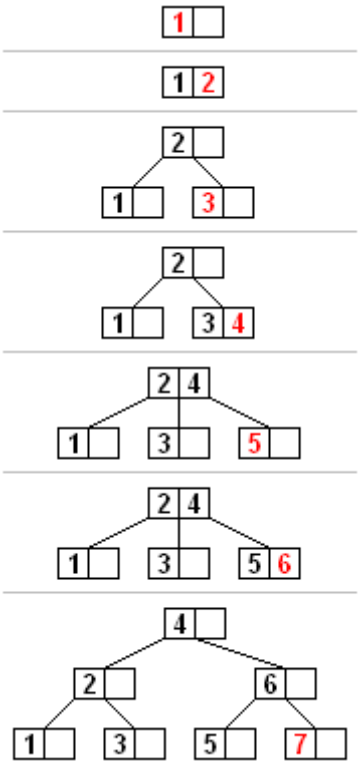
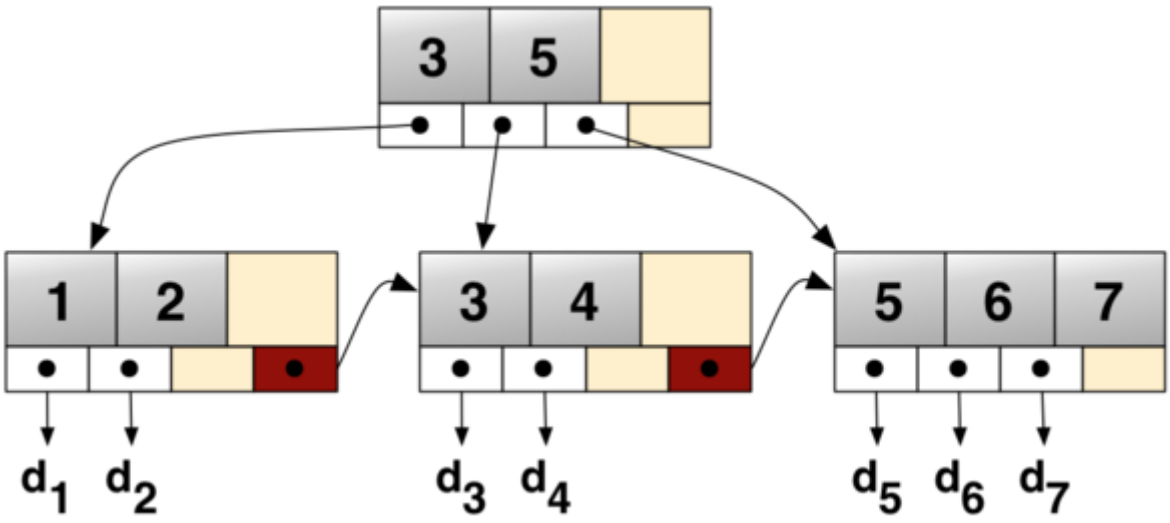
为什么是索引这种数据结构

> 此处背景为5.7的mysql InnoDB数据库引擎

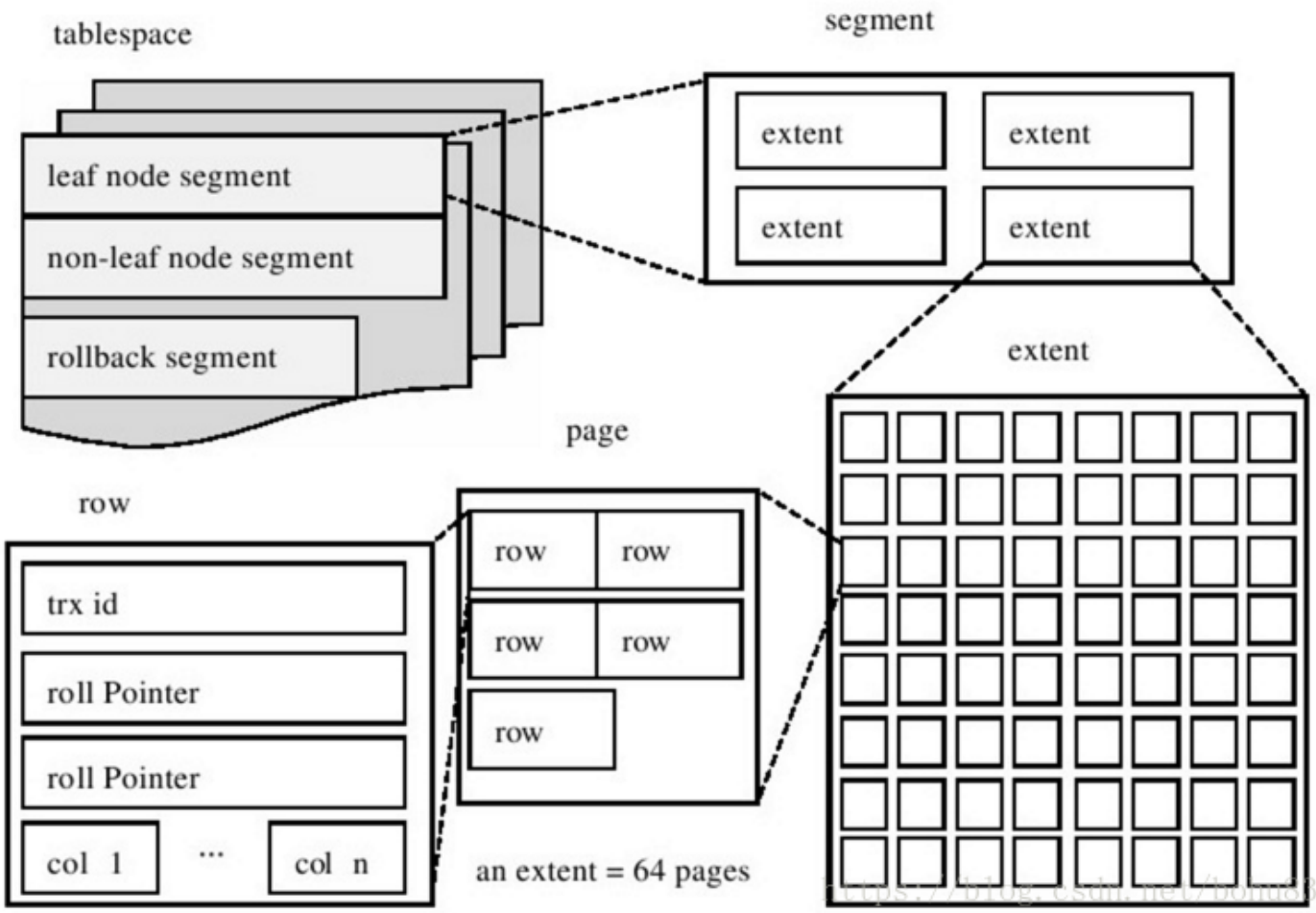
数据结构

数据结构	优点	缺点	场景
哈希	查找、添加速度快，搭配拉链法， 碰撞也还好	区间查询、order by这种耗时巨大	等值查询 NOSQL引擎 Mercached
有序数组	区间查询快，等值查询O(log(n))	更改慢，并发时麻烦多	静态存储引擎
搜索树 （二叉搜索树 ）	O(log(n))的查询、更新效率	树太高了， 导致磁盘访问数据块次数多，且此处耗时长， 10ms机械硬盘寻址， 空间上也复杂	map、 set中的升级 二叉红黑树之类的
B树 [BalanceTree]	多叉树，减少高度， 提高了查找效率，减少寻址次数， 方便一次取多个数据出来缓存	当数据跨叶（层）时， 就需要中序遍历， 同时B树的调整也是一个问题	
B+树	升级版肯定好一点： 叶子才存数据， 非叶子可以存更多的信息， 层级少，效率高 （当然如果B树的数据离根很近， 那肯定会慢一点）， 叶子数据间链表串联，有序， 区间查询快，全节点遍历也快	随机IO导致性能降低 （这其中包括了， 主键非有序导致数据迁移 以及空间碎片， 或者增删改的空间碎片， 或者随机访问量本来就大	文件索引、 数据库索引





InnoDB是什么结构



数据怎么存放的

> InnoDB与MyISAM不同的是，将数据直接存放在了主键索引文件中，保存了所有数据信息，而二级索引只是存储InnoDB的索引的值，便于回查

主键索引

- 怎么设置比较好?
  - > 所以为标识列选择数据类型时，应该选择跟表关联表中的对应列一样的类型 整数通常是标识列最好的选择，因为他们很快并且可以使用 AOTO_INCREMENT 高性能mysql

  - 首先主键id连续的话，避免了插入数据时可能会导致的树结构变更（页分裂）以及（页合并）
  - 当然若只是需要key-value场景，只需要一个索引且为唯一索引时，作为主键索引也没关系
  - 若不建主键索引会怎么样，数据怎么放？（数据都是在主键索引上（聚簇索引））
    - mysql会默认给你建个主键索引，①首先看表中有无非空的唯一索引，例如刚刚说的key-value场景，适合作为主键
    - 若没有合适的索引，则mysql建一个6字节的指针，以此建立聚簇索引GEN_CLUSTER_INDEX

```
CREATE TABLE `test_no_index`
(
    `id` int(11),
    `a`  varchar(20),
    `b`  tinyint(3),
    `c`  varchar(4),
    d    tinyint,
    e    varchar(5)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

show indexes from test_no_index;
-- 查不到任何东西

-- 创建唯一索引
DROP TABLE IF EXISTS `test_no_index`;
CREATE TABLE `test_no_index`
(
    `id` int(11) not null ,
    `a`  varchar(20),
    `b`  tinyint(3),
    `c`  varchar(4),
    d    tinyint,
    e    varchar(5),
    unique index(id)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
show indexes from test_no_index;
-- 可以看到建了索引
-- test_no_index        0       id      1       id      A       1                       ""

-- 插入数据
insert into test_no_index values (1,2,3,4,5,6);
-- 查看数据以及主键id，可以看出id和_rowid一致，也被建为主键id
select id,a,b,c,d,e,_rowid from test_no_index;
-- 1    2       3       4       5       6       1
```

怎么看自己有没有用到索引呢

执行计划

- `explain select ....` 对于drds时是展示分库信息
- `explain execute select ....` 对于drds是展示执行计划
  - drds只能看到select的执行计划，对于其他语句，将相关内容改为select进行解读就好

```
explain select id from test_no_index;
-- 查看执行计划
+----+-------------+---------------+------------+-------+---------------+-----+---------+-------
| id | select_type | table         | partitions | type  | possible_keys | key | key_len | ref
+----+-------------+---------------+------------+-------+---------------+-----+---------+-------
| 1  | SIMPLE      | test_no_index | <null>     | index | <null>        | id  | 4       | <null>
+----+-------------+---------------+------------+-------+---------------+-----+---------+-------

explain select id from test_index;
+----+-------------+------------+------------+-------+---------------+---------+---------+------
| id | select_type | table      | partitions | type  | possible_keys | key     | key_len | ref
+----+-------------+------------+------------+-------+---------------+---------+---------+------
| 1  | SIMPLE      | test_index | <null>     | index | <null>        | PRIMARY | 4       | <null
+----+-------------+------------+------------+-------+---------------+---------+---------+------
```

执行计划解读

字段	作用	关注点
id	执行顺序，id值越大， 越先执行，null时表结果集， 用于union等查询语句	
select_type	sql的查询类型（无子查询、 from中查询…）	关注不同部分的执行计划
partitions	分区 （创建表时指定的分表列信息 ）	关注sql对于分库的查询
type	怎么查的（全表、索引、 range、子查询中用ref、 const（主键索引or唯一索引 ）、full_text等	关注使用到的索引是什么类型的， 能不能优化使用的索引类型。
possible_keys	可能用到的索引	
key	使用到的索引， 是上者的子集合	此处关注索引类型
key_len	索引长度	关注索引长度 len=charType+length+1(允许null)+2（变长列）

字段	作用	关注点
ref	表示上述表的连接匹配条件	若用等值等数查询，此处为const。 如果为连接查询， 被驱动表的执行计划显示驱动表的关联字段， 若为表达式or函数，则此处为func
rows	扫描行数	通常情况下,rows越小,效率越高
filtered	结果集占查询数据量的比	
extra	额外信息（using index 使用覆盖索引）（using where 使用where子句过滤结果集）	

> 附索引类型的执行效率： `all<index<range<ref<eq_ref<const<system`

重点关注项

- type 本次查询表链接类型
- key 最终选择的索引
- ken_len 本次查询用于结果过滤的索引实际长度（关注字段类型越短越好）
- rows 扫描行数
- extra 确认有无出现
  - Using filesort

    > MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the WHERE clause.

```
  CREATE TABLE `test_index`
(
    `id` int(11) not null auto_increment,
    `a`  varchar(20),
    `b`  tinyint(3),
    `c`  varchar(4),
    d    tinyint,
    e    varchar(5),
    primary key (id)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
alter  table test_index add index idx(b);
alter table test_index add index idx_c(c);
explain select * from test_index where b=3 order by c ;
```

当查询的条件与order by条件不一致时，先根据索引查出来值，然后再根据另外的值进行排序，也就是using filesort

```
+----+-------------+------------+------------+------+---------------+-----+---------+--
| id | select_type | table      | partitions | type | possible_keys | key | key_len | r
+----+-------------+------------+------------+------+---------------+-----+---------+--
| 1  | SIMPLE      | test_index | <null>     | ref  | idx           | idx | 2       | c
+----+-------------+------------+------------+------+---------------+-----+---------+--
```

解决方法：添加联合索引idx_b_c(b,c)

```
alter table test_index add index idx_b_c(b,c);
+----+-------------+------------+------------+------+---------------+---------+--------
| id | select_type | table      | partitions | type | possible_keys | key     | key_len
+----+-------------+------------+------------+------+---------------+---------+--------
| 1  | SIMPLE      | test_index | <null>     | ref  | idx,idx_b_c   | idx_b_c | 2
+----+-------------+------------+------------+------+---------------+---------+--------
```

- Using temporary 需要创建一个临时表来存储结果，这通常发生在对没有索引的列进行GROUP BY时，或者ORDER BY里的列不都在索引里
- Using index 覆盖索引
- Using where 通常是进行了全表引扫描后再用WHERE子句完成结果过滤
- Impossible WHERE 对Where子句判断的结果总是false而不能选择任何数据，例如where 1=0，无需过多关注
- Select tables optimized away 使用某些聚合函数来访问存在索引的某个字段时，优化器会通过索引直接一次定位到所需要的数据行完成整个查询，例如MIN()\MAX()