                                                                                                                                                                                                       操作:只需更新内部视图,并将辅助索引的空间(内存)标记为可用,同时删除
                                                                                                                            键 (Key) : 聚集索引的键就是每行数据的主键。
                                                                                                                                                                                                                                                                                       MySQL数据库内部视图上对该表的索引定义即可
                                                                                                                                                                                                                                                                                        自 MySQL 5.6 开始,InnoDB 使用一种称为"Online DDL"的 —— 也可以用来回答Innodb加索引,这个时候会锁表吗?
                                                                                                                           - 值 (Value) ----- : 在聚集索引的上下文中,值是除了主键以外的所有数据列。
                                                                                                                                                                                                                                                                                                                                                             索引构建期间仍然可能出现锁定和阻塞。例如,在添加索引时,如果表中有许多未提
                                                                                                            和实际数据库的对应
                                                                                                                                                                                                                                                                                        技术,允许在不阻塞其他会话的情况下创建或删除索引
                                                                                                                                                                                                                                                                                                                                                             交的事务,则需要等待这些事务提交后才能开始索引构建
                                                                                                                                        :如果主键包含多个列(复合主键),聚集索引的键会包括所有这些列。在索引中,
                                                                                                                                       这些列的组合会被当作一个单一的键来排序和存储。
                                                                                                                                                                                                                                                                                                                        辅助索引的创建与删除
                                                                                                                       一 键 (Key) : 辅助索引的键是索引所基于的列 (可以是非主键列) 。
                                                                                                                                                                                                                                                                                                                        改变自增长值
                                                                                                                                                                                                                                                                                        这几类DDL操作都可以通过"在线"的方式进行操作
                                                                                                         和实际数据库的对应
                                                                                                                        指针(Pointer):辅助索引的值是一个指针,指向与键值对应的行数据的主键(也
                                                                                                                                                                                                                                                                                                                        添加或删除外键约束
                                                                                                                        就是聚集索引的键)。因为在InnoDB中,所有数据行都是按照主键的顺序存储的。
                                                                                                                                                                                                                                                                                                                        列的重命名
                                                                                                                叶子节点并不包含行记录的全部数据。叶子节点除了包含键值以外,每个叶子节点中
                                                                                                                的索引行中还包含了一个指针(主键的值)。
                                                                                                                                                                                                                                                                                                                                   ALTER TABLE tbl_name
                                                                                                                                                                                                                                                       Online DDL(在线数据定义)/InnoDB加索引时会锁表吗
                                                                                                                                                                                                                                                                                                                                    |ADD{INDEX|KEY}[index_name]
                                                                                                                   查询非主键列:首先在辅助索引中查找基于非主键列的键值。
                                                                                                                                                                                                                                                                                                                                   [index_type](index_col_name,...)[index_option]...
                                                                                    · 辅助索引 (secondary index)
                                                                                                                  - 通过书签找到主键:找到键值后,通过辅助索引中的书签/指针找到行数据的主键。
                                                                                                                                                                                                                                                                                                                                    ALGORITHM[=]{DEFAULT|INPLACE|COPY}
                                                                                                                                                                                                                                                                                                                                    LOCK[=]{DEFAULT|NONE|SHARED|EXCLUSIVE
                                                                                                                   利用主键查询聚集索引: 然后, 使用这个主键在聚集索引中找到实际的行数据。即使
                                                                                                                   查询的目标不是主键,查询过程通常还是需要通过聚集索引来获取完整的行数据。
                                                                                                                                                                                                                                                                                                                           ALGORITHM指定了创建或删除索引的算法
                                                                                                                     当我们根据非聚簇索引查询的时候,会先通过非聚簇索引查到主键的值,之后,还需要
                                                                                                                                                                                                                                                                                                                           COPY表示按照MySQL 5.1版本之前的工作模式,即创建临时表的方式
                                                                                                                     再通过主键的值再进行一次查询才能得到我们要查询的数据。而这个过程就叫做回表
                                                                                                                                                                                                                                                                                                                           INPLACE表示索引创建或删除操作不需要创建临时表
                                                                                                                      在InnoDB 中,使用主键查询的时候,是效率更高的, 因为这个过程不需要回表
                                                                                                                                                                                                                                                                                         通过新的ALTER TABLE语法,用户可以选择索引的创建方式
                                                                                                                                                                                                                                                                                                                           DEFAULT表示根据参数old alter table来判断是通过INPLACE还是COPY的算法,该参数的默认值为OFF,表示采用INPLACE的方式
                                                                                                                      联合索引的创建方法与单个索引创建的方法一样,不同之处仅在于有多个索引列
                                                                                                                                                                                                                                                                                                                                                               NONE 执行索引创建或者删除操作时,对目标表不添加任何的锁,即事务仍然可以
                                                                                                                      联合索引也是B+树,但是的键值的数量不是1,而是大于等于2
                                                                                                                                                                                                                                                                                                                                                               进行读写操作,不会收到阻塞。因此这种模式可以获得最大的并发度。
                                                                                              联合索引是指对表上的多个列进行索引
                                                                                                                                                                     对于查询SELECT*FROM TABLE WHERE a=xxx and b=xxx,显然是可以使用(a,
                                                                  - B+树索引
                                                                                                                                                                                                                                                                                                                                                               SHARE 这和之前的FIC类似,执行索引创建或删除操作时,对目标表加上一个S锁。
                                                                                                                                                                     b)这个联合索引的。对于单个的a列查询SELECT*FROM TABLE WHERE a=xxx,也
                                                                                                                                                                                                                                                                                                                                                               对于并发地读事务,依然可以执行,但是遇到写事务,就会发生等待操作。如果存储
                                                                                                                                                                     可以使用这个(a,b)索引。
                                                                                                                      它会按照最短前缀原则排序,键值的顺序从小到大排列,比如(1,2)(1,3)
                                                                                                                                                                                                                                                                                                                                                               引擎不支持SHARE模式,会返回一个错误信息。
                                                                                                                       (2,1) ,主要还是先a后b
                                                                                                                                                                     但对于b列的查询SELECT*FROM TABLE WHERE b=xxx,则不可以使用这棵B+树索
                                                                                                                                                                                                                                                                                                                           LOCK部分为索引创建或删除时对表添加锁的情况,可有的选择
                                                                                                                                                                                                                                                                                                                                                               EXCLUSIVE 在EXCLUSIVE模式下,执行索引创建或删除操作时,对目标表加上一个
                                                                                                                                                                     - 引。可以发现叶子节点上的b值为1、2、1、4、1、2,显然不是排序的,因此对于b
                                                                                                                                                                                                                                                                                                                                                               X锁。读写事务都不能进行,因此会阻塞所有的线程,这和COPY方式运行得到的状
                                                                                                                                                                     列的查询使用不到 (a, b) 的索引。除非指定a的值
                                                                                                                                                                                                                                                                                                                                                               态类似,但是不需要像COPY方式那样创建一张临时表。
                                                                                    联合索引
                                                                                                                                                        - 创建一个组合索引 (col1, col2, col3)的时候,先按照col1排序,在col1相同时再按照col2排序的,col2相同再按照col3排序。
                                                                                                                                                                                                                                                                                                                                                               DEFAULT DEFAULT模式首先会判断当前操作是否可以使用NONE模式,若不能,
                                                                                                             非最左前缀查询(违反最左前缀原则) ——— 跳过第一个字段不会被引用
                                                                                                                                                        另外,如果不涉及到联合索引,单个字段的索引也需要遵守最左前缀,即有一个字段值为"abc"时,当我们使用like进行
                                                                                                                                                                                                                                                                                                                                                               则判断是否可以使用SHARE模式,最后判断是否可以使用EXCLUSIVE模式。 (通过
                                                                                                                                                         模糊匹配时,like "ab%"是可以走索引的,而"%bc"是不行的,就是因为后者不遵守最左前缀匹配的原则了。
                                                                                                                                                                                                                                                                                                                                                               判断事务的最大并发性来判断执行DDL的模式)
                                                                                                             使用了OR条件 ——— 当查询中使用了OR条件,并且OR条件连接的各个条件分别涉及到联合索引中的不同列时,联合索引可能不会被有效使用
                                                                                                             对索引列进行了函数操作或计算 ——— 如果在查询中对联合索引的列进行了函数操作(如`ROUND()`、`SUBSTRING()`等)或者计算(如加减乘除等),数据库可能无法使用联合索引
                                                                                                                                                                                                                                         EXPLAIN 语句的基本用法很简单,你只需在你的查询前加上 EXPLAIN 关键字。
                                                                                                             · 使用了不等式条件 ——— 当对联合索引中的第一列使用不等式 (如`>`、`<`等) 时,对该列之后的列的索引可能不会被使用
                                                                                              联合索引失效的情况
                                                                                                                                                                                                                                                    当你执行 EXPLAIN 语句时,MySQL 会返回一张表,其中包含了以下列
                                                                                                             类型隐式转换 如果查询条件中的数据类型与索引列的数据类型不匹配,可能会导致数据库进行隐式类型转换,这可能会导致索引失效
                                                                                                                                                                                                                                                    id:查询的标识符,如果有多个 SELECT 组件,每个组件都会有不同的 id。
                                                                                                             使用了LIKE操作符且通配符不是后缀 ——— 使用LIKE操作符进行模糊匹配时,如果模式的开始部分是通配符(如`%`或`_`),那么联合索引可能不会被使用
                                                                                                                                                                                                                                                    select type: 查询的类型, 比如 SIMPLE (简单的SELECT, 不使用表连接或子查
                                                                                                             索引列参与了表达式或链接 ——— 如果查询中将索引列与其他列或常量进行了运算或连接操作,索引可能不会被使用
                                                                                                                                                                                                                                                    询)、PRIMARY(主查询,最外层的查询)、SUBQUERY(子查询中的第一个
                                                                                                                                                                                                                                                    SELECT) 、UNION (UNION 中的第二个或后续的 SELECT 查询) 等。
                                                                                                             如果查询中的WHERE子句条件(与联合索引无关)不足以利用索引 ——— 例如,如果WHERE子句中的条件对于联合索引的列来说过于宽泛或不相关,索引可能不会被使用。
                                                                                                                                                                                                                                                    table:输出行所引用的表。
                                                                                              指一个查询语句的执行只用从索引中就能够取得,不必从数据表中读取。
                                                                                                                                                                                                                                                    type:显示了连接类型,常见的有 ALL (全表扫描)、index (索引全扫描)、range (索
                                                                                                                                             · 假设我们有一个employees表,经常执行查询来获取特定department所有员工的name
                                                                                                                                                                                                                                                    引范围扫描)、ref (使用非唯一性索引扫描或唯一性索引前缀扫描) 、eq ref (使用唯一
                                                                                                                                             如果我们在department列上创建一个索引,这个索引会帮助我们快速找到销售部门
                                                                                                                                                                                                                                         - 返回的信息
                                                                                                                                                                                                                                                    性索引扫描)、const/system (当查询的表最多有一个匹配行时使用)等。
                                                                                              使用覆盖索引的一个好处是辅助索引不包含整行记录的所有信息,故其大小要远小于
                                                                                                                                             的所有员工,但数据库需要回到原始的数据行来获取这些员工的name。
                                                                                              聚集索引,因此可以减少大量的IO操作
                                                                                                                                                                                                                                                    possible_keys:显示可能应用在这张表上的索引。
                                                                                                                                             如果我们创建一个包含department和name的联合索引,那么这个索引就能覆盖上
                                                                                                                                                                                                                                                    key:实际使用的索引。
                                                                                                                                             述查询,因为索引本身就包含了查询需要的所有数据。
                                                                                    覆盖索引
                                                                                                                                                                                                                                                    key_len:使用的索引的长度。
                                                                                                               MySql5.6引入的一项技术,可以在联合索引遍历过程中,对联合索引中包含的字段先做判断,直接过滤掉不满足条件的记录,减少回表次数...不是无
                                                                                                              一   脑将记录返回给Server层,再交由Server层做判断,而是充分利用索引筛掉不想要的数据,再返给Server层,再舍弃一些不符合条件的,比如没有建立  
                                                                                                                                                                                                                                                    ref:显示索引的哪一列被使用了。
                                                                                                               索引的字段,索引失效的字段。比如select * from table where a > 1 and b = 2 会下推到b字段 存储引擎能筛选出b=2的再返回给Server层
                                                                                                                                                                                                                                                    rows: 估计要检查的行数, 这是一个估算值。
                                                                                                                                       减少了不必要的表数据访问,从而提高了查询的效率。在处
                                                                                                        其实是解决索引失效带来的效率低的问题的一种手段 -
                                                                                                                                       理大量数据和复杂查询时,索引下推的性能提升尤为明显
                                                                                                                                                                                                                                                    Extra: 包含不适合在其他列中显示的额外信息,如 "Using index" (仅通过索引检索
                                                                                                                                                                                                                                                    数据,不必访问表)、"Using where" (在存储引擎检索行后进行额外的过滤) 等。
                                                                                                        索引下推允许数据库在索引层面过滤数据,而不是在找到所有匹配索引项后再在表层面进行过滤
                                                                                                                                                                                                                                                   - 通过 EXPLAIN 的结果,你可以识别查询中的性能瓶颈
                                                                                                                         假设你有一个包含用户信息的表格,其中有一个复合索引包括了(姓名,年龄)。如
                                                                                                                        果你要查询名字以"张"开头、年龄大于18岁的用户,没有使用索引下推的数据库可
                                                                                                                                                                                                                                                   全表扫描(type 为 ALL)通常表示缺少有效索引。
                               noDB支持的常见的索引(按数据结构分
                                                                                                                        能会这样工作:
                                                                                                                                                                                                                                                   大量的 rows 值提示可能需要优化查询条件或增加索引来减少扫描的行数。
                                                                                                        不使用索引下推的情况
                                                                                                                        首先,数据库使用索引找到所有名字以"张"开头的记录。
                                                                                                                                                                                                                                                   Extra 列中出现的一些特定内容,如 "Using temporary" 或 "Using filesort",表明
                                                                                                                        然后,数据库读取这些记录对应的表数据,检查年龄是否大于18
                                                                                                                                                                                                                                                   MySQL 在处理查询时需要额外的资源,可能会影响性能。
                                                                                                                                                                                                                                                   从大数据量表中删除记录 ——— 先删除索引,再删除无用数据,删除完成后再添加索引
                                                                                                                       数据库在索引中查找名字以"张"开头的记录时,同时也会检查索引中的年龄信息。
                                                                                                        使用索引下推的情况
                                                                                                                                                                                                                                                                     中间表: 创建一个临时的新表,把旧表的结构完全复制过去,添加字段,再把旧表数
                                                                                                                       只有当名字和年龄条件同时满足时,数据库才会去读取表中的数据
                                                                                                                                                                                                                                                                     据复制过去,新表命名为原来的旧表名字..过程中可能会丢失一些数据
                                                                                                                                                                                                                                                   从大数据量表中添加字段
                                                                                                                                                         OLTP中查询操作只从数据库中取得一小部分数据,一般可能都在10条记录以下
                                                                                                                                                                                                                                                                     先在从库添加,因为从库停顿很久没关系,不会影响写,影响读也可以使用其它从
                                                                                                                                                                                                                                 场景题 ——— 运维
                                                                                                                       关注快速处理大量的小型事务, 如插入、更新和删除单条记
                                                                                                                                                         在这种情况下,B+树索引建立后,对该索引的使用应该只是通过该索引取得表
                                                                                                                                                                                                                                                                    库,然后进行主从切换
                                                                                    · OLTP (Online Transaction Processing, 联机事务处理)
                                                                                                                      录。这里, B+树索引对于快速定位单条记录非常有效
                                                                                                                                                                                                                                                                 使用top查看是否是mysqld造成的
                                                                                                                                                         这时建立B+树索引才是有意义的,否则即使建立了优化器也可能选择不使用索引
                                                                                                                                                                                                                                                                 show processlist 查看session情况 确定是否有消耗资源的sql在运行
                                                                                                                      在OLAP中索引的添加根据的应该是宏观的信息,最终要得到的结果是提供给决策者
                                                                                                                                                                                                                                                   CPU飙升怎么处理
                                                                                                                                                                                                                                                                 找出消耗号高的sql 查询慢日志
                                                                                                                      对于OLAP中的复杂查询,要涉及多张表之间的联接操作,因此索引的添加依然有意义
                                                                                                                                                                                                                                                                                                    可以改sql (是不是查了无用的列 是不是使用了子查询 是不是join了太多的表,是不
                                                                                                                                                                                                                                                                 使用explain 查看索引是否失效 或者是否是因为数据量太大
                                                                                    · OLAP (Online Analytical Processing,联机分析处理)
                                                                                                                                                                 Hash Join通过构建哈希表来实现高效的联
                                                                                                                                                                                                                                                                                                   是排序没有走索引,是不是用的union)、改索引、分页优化。
                                                                                                                      但是,如果联接操作使用的是Hash Join,那么索引可能又变得不是非常重要了
                                                                                                                                                                                                                                   服务器优化(增加CPU、内存、网络、更换高性能磁盘)
                                                                                                                      在OLAP应用中,通常会需要对时间字段进行索引,因大多数统计需要根据时间维度来进行数据的筛选
                                                                                                                                                                                                                                   表设计优化(字段长度控制、添加必要的索引)
                                                                                    将存储于数据库中的所有的整本书或整篇文章中的任意内容信息查找出来的技术
                                                                                                                                                                                                                                   架构部署优化 (一主多从集群部署)
                                                                  Full-text索引/全文索引
                                                                                             倒排索引 (inverted index)
                                                                                                                                                                                                                                                                 · 分库:解决并发量大的问题,通过增加数据库实例提升系统并发能力。  ———   常见场景:微服务拆分、历史数据迁移。
                                                                                                                                                  通常利用关联数组实现
                                                                                                                                                                                                                                            - 分库、分表、分库分表的概念:
                                                                                                                                                                                                                                                                一 分表:解决数据量大的问题,通过减少单表数据量提升查询速度。 ——— 常见标准:单表行数超过500万行或单表容量超过2GB。
                                                                                             在辅助表 (auxiliary table) 中存储了单词与单词自身在一个或多个文档中所在位置之间的映射
                                                                                                                                                                inverted file index,其表现形式为{单词,单词所在文档的ID}
                                                                                                                                                  拥有两种表现形式
                                                                                                                                                                                                                                                                 一分库分表:解决既有高并发又有大数据量的问题。
                                                                                                                                                                - full inverted index,其表现形式为{单词,(单词所在文档的ID,在具体文档中的位置)}
                                                                                                                                                                                                                                                       横向拆分(水平拆分):将不同记录分散到多张表中。 ———— 例子:不同用户的订单拆分到不同的表中。
                                                                            哈希索引是基于哈希表实现的一种索引结构,它通过哈希函数将键值映射到表中的一个位置以便快速访问记录,主要用于等值查询。哈希索引非常
                                                                                                                                                                                                                                                       纵向拆分 (垂直拆分) : 将记录的多个字段拆分到多张表中。 ——— 例子: 商品详情、价格、库存信息分别存储在不同的表中。
                                                                            高效,因为理论上它们可以在常数时间复杂度(O(1))内完成查找,但这种性能是在理想的哈希函数和足够大的哈希表空间的假设下得到的
                                                                                                                                                                                                                                                         根据业务需求选择:如用户、时间、地区等。
                                                                                      哈希函数:哈希索引使用一个哈希函数来计算记录的键值的哈希值。哈希函数的设计决定了键值到哈
                                                                                                                                                                                                                                            分表字段选择:
                                                                                      希表存储位置的映射方式。理想的哈希函数应该能将键值均匀分布在哈希表中,以减少哈希冲突。
                                                                                                                                                                                                                                                         常用字段:用户ID、订单ID、时间戳等。
Mysql索引和性能调优
                                                                                                                                                                                                                                   分库分表
                                                                                      键值映射:哈希函数计算得到的哈希值决定了记录在哈希表中的存储位置。当需要查询一个键值
                                                                                                                                                                                                                                                       常用算法:哈希算法、范围分片等。
                                                                                      时,哈希索引通过同样的哈希函数计算出键值的哈希值,然后直接访问对应的位置来找到记录。
                                                                                                                                                                                                                                            分表算法:
                                                                                                                                                                                                                                                       要求:一致性和不可变性。
                                                                                      处理哈希冲突:哈希冲突发生在不同的键值通过哈希函数映射到了哈希表的同一个位
                                                                                      置。处理哈希冲突的常见方法包括开放寻址法和链表法(也称为拉链法)
                                                                                                                                                                                                                                                          - UUID: 确保全局唯一性。
                                                                                                                                                                                                                                            全局ID生成方式:
                                                                                    高效的查找性能:在最佳情况下,哈希索引可以提供常数时间复杂度的查找性能。
                                                                                                                                                                                                                                                          雪花算法: 生成分布式全局唯一ID。
                                                                                    只适用于等值查询:哈希索引非常适合用于等值查询操作(如WHERE key = value),但不适合范围查询(如WHERE key > value)。
                                                                                                                                                                                                                                                         Sharding-JDBC: ——— 定位:轻量级Java框架,增强版JDBC驱动。 ——— 特点:客户端直连数据库,兼容JDBC和ORM框架。
                                                                                    不支持排序和分组操作:由于哈希索引基于哈希表,它们不保留键值之间的顺序,因此不适合执行需要排序或分组的操作。
                                                                                                                                                                                                                                                      空间效率:与其他类型的索引相比,哈希索引通常更加紧凑,尤其是当键值的大小远大于哈希值的大小时。
                                                                                                                                                                                                                                                         · Mycat: · —— 定位:分布式关系型数据库中间件。 —— 特点:支持分布式SQL查询,兼容MySQL通信协议。
                                                                                                                                                                                                                                                          - 定义:读写分离是将数据库的读操作和写操作分离到不同的数据库实例上,从而提高系统的读写性能和可扩展性。
                                                                            InnoDB存储引擎支持的哈希索引是自适应的,InnoDB存储引擎会根据表的使
                                                                                                                                                                                                                                            读写分离的概念:
                                                                            用情况自动为表生成哈希索引,不能人为干预是否在一张表中生成哈希索引
                                                                                                                                                                                                                                                          目的:提升系统的读写性能,减轻主库压力,提高读操作的并发处理能力。
                                                                                      内存数据库和一些键-值存储系统经常使用哈希索引来实现高效的数据访问
                                                                                                                                                                                                                                                                         负责所有的写操作和更新操作。
                                                                                                                                                                                                                                                          - 主库 (Master):
                                                                                      由于哈希索引的限制,它们通常不适用于需要范围查询、排序或分组的场景
                                                                                                                                                                                                                                                                         保持数据的一致性和完整性。
                                                                                                                                                                                                                                            读写分离的组成:
                                                                                                                                                                                                                                   读写分离
                                                                                                                                                                                                                                                                        负责所有的读操作。
                                              - 联合查询要能正确使用需要遵循最左原则
                                                                                                                                                                                                                                                          从库 (Slave) :
                                                                                                                                                                                                                                                                        从主库同步数据,通常是异步同步。
                                               左或者左右模糊匹配,也就是like %xxx和like %xxx%都会索引失效
                                                                                                                                                                                                                                                                        主库同步数据到从库,通常是异步复制。
                                               在查询中对索引列做了计算、函数、类型转换操作(因为转换后可能导致不是有序的了)
                                                                                                                                                                                                                                                                        确保从库中的数据与主库的一致性。
                                               在WHERE字句中,如果在OR前的条件列是索引列,而在OR后的条件列不是索引列
                                                                                                                                                                                                                                            读写分离的实现方式:
                                                                                                                                                                                                                                                                        通过负载均衡器分配读请求到不同的从库。
                                               不等于(<>!=)和NOT IN 使用IS NULL 或 IS NOT NULL
                                                                                                                                                                                                                                                             负载均衡:
                                                                                                                                                                                                                                                                        提高读操作的并发处理能力。
                                               注意有个时候上述可能并不一定完全就用不到索引,这取决于优化器的成本计算。比如like '%x'就
                                                                                                                                                                                                                                                                                                                                                                  在番茄Plus应用中,假设有一个功能是用户可以查看他们的历史番茄钟使用记录,这
                                               不一定一定会失效,如果满足覆盖索引的话。这里但是是无序的,所以也就只能全扫描二级索引树
                                                                                                                                                                                                                                                                                                                                                                   个记录包括每个番茄钟的开始时间、结束时间、休息时间以及完成的任务类型。随着
                                                                                                                                                                                                                                                                                                                                                                  用户使用应用的时间增长,每个用户累积的数据量逐渐增加。这些数据可能被存储在
                                                                                                                                                                                                                                                                                                                                                                  MySQL数据库中,用于生成历史统计和分析报告。
                                                                     前缀索引是指在创建索引时,只对字段的一部分内容建立索引,而不是整个字段。这种方法在处理非
                                                                    常长的文本字段时非常有用,如URLs、email地址或长文本,因为它可以显著减少索引所占的空间。
                                                                                                                                                                                                                                                                                                                                                                   当用户尝试加载他们的历史记录时,应用需要从数据库中检索大量数据,并可能进行
                                                                                                                                                                                                                                                                       ·比如是某一次线下报警出现了慢SQL,或者是接口RT比较长,做了性能分析发现瓶颈是在SQL查询上面都可以。但是不管怎么样,一定要有背景
                                                                                                                                                                                                                                                                                                                                                                  一些复杂的计算,比如计算总的专注时间、平均每次专注的时间等。如果这些数据没
                                                        前缀索引优化
                                                                            ORDER BY 无法使用前缀索引: 当使用前缀索引时, 因为索引并未包含字段的完整
                                                                                                                                                                                                                                                                                                                                                                  有得到有效的索引支持或查询优化,就可能导致查询速度缓慢。
                                                                           值,所以无法保证完全的排序,因此无法使用前缀索引进行ORDER BY操作。
                                                                                                                                                                                                                                                                                                                                                                                     实现分页机制,只加载用户请求查看的那部分数据,而不是一次性加载所有历史记录
                                                                            无法作为覆盖索引:由于前缀索引只包含部分数据,当查询只需要索引中的列时,仍
                                                                                                                                                                                                                                                                                                                                                                  - 解决办法:分页和延迟加载
                                                                            然需要回到原始数据行来获取完整的列数据
                                                                                                                                                                                                                                                                                                                                                                                     可以在用户界面实现"滚动加载更多"功能,当用户滚动到页面底部时再加载更多数
                                                                                                                                                                                                                                                                                                                                                                                     据,这样可以减轻服务器的负载,提高响应速度。
                                                                    覆盖索引是指一个索引包含了查询所需的所有字段,因此查询可以直接通过索引来获取数据,而无需回到数据表
                                                                    中查找。这大大减少了IO操作,因为索引通常比原始的数据表小很多,且在数据库的缓存中被更频繁地访问。
                                                                                                                                                                                                                                                                                                              查看是否开启: SHOW VARIABLES LIKE '%slow_query_log%';
                                                         主键索引最好是自增的,插入一条新记录,都是追加操作,不需要重新移动数据,不会造成页分裂而导致的性能问题以及内存碎片问题,影响查询效率
                                                                                                                                                                                                                                                                                                              开启当前数据库, 重启失效: SET GLOBAL slow_query_log = 1;
                                                        索引最好设置为NOT NULL,第一,索引列存在NULL,会导致优化器在做索引选择的时候更加复杂,为NULL的列会使索引、索引统计、值比较更加复杂。比如进行索引
                                                                                                                                                                                                                                                                                               开启慢查询日志功能
                                                                                                                                                                                                                                                                                                              永久生效:在my.cnf里面:
                                                        统计时,count会统计NULL的行,第二 NULL值是一个没意义的值,但是会占用物理空间,所以会带来存储空间的问题 在行格式里至少会用1字节空间存储NULL值列表
                                                                                                                                                                                                                                                                                                              [mysqld]
                                                                                                                                                                                                                                                                                                              # 开启慢查询
                                                        使用联合索引进行排序 ---- 就是使用索引来天然排序,但是没必要单独又创建一个独立索引,创建一个联合索引即可
                                                                                                                                                                                                                                                                                                              slow query log=ON
                                                                              左或者左右模糊匹配,也就是like %xxx和like %xxx%都会索引失效
                                                                                                                                                                                                                                                                                                              # 指定存储慢查询日志的文件。如果这个文件不存在,会自动创建
                                             一 索引优化
                                                                                                                                                                                                                                                                                                              slow_query_log_file=/var/lib/mysql/slow.log
                                                                              在查询中对索引列做了计算、函数、类型转换操作(因为转换后可能导致不是有序的了)
                                                                                                                                                                                                                                                                                                               - 查看阈值:SHOW VARIABLES LIKE 'long_query_time%';注意必须是超过阈值,等于的不算
                                                                                                                                                    select * from t_table where a > 1 and b = 2(大于1 而b无序 要使用索引必须索引列
                                                                                                                                                                                                                                                                                                              一 开启当前数据库,重启失效: set global long_query_time=3;
                                                                                                                                                                                                                                                                                               设置慢查询的时间阈值。
                                                                                                                                                    select * from t table where a >=1and b = 2 (存在a = 1时,b有序所以可以使用
                                                                                                                                                                                                                                                                                                               永久生效: [mysqld]
                                                                                                        注意并不是查询过程中使用到了联合索引查询,就代表联合索引中的所有字段
                                                                              联合查询要能正确使用需要遵循最左原则
                                                                                                                                                                                                                                                                                                               long_query_time=1
                                                                                                                                                                                                                                                                                   慢查询日志
                                                                                                        都用到了联合索引进行索引查询, 比如说范围查询后面的字段可能无法用到
                                                                                                                                                    select * from t_table where a between 2 and 8 and b = 2
                                                                                                                                                                                                                                                                                                              查看慢查询日志文件中的总记录数: SHOW GLOBAL STATUS LIKE '%Slow_queries%';
                                                                                                                                                    select * from t_user where name like 'j%' and age = 22,也是存在一样的完全一
                                                                                                                                                                                                                                                                                               基本查看慢查询日志
                                                                                                                                                                                                                                                                                                                         - Query time:实际查询时间,单位是秒
                                                                                                                                                                                                                                                                                                              显示每一条的 <del>(</del> Lock time: 锁时间
                                                                              在WHERE字句中,如果在OR前的条件列是索引列,而在OR后的条件列不是索引列
                                                                                                                                                                                                                                                                                                                         - select sleep(4): 超时的语句
                                                                                                        - possible_keys:表示可能用到的索引
                                                                                                                                                                                                                                                                                                               - 手工分析日志,查找,分析Sql是一个体力活,,可以使用mysqldumpslow日志分析工具进行快速统计
                                                       下 防止索引失效
                                                                                                        - key:实际用到的索引,如果这一项为NULL,说明没有用到索引
                                                                                                                                                                                                                                  - SQL优化(避免SQL命中不到索引) ——— 如何进行sql优化
                                                                                                         key_len: 查看索引的长度
                                                                                                                                                                                                                                                                                                               实际中查询示例: # 得到返回记录集最多的10个SQL
                                                                                                                                                                                                                                                                       慢Sql如何定位
                                                                                                        rows:表示扫描的数据行数,预估值
                                                                                                                                                                                                                                                                                               使用日志分析工具查看
                                                                                                                                                                                                                                                                                                               mysqldumpslow -s r -t 10 /var/lib/mysql/slow.log
                                                                                                                       ✓ All (全表扫描)
                                                                                                                                                                                                                                                                                                               # 得到访问次数最多的10个SQL
                                                                                                                        - index:全索引扫描....在like失效的时候当有覆盖索引时,会走这个而不是All,count(*)也是
                                                                                                                                                                                                                                                                                                               mysqldumpslow -s c -t 10 /var/lib/mysql/slow.log
                                                                                                                        - range:索引范围扫描...在where中 < > in between等...尽量从这个级别开始
                                                                    使用explain命令查看执行计划到底是否用了索引 ——— 参数·
                                                                                                                                                                                                                                                                                                               # 得到按照时间排序的前10条里面含有左连接的查询语句
                                                                                                                                                                                                                                                                                                               mysqldumpslow -s t -t 10 -g "left join" /var/lib/mysql/slow.log
                                                                                                         type:数据扫描类型
                                                                                                                        __ref:非唯一索引扫描...就是索引列值不唯一,有重复,即使找到了第一条数据,也不能停止扫描,要
                                                                                                                         在目标值附件的小范围扫描,因为索引是有序的,即使有重复值,也是在一个非常小的范围内扫描
                                                                                                                                                                                                                                                                                                               #另外建议使用这些命令时结合|和more使用,否则出现爆屏的情况
                                                                                                                                                                                                                                                                                                               mysqldumpslow -s r -t 10 /var/lib/mysql/slow.log | more
                                                                                                                        - eq_ref:唯一索引扫描...使用主键或唯一索引时,通常发生在多表联查中,在被驱动表中关联字段使用上索引
                                                                                                                                                                                                                                                                                    服务监控
                                                                                                                        const:结果只有一条的主键索或唯一索引 直接是主键或者唯一索引与常量值进行比较,const是与常量进行
                                                                                                                        比较,查询效率更快,而eq_ref通常用于多表联查中
                                                                                                                                                                                                                                                                                       避免不必要的列
                                                                                                                一 Using filesort:包含group by操作,但是无法利用索引完成排序操作 使用文件排序 效率很低
                                                                                                                                                                                                                                                                                                         延迟关联:使用连接查询,查出id作为一张表再与自己作连接查询
                                                                                                                                                                                                                                                                                       分页优化 (优化深分页)
                                                                                                                一 Using temporary:使用了临时表保存中间结果 常见于order by 、分组 group by 效率低
                                                                                                                                                                                                                                                                                                         书签方式:先通过子查询查到起始id,再利用where条件和limit进行筛选
                                                                                                                - Using index:所需数据只需在索引即可全部获得,就是覆盖索引
                                                           ———   在决定创建索引之前,需要分析查询频率和效率。对于频繁查询的列,可以创建索引来加速查询,但对于不经常查询或者数据量较少的列,可以不创建索引
                                                                                                                                                                                                                                                                                                 自增主键
                                               选择适合的索引类型 ——— MySQL提供了多种索引类型,如B+Tree索引、哈希索引和全文索引等。不同类型的索引适用于不同的查询操作,需要根据实际情况选择适合的索引类型
                                                                                                                                                                                                                                                                                                设置NOT NULL
                                                         - 尽量不要选择区分度不高的字段作为索引,比如性别。但是也并不绝对,对于一些数据倾斜比较严重的字段,虽然区分度不高,但是如果有索引,查询占比少的数据时效率也会提升
                                                                                                                                                                                                                                                                                                             模糊查询,避免使用'%x'
                                                          - 联合索引是将多个列组合在一起创建的索引。当多个列一起被频繁查询时,可以考虑创建联合索引
                                                                                                                                                                                                                                                                                                             联合索引,满足最左前缀
                                                          联合索引可以通过索引覆盖而避免回表查询,可以大大提升效率,对于频繁的查询,
                                               考虑索引覆盖
                                                                                                                                                                                                                                                                                                            - 低版本避免使用OR , OR前面使用索引后面没用。可以使用union替换
                                                           可以考虑将select后面的字段和where后面的条件放在一起创建联合索引。
                                                                                                                                                                                                                                                                       优化慢Sql的几种方式
                              索引的设计原则
                                                                                                                                                                                                                                                                                                             列上使用函数,运算,类型转换
                                                              创建过多的索引会占用大量的磁盘空间,影响写入性能。并且在数据新增和删除时也需要对索
                                               避免创建过多的索引
                                                              引进行维护。所以在创建索引时,需要仔细考虑需要索引的列,避免创建过多的索引。
                                                                                                                                                                                                                                                                                                             避免使用不等于!= <> 或者not in、IS NULL、IS NOT NULL
                                                             一 索引列的长度越长,索引效率越低。在创建索引时,需要选择长度合适的列作为索引列。对于文本列,可以使用前缀索引来减少索引大小。
                                                                                                                                                                                                                                                                                                 优化子查询:使用JOIN替换子查询。因为子查询会产生临时表。临时表的创建和销
                                                                                                                                                                                                                                                                                                 毁比较费时间,特别是查询结果集大的
                                                            虽然索引不建议太长,但是也要合理设置,如果设置的太短,比如身份证号,但是只
                                                            把前面6位作为索引,那么可能会导致大量锁冲突。
                                                                                                                                                                                                                                                                                                 - 小表驱动大表:因为总会遍历驱动表,可以在被驱动表建立索引
                                                          多用执行计划分析,因为随着数据库的数据量变化、索引数量变化,最终使用的索引
                                                                                                                                                                                                                                                                                                 适当增加冗余字段
                                               执行计划分析 -
                                                          可能也不太一样, 所以需要经常查看索引是否有使用对。
                                                                                                                                                                                                                                                                                                 避免join太多的表 可以优化成子查询 (将三个left join 变成两个 加上having 1 > 0使得MySql优化器强行使用
                                                                                                                                                                                                                                                                                                 子查询而不被优化成原来的连接查询 降低笛卡尔积) 看速度是否提升
                                                       主键索引是基于表中的主键字段创建的。主键具有唯一性,通常是在表创建时同时定
                                                                                                                                                                                                                                                                                               重复利用索引来自然排好序,而非使用文件排序
                                                       义的。这种索引保证了表中每条记录的唯一性,使得基于主键的查询非常快速
                                                                                                                                                                                                                                                                                                 如果没有去重业务,尽量使用union all。最好手工将条件放到各自的查询中,而不要
                                                                        唯一索引在 MySQL 中可以允许 NULL 值的,但是这些NULL的表现是未知的,未知
                                                                                                                                                                                                                                                                                       union优化
                                                                                                                                                                                                                                                                                                 等到最后去过滤 不然可能会失效
                                                                        就是他们不相等,但是也不能说他们不等
                                                      唯一索引允许NULL值吗
                                                                                                                                                                                                                                                             设置Binlog_group_commit_sync_delay和Binlog_group_commit_sync_no_delay_count参数,延迟刷盘时机,增加额外的故意等待。即使MySql进
                                                                         每个NULL值都被视为互不相同 ——— 所以即使一个列被定义了唯一索引,它也可以包含多个NULL值
                                                                                                                                                                                                                                                             程中途挂了,也没有丢失数据的风险,因为binlog早被写入到page cache中了,只要系统没有宕机,缓存在page cache里的binlog就会被持久化到磁盘
                                                                         唯一性索引需要保证索引列的唯一性,因此在插入数据时需要检查是否存在相同的索
                                                                                                                                                                                                                                                             将sync binlog设置为大于1的值(100-1000),表示每次提交事务的write,但累计了N个事务才
                                                                         引值,这会对插入性能产生一定的影响
                                                                                                                                                                                                                                  MySql的磁盘I/O很高,有什么优化的办法
                                                                                                                                                                                                                                                             fsync。延迟了binlog刷盘的时机,但是风险是,主机掉电会丢失N个事务中的binlog日志
                                                      唯一性索引有什么缺点吗
                                                                         如果需要更新唯一性索引列的值,需要先删除旧记录,再插入新记录,这会对更新操
                                                                                                                                                                                                                                                             innodb flush log at trx commit设置为2,每次提交事务都只是将其redo log buffer中的redo log写入到操
                                                                                                                                                                                                                                                             作系统中的Page Cache。风险是主机掉电肯会丢失数据
                                            - 唯一索引 -
                                                                                                    因为唯一性索引能够快速定位到唯一的记录,而非唯一性索引则需要扫描整个索引并
                                                      唯一性索引查询更快吗
                                                                       唯一性索引查询通常会比非唯一性索引查询更快一
                                                                                                                                                                                                                                                               一主多从,主负责读写,从负责读
                                                                                                    匹配符合条件的记录
                             按字段特性分
                                                                                                                                                                                                                                                                                            数据写入到master主节点,更新binlog
                                                                                 MySQL通常使用B树(或变种如B+树)作为唯一性索引的数据结构。这种结构允许高效的数据检索和插入操作
                                                                                                                                                                                                                                                                                            master创建一个dump线程向slave从节点推送binlog
                                                                                 当插入一个新行或更新现有行的索引列时,MySQL首先在索引中检查是否已经存在相同的键
                                                                                                                                                                                                                                                               · 原理: 主从复制是基于binlog日志文件实现的 ·
                                                                                                                                                                                                                                                                                           - slave从节点连接到master的时候,创建一个IO线程接受binlog,并记录到relay log中继日志中
                                                                                 如果发现索引列的新值已经存在于唯一性索引中,MySQL将阻止该插入或更新操
                                                                                                                                                                                                                                                                                           slave从节点再开启一个sql线程读取relay log日志,执行sql,完成同步
                                                       MySQL是如何保证唯一性索引的唯一性的
                                                                                 作,并返回一个错误
                                                                                                                                                                                                                                                                                           slave记录自己的binlog
                                                                                                                当一个事务正在修改索引列时,其他事务对相同键值的修改会被适当地阻塞,直到第
                                                                                 InnoDB中事务机制和锁定协议帮助维护索引的唯一性 ——
                                                                                                                                                                                                                                                                         产生原因:当有大并发的更新操作时,从节点中的读取binlog线程只有一个,当某个SQL在从服务器上执行的
                                                                                                                 一个事务提交或回滚,确保了数据的一致性和唯一性
                                                                                                                                                                                                                                                                         时间稍长,主服务器的SQL大量积压,未被同步到从服务器里面,这就导致了主从不一致,主从延迟
                                                                                 在实际写入数据到磁盘之前,MySQL也会执行约束检查,确保不会违反唯一性约
                                                                                                                                                                                                                                                               - 主从延迟 -
                                                                                                                                                                                                                                                                                   写操作后的读操作重新指定给数据库主服务器
                                                       前缀索引是指在创建索引时,只对字段的一部分内容建立索引,而不是整个字段。这种方法在处理非
                                                                                                                                                                                                                                                                         解决办法 😽
                                                                                                                                                                                                                                                                                   - 读从机失败后再读一次主机
                                                      常长的文本字段时非常有用,如URLs、email地址或长文本,因为它可以显著减少索引所占的空间。
                                                                                                                                                                                                                                                                                   关键业务读写操作全部指向主机,非关键业务采用读写分离
                                                               ORDER BY 无法使用前缀索引: 当使用前缀索引时, 因为索引并未包含字段的完整
                                                              值,所以无法保证完全的排序,因此无法使用前缀索引进行ORDER BY操作。
                                                                                                                                                                                                                                                                                同步复制:MySql提交事务的线程必须要等待所有从库的复制成功响应,才返回客户端结果。这种方式性能太
                                                                                                                                                                                                                                                                                差,而且可用性也很差,主库和所有从库任何一个一个数据库出问题,都会影响业务
                                                              无法作为覆盖索引:由于前缀索引只包含部分数据,当查询只需要索引中的列时,仍
                                                              然需要回到原始数据行来获取完整的列数据
                                                                                                                                                                                                                                                                                异步复制(默认模型): MySql主库提交事务的线程并不会等待binlog同步到各从库,就返回客户端结
                                                                                                                                                                                                                                                               - 主从复制还有哪些模型
                                                                                                                                                                                                                                                                                果。这种模式一旦主库宕机,数据就会发生丢失
                                             · 聚簇索引/聚集索引 (clustered inex)
                                                                                                                                                                                                                                                                                半同步复制:MySql5.7版本之后新增的,只要一部分从库复制成功,就能响应成功。兼顾了上面两
                                                                                                                                                                                                                                                                                种方案的优点,即使主库宕机,至少还有一个从库有最新的数据,不存在数据丢失的风险
                                             二级索引/辅助索引 (secondary index)
                                                                                                                                                                                                                                                                            数据库服务器搭建主从集群,一主一从,一主多从都可以
                                                                                                                                                                                                                                                                            数据库主机负责读写操作,从机只负责读操作
                                                                                                                                                                                                                                                                 - 基本原理
                              按字段个数分
                                                                                                                                                                                                                                                                            数据库主机通过复制将数据同步到从机,每台数据库服务器都存储了所有数据业务
                                             聚合索引/联合索引
                                                                                                                                                                                                                                                                            业务服务器将写操作发给数据库主机,将读操作发给数据库从机
                                                                                                                                                                                                                                                                            程序代码封装:比如使用sharding-jdbc
                                                                                                                                                                                                                                   如何维持高可用 ——— 高可用 -
                                                                                                                                                                                                                                                                            中间件封装:独立一套系统,对于业务服务器访问中间件和访问数据库没有区别,比
                                                                                                                                                                                                                                                                            如用sharding-proxy
                                                                                                                                                                                                                                                                         垂直分库: 以表业务为依据, 按照业务归属的不同, 将不同的表拆分到不同的库中
                                                                                                                                                                                                                                                                         水平分库:以字段为依据,按照一定的策略(hash、range),将一个库中的数据拆
                                                                                                                                                                                                                                                                          分到多个库中,还是将表拆开分散到各个库
                                                                                                                                                                                                                                                                         垂直分表: 以字段为依据, 按照字段的活跃性, 将表中的字段拆到不同的表
                                                                                                                                                                                                                                                                         水平分表:以字段为依据,按照一定的策略(hash、range)将一个表中的数据拆到多
                                                                                                                                                                                                                                                                                            范围路由:选取有序的数据列(整形、时间戳等)作为路由的条件,不同范围的被分
                                                                                                                                                                                                                                                                                            散到不同的表中。优点是随着数据的增加可以平滑的扩充新的表,缺点是可能会造成
                                                                                                                                                                                                                                                                                            分布不均匀,比如某个分段数据量很好,其它分段数据量很多
                                                                                                                                                                                                                                                                                            Hash路由:选取某个列或多个列的组合的值进行Hash计算,然后根据Hash计算的结
                                                                                                                                                                                                                                                                         水平分表有哪几种路由方式
                                                                                                                                                                                                                                                                                            果分散到不同的数据库表中。优缺点和范围路由刚好相反
                                                                                                                                                                                                                                                                                            配置路由: 使用一张独立的表来记录路由信息。优点是非常灵活 缺点就是需要再次
                                                                                                                                                                                                                                                                                            - 查询一次,如果路由表本身也很大的话,可能需要对路由表进行分库分表造成死循环
                                                                                                                                                                                                                                                                                 第一阶段: 在线双线, 查询走老库; 就是添加一个新库, 新的往里面写, 老库还是和
                                                                                                                                                                                                                                                                                 : 以前一样,读和写。然后使用数据迁移程序,将旧库中的历史数据迁移到新库。最好
                                                                                                                                                                                                                                                                                 可以使用定时任务,将新旧库的数据对比,把差异补齐
                                                                                                                                                                                                                                                                                 第二阶段:在线双写,查询走新库:现在就是基本可以使用新库了,查询和写都在新
                                                                                                                                                                                                                                                                  不停机扩容如何实现
                                                                                                                                                                                                                                                                                一 库,但是为了确保觉得安全,还是不要下线旧库,依旧往旧库里面写。等待新库稳定
                                                                                                                                                                                                                                                                                 第三阶段: 旧库下线, 旧库不再写入新的数据。经过一段时间, 确定旧库没有请求之
                                                                                                                                                                                                                                                                                 后,下线旧库
                                                                                                                                                                                                                                                                  常见的分库分表中间件有哪些: Sharding-JDBC、MyCat、TDDL (阿里巴巴) 、Vitess
                                                                                                                                                                                                                                                                                   事务一致性问题 -
                                                                                                                                                                                                                                                                                                 使用柔性事务如TCC 或者消息队列实现最终一致性
                                                                                                                                                                                                                                                                                                                在应用层将多个查询结果进行合并
                                                                                                                                                                                                                                                                                                                使用分布式数据库中间件:如Sharding-JDBC、MyCat、TDDL等
                                                                                                                                                                                                                                                                                   - 跨库关联查询问题 (会导致无法直接跨库查询)
                                                                                                                                                                                                                                                                                                                增加一些冗余字段,减少关联查询
                                                                                                                                                                                                                                                                  分库分表会带来哪些问题
                                                                                                                                                                                                                                                                                                                 使用数据异构,将需要跨库的数据异构到其它存储结构中,如ES
                                                                                                                                                                                                                                                                                                                使用分布式的计算框架,比如Hadoop、Spark等
```

索引是什么:就像书的目录,能大致确定范围,从最底下开始选取最小或者最大的放

到上一层,直到顶层。总的来说相当于排好序的数据结构

聚集索引的叶子节点直接包含了数据本身,而不是仅仅包含指向数据的指针

数据迁移和扩容问题 ——— 可以采用阿里巴巴的DataX工具

主键冲突问题(分库分表之后原先的主键自增不能用,需要采用其它方式产生全局唯

设置自增步长

可以使用UUID

一分布式ID算法:如雪花算法、Redis的分布式ID

辅助索引的叶子节点包含索引键值以及指向数据行的指针

聚集索引中数据库表的行是按照聚集索引键的顺序来存储的