1. B+树索引查询流程
2. 索引的代价

索引虽好，可不要贪杯哦，在时间和空间上都会有代价。空间上每建立一个索引，都要建立一棵B+树，树中的每个节点都是一个索引页，每个页的存储空间都是16KB，这将会占用很大的存储空间。时间上，对记录的增删改操作会破坏节点和记录的排序，如果建立了很多索引，那么每个索引对应的B+树都需要进行维护操作，影响性能。

且执行查询语句之前，需要生成一个执行计划，需要计算使用不同索引执行查询时所需要的成本，选取成本最低的那个索引执行查询，当索引过多时会导致成本分析耗时过多，影响查询语句性能。

1. 扫描区间和边界条件

全表扫描表示扫描表中的所有记录，判断每一条记录是否符合搜索条件。如果我们只需要扫描某个区间或者某些区间中的记录，就可以减少需要扫描的记录数量。待扫描记录的id值所在的区间称为扫描区间，把形成这个扫描区间的搜索条件称为形成这个扫描区间的边界条件。

单点扫描区间为只包含一个值的扫描区间；范围扫描区间为包含多个值的扫描区间，查询列表是需要最终结果集中需要查询出来的列名称，每次在扫描区间中获取一条二级索引记录，就需要根据该二级索引记录的主键列的值执行回表操作，到聚簇索引中找到相应的聚簇索引 记录。如果想使用某个索引来执行查询，但是又无法通过搜索条件形成合适的扫描区间来减少需要扫描的记录数量时，则不考虑使用这个索引进行查询。

在扫描区间中，只要通过B+树定位到了该扫描区间中的第一条记录，就可以沿着记录所在页的单向链表向后扫描，如果该页的记录扫描完了，就通过页之间的双向链表到下一页中扫描，直到某条记录不符合形成该扫描边界的边界条件为止。

1. 索引用于排序
2. 概述

查询语句中，经常使用ORDER BY 子句对查询出来的记录按照某种规则排序，一般情况下，将记录加载到内存中，使用排序算法排序，MySQL中这种在内存或磁盘中进行排序的方式称为文件排序。如果将索引用于排序中，就可以省去性能上的消耗了，即直接在ORDER BY子句中使用索引列进行排序，按照索引的B+树的方式进行排序。

1. 不可使用索引排序的情况
2. ASC , DESC混用

对于使用联合索引进行排序的场景，要求各个排序列的排序规则是一致的，要么都是升序（ASC），要么都是降序（DESC）。

1. 排序列包含非同一个索引的列

如果排序列中有不在当前索引中的列，这些列是无法按照当前索引的结构进行排序的。

1. 排序列是某个联合索引的索引项，但是这些排序列在联合索引中不连续

联合索引的列的顺序都是在定义索引时设置好的，索引的结构也是按照这种顺序搭建的，如果排序列不是联合索引中的连续列，就无法一一按照索引中的顺序进行排序。

1. 用来形成扫描区间的索引列于排序列不同
2. 排序列不是以原列名出现在ORDER BY子句中

索引列以修饰过的形式出现在ORDER BY子句中，那么InnoDB直接以修饰后的索引列名进行排序，不会按照原有的索引列名排序。

1. 索引用于分组

原来的分组会创建用于统计的临时表，在扫描聚簇索引的记录时将统计的中间结果填入临时表中，扫描完记录后，再将临时表中的结果作为结果集发送给客户端。分组也可以利用索引，都是树结构的，使用索引就不再需要创建临时表了。

1. 回表的代价

对于InnoDB存储引擎，索引中的索引页都必须存放在磁盘中，等到需要使用时加载到内存使用。由于每个页之间是不一定相邻的，因此在进行回表操作时，如果对应的聚簇索引记录所在的页面不在内存中，就需要将该页面从磁盘加载到内存中，由于要读取很多主键值不连续的聚簇索引记录，分布在不同的索引页中，毫无规则，因此会造成大量的随机I/O。

所以到底需不需要进行二级索引+回表，还是直接进行全表扫描，这得由查询优化器事先针对表中的记录进行统计，计算需要执行回表操作的记录数，如果需要执行回表操作的记录数越多，则越倾向于使用全表扫描。

1. 如何更好地创建和使用索引
2. 只为用于搜索，排序，分组的列创建索引

只为了出现在WHERE子句，ORDER BY子句，GROUP BY子句，连接子句中的连接列创建索引，仅出现在查询列表中的列就没必要创建索引了。

1. 考虑索引列中不重复值的个数

在为某个列创建索引时，如果该列中不重复值的个数比例越低，则说明该列包含过多重复值，那么会进行过多的回表操作。

1. 索引列的类型进行小

索引列尽量使用较小的类型，因为越小索引占用的存储空间就越小，在一个索引页中就可以存放更多的记录，磁盘I/O带来的性能损耗就越小。这对于主键也是适用的，因为不仅聚簇索引会存储主键，二级索引都会存储主键。

1. 覆盖索引

如果只在查询列表中包含索引列，那么就能彻底告别回表，直接从二级索引记录中读出查询列表中的列即可。这种索引中已经包含了所有需要读取的列的查询方式称为覆盖索引。

1. 索引列名需要单独出现

如 WHERE key \* 2 < 4

这种WHERE子句InnoDB不会尝试简化表达式，会直接将 key \* 2当做列名尝试形成扫描区间，明显无法形成合适的扫描区间来较少需要扫描的记录数量。

而WHERE key < 4/2

key就是单独出现的，能够通过key列进行索引。