MySQL学习笔记（Day015-Day016：索引/B+树/Explain）

MySQL学习

一. 索引

1. 索引的定义

> 索引是对记录按照一个或者多个字段进行排序的一种方式。对表中的某个字段建立索引会创建另一种数据结构，其中保存着字段的值，每个值又指向与它相关的记录。这种索引的数据结构是经过排序的，因而可以对其执行二分查找。且性能较高

2. 二分查找

> **摘自《MySQL技术内幕：InnoDB存储引擎（第2版）》5.2.1 小节**
>
> 二分查找法（binary search）也称为折半查找法，用来查找一组有序的记录数组中的某一记录，其基本思想是：将记录按有序化（递增或递减）排列，在查找过程中采用跳跃式方式查找，即先以有序数列的中点位置作为比较对象，如果要找的元素值小于该中点元素，则将待查序列缩小为左半部分，否则为右半部分。通过一次比较，将查找区间缩小一半。*（O(logN)的时间复杂度）*

```
# 给出一个例子，注意该例子已经是升序排序的，且查找 数字 48
数据：5、 10、 19、 21、 31、 37、 42、 48、 50、 52
下标：0、  1、  2、  3、  4、  5、  6、  7、  8、  9
```

- 步骤一：设 low 为下标最小值 0 ， high 为下标最大值 9；
- 步骤二：通过 low 和 high 得到 mid ，mid=（low + high）/ 2，初始时 mid 为下标 4 *(也可以=5，看具体算法实现)*；
- 步骤三 ： mid=4 对应的数据值是31，31 < 48（我们要找的数字）；
- 步骤四：通过二分查找的思路，将 low 设置为31对应的下标 4 ， high 保持不变为 9 ，此时 mid 为 6 ；
- 步骤五 ： mid=6 对应的数据值是42，42 < 48（我们要找的数字）；
- 步骤六：通过二分查找的思路，将 low 设置为42对应的下标 6 ， high 保持不变为 9 ，此时 mid 为 7 ；
- 步骤七 ： mid=7 对应的数据值是48，48 == 48（我们要找的数字），查找结束；

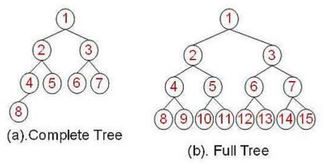
通过3次 二分查找 就找到了我们所要的数字，而顺序查找需8次。

二. 二叉树(Binary Tree)

> 二叉树 wiki

1. 二叉树的定义

- 每个节点至多只有二棵子树；
- 二叉树的子树有左右之分，次序不能颠倒；
- 一棵深度为k，且有 2^k-1 个节点，称为满二叉树(Full Tree)；
- 一棵深度为k，且root到k-1层的节点树都达到最大，第k层的所有节点都 连续集中 在最左边，此时为完全二叉树（Complete Tree）



(a).Complete Tree

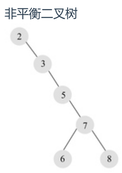
(b). Full Tree

2. 平衡二叉树（AVL-树）

> 平衡二叉树 wiki

- 左子树和右子树都是平衡二叉树；
- 左子树和右子树的高度差绝对值不超过1；
  - 平衡二叉树

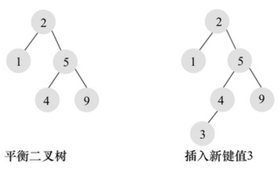
  - 非平衡二叉树



- **平衡二叉树的遍历**
  以上面平衡二叉树的图例为样本，进行遍历：
  - 前序 ： 6 , 3, 2, 5, 7, 8（ROOT节点在开头，中 -左-右 顺序）
  - 中序 ：2, 3, 5, 6 , 7, 8（中序遍历即为升序，左- 中 -右 顺序）
  - 后序 ：2, 5, 3, 8, 7, 6（ROOT节点在结尾，左-右- 中 顺序）

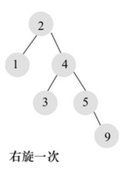
> 1：可以通过 前序 和 中序 或者是 后序 和 中序 来推导出一个棵树
> 2： 前序 或者 后序 用来得到ROOT节点， 中序 可以区分左右子树

- **平衡二叉树的旋转**

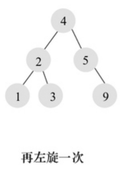


平衡二叉树

插入新键值3



右旋一次



再左旋一次

> 需要通过旋转（左旋，右旋）来维护平衡二叉树的平衡，在添加和删除的时候需要有额外的开销。

三. B树/B+树

> B树 wiki介绍
> B+树 wiki介绍
>
> **注意:B树和B+树开头的 B 不是Binary，而是 Balance**

1. B树的定义

阶为M *(节点上关键字(Keys)的个数)* 的B树的定义：

- 每个节点最多有M个孩子；
- 除了root节点外，每个非叶子(non-leaf)节点至少含有(M/2)个孩子；
- 如果root节点不为空，则root节点至少要有两个孩子节点；
- 一个非叶子(non-leaf)节点如果含有K个孩子，则包含k-1个keys；
- 所有叶子节点都在同一层；
- B树中的非叶子(non-leaf)节点也包含了数据部分；

2. B+树的定义

在B树的基础上，B+树做了如下改进

- 数据只存储在叶子节点上，非叶子节点只保存索引信息；
  - 非叶子节点（索引节点）存储的只是一个Flag，不保存实际数据记录；
  - 索引节点指示该节点的左子树比这个Flag小，而右子树大于等于这个Flag
- 叶子节点本身按照数据的升序排序进行链接(串联起来)；
  - 叶子节点中的数据在 物理存储上是无序 的，仅仅是在 逻辑上有序 （通过指针串在一起）；

3. B+树的作用

- 在块设备上，通过B+树可以有效的存储数据；
- 所有记录都存储在叶子节点上，非叶子(non-leaf)存储索引(keys)信息；

- B+树含有非常高的扇出（fanout），通常超过100，在查找一个记录时，可以有效的减少IO操作；

3. B+树的操作

- **B+树的插入**
  B+树的插入必须保证插入后叶子节点中的记录依然排序。
  - 插入操作步骤*（引用自姜老师的书《MySQL技术内幕：InnoDB存储引擎（第2版）》第5.3.1小节）*

Leaf Page 满	Index Page 满	操作
No	No	直接将记录插入到叶子节点
Yes	No	1）拆分 Leaf Page 2）将中间的节点放入到 Index Page 中 3）小于中间节点的记录放左边 4）大于或等于中间节点的记录放右边
Yes	Yes	1）拆分 Leaf Page 2）小于中间节点的记录放左边 3）大于或等于中间节点的记录放右边 4）拆分 Index Page 5）小于中间节点的记录放左边 6）大于中间节点的记录放右边 7）中间节点放入上一层 Index Page

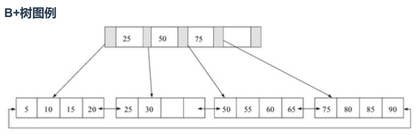
> B+树总是会保持平衡。但是为了保持平衡对于新插入的键值可能需要做大量的拆分页（split）操作；部分情况下可以通过B+树的旋转来替代拆分页操作，进而达到平衡效果。

- **B+树的删除**
  B+树使用填充因子（fill factor）来控制树的删除变化，50%是填充因子可设的最小值。B+树的删除操作同样必须保证删除后叶子节点中的记录依然排序。与插入不同的是，删除根据填充因子的变化来衡量。
  - 删除操作步骤*（引用自姜老师的书《MySQL技术内幕：InnoDB存储引擎（第2版）》第5.3.2小节）*

叶子节点小于填充因子	中间节点小于填充因子	操作
No	No	直接将记录从叶子节点删除，如果该节点还是 Index Page 的节点，用该节点的右节点代替
Yes	No	合并叶子节点和它的兄弟节点，同时更新 Index Page
Yes	Yes	1）合并叶子节点和它的兄弟节点 2）更新 Index Page 3）合并 Index Page 和它的兄弟节点

3. B+树的扇出(fan out)

- **B+树图例**



  - 该 B+ 树高度为 2
  - 每叶子页（LeafPage）4条记录
  - 扇出数为5
  - 叶子节点(LeafPage)由小到大（有序）串联在一起

> 扇出 是每个索引节点(Non-LeafPage)指向每个叶子节点(LeafPage)的指针
> 扇出数 = 索引节点(Non-LeafPage)可存储的最大关键字个数 + 1
> 图例中的索引节点(Non-LeafPage)最大可以存放4个关键字，但实际使用了3个；

4. B+树存储数据举例

假设B+树中页的大小是16K，每行记录是200Byte大小，求出树的高度为1，2，3，4时，分别可以存储多少条记录。

- 查看数据表中每行记录的平均大小

```
mysql> show table status like "%employees%"\G
*************************** 1. row ***************************
           Name: employees
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 298124
 Avg_row_length: 51    -- 平均长度为51个字节
    Data_length: 15245312
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2015-12-02 21:32:02
    Update_time: NULL
     Check_time: NULL
      Collation: utf8mb4_general_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

- **高度为1**
  - 16K/200B 约等于 80 个记录（数据结构元信息如指针等忽略不计）
- **高度为2**
  非叶子节点中存放的仅仅是一个索引信息，包含了 `Key` 和 `Point` 指针； `Point` 指针在MySQL中固定为 `6Byte` 。而 `Key` 我们这里假设为 `8Byte` ，则单个索引信息即为14个字节， `KeySize = 14Byte`
  - 高度为2，即有一个索引节点（索引页），和N个叶子节点
  - 一个索引节点可以存放 16K / KeySize = 16K / 14B = 1142个索引信息，即有（1142 + 1）个扇出，以及有（1142 + 1）个叶子节点（数据页）*（可以简化为1000）*
  - 数据记录数 =（16K / KeySize + 1）x (16K / 200B) 约等于 80W 个记录
- **高度为3**
  高度为3的B+树，即ROOT节点有1000个扇出，每个扇出又有1000个扇出指向叶子节点。每个节点是80个记录，所以一共有 8000W个记录
- **高度为4**
  同高度3一样，高度为4时的记录书为（8000 x 1000）W

> 上述的8000W等数据只是一个理论值。线上实际使用单个页的记录数字要乘以70%，即第二层需要70% x 70% ，依次类推。
>
> 因此在数据库中，B+树的高度一般都在2～4层，这也就是说查找某一键值的行记录时最多只需要2到4次IO，2～4次的IO意味着查询时间只需0.02～0.04秒（假设IOPS=100，当前SSD可以达到50000IOPS）。

从5.7开始，页的预留大小可以设置了，以减少split操作的概率（空间换时间）

四. MySQL索引

1. MySQL 创建索引

> ALTER TABLE 方式
> CREATE INDEX 方式

```
--
-- ALTER TABLE
--
mysql> create table test_index_1(a int, b int , c int);
Query OK, 0 rows affected (0.20 sec)

mysql> show create table test_index_1\G
*************************** 1. row ***************************
       Table: test_index_1
Create Table: CREATE TABLE `test_index_1` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `c` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)

mysql> insert into test_index_1 values
    -> (1,10,100),(2,20,200),
    -> (3,30,300),(4,40,400);
Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from test_index_1;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 |   10 |  100 |
|    2 |   20 |  200 |
|    3 |   30 |  300 |
|    4 |   40 |  400 |
+------+------+------+
4 rows in set (0.00 sec)

mysql> explain select * from test_index_1 where a=3\G  -- 看执行计划，使用的是扫描整张表的方式
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4
     filtered: 25.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

-- 给字段a 增加索引
mysql> alter table test_index_1 add index idx_a (a);  -- 给字段a添加索引。索引名为idx_a
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select * from test_index_1 where a=3\G  -- 看执行计划，使用的key为idx_a，走了索引
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_1
   partitions: NULL
         type: ref
possible_keys: idx_a
          key: idx_a
      key_len: 5
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)

-- 使用create index
mysql> explain select * from test_index_1 where b=30\G  -- 同样b字段也没有索引
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4
     filtered: 25.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

-- 给b字段增加索引
mysql> create index idx_b on test_index_1 (b);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select * from test_index_1 where b=30\G  -- 查看执行计划，使用的key为idx_b，走了索引
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_1
   partitions: NULL
         type: ref
possible_keys: idx_b
          key: idx_b
      key_len: 5
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

2. MySQL 查看索引

```
--
-- 方式一
--
mysql> desc orders;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| o_orderkey      | int(11)     | NO   | PRI | NULL    |       | -- 索引
| o_custkey       | int(11)     | YES  | MUL | NULL    |       | -- 索引
| o_orderstatus   | char(1)     | YES  |     | NULL    |       |
| o_totalprice    | double      | YES  |     | NULL    |       |
| o_orderDATE     | date        | YES  | MUL | NULL    |       | -- 索引
| o_orderpriority | char(15)    | YES  |     | NULL    |       |
| o_clerk         | char(15)    | YES  |     | NULL    |       |
| o_shippriority  | int(11)     | YES  |     | NULL    |       |
| o_comment       | varchar(79) | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
9 rows in set (0.00 sec)

--
-- 方式二
--
mysql> show create table orders\G
*************************** 1. row ***************************
       Table: orders
Create Table: CREATE TABLE `orders` (
  `o_orderkey` int(11) NOT NULL,
  `o_custkey` int(11) DEFAULT NULL,
  `o_orderstatus` char(1) DEFAULT NULL,
  `o_totalprice` double DEFAULT NULL,
  `o_orderDATE` date DEFAULT NULL,
  `o_orderpriority` char(15) DEFAULT NULL,
  `o_clerk` char(15) DEFAULT NULL,
  `o_shippriority` int(11) DEFAULT NULL,
  `o_comment` varchar(79) DEFAULT NULL,
  PRIMARY KEY (`o_orderkey`),  -- 索引
  KEY `i_o_orderdate` (`o_orderDATE`), -- 索引
  KEY `i_o_custkey` (`o_custkey`),  -- 索引
  CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`o_custkey`) REFERENCES `customer` (`c_custkey`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)


--
-- 方式三
--
mysql> show index from orders\G
*************************** 1. row ***************************
        Table: orders
   Non_unique: 0        -- 表示唯一的
     Key_name: PRIMARY  -- key的name是primary
 Seq_in_index: 1
  Column_name: o_orderkey
    Collation: A
  Cardinality: 1306748  -- 基数，这个列上不同值的记录数
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE    -- 索引类型是BTree
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: orders
   Non_unique: 1       -- Non_unique为True，表示不唯一
     Key_name: i_o_orderdate
 Seq_in_index: 1
  Column_name: o_orderDATE
    Collation: A
  Cardinality: 2405
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: orders
   Non_unique: 1
     Key_name: i_o_custkey
 Seq_in_index: 1
  Column_name: o_custkey
    Collation: A
  Cardinality: 95217
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)

mysql>  select count(*) from orders;
+----------+
| count(*) |
+----------+
|  1500000 |  -- orders中有150W条记录，和Cardinality 是不一致的
+----------+
1 row in set (0.25 sec)
```

3. Cardinality（基数）

`Cardinality` 表示该索引列上有多少 `不同的记录` ，这个是一个 `预估的值` ，是采样得到的（由InnoDB触发，采样20个页，进行预估），该值 `越大越好` ，即当 `Cardinality / RowNumber 越接近1越好` 。表示该列是 `高选择性的` 。

- 高选择性：身份证 、手机号码、姓名、订单号等
- 低选择性：性别、年龄等

即该列是否适合创建索引，就看该字段是否具有高选择性

```
mysql>  show create table lineitem\G
*************************** 1. row ***************************
       Table: lineitem
Create Table: CREATE TABLE `lineitem` (
  `l_orderkey` int(11) NOT NULL,
  `l_partkey` int(11) DEFAULT NULL,
  `l_suppkey` int(11) DEFAULT NULL,
  `l_linenumber` int(11) NOT NULL,
  `l_quantity` double DEFAULT NULL,
  `l_extendedprice` double DEFAULT NULL,
  `l_discount` double DEFAULT NULL,
  `l_tax` double DEFAULT NULL,
  `l_returnflag` char(1) DEFAULT NULL,
  `l_linestatus` char(1) DEFAULT NULL,
  `l_shipDATE` date DEFAULT NULL,
  `l_commitDATE` date DEFAULT NULL,
  `l_receiptDATE` date DEFAULT NULL,
  `l_shipinstruct` char(25) DEFAULT NULL,
  `l_shipmode` char(10) DEFAULT NULL,
  `l_comment` varchar(44) DEFAULT NULL,
  PRIMARY KEY (`l_orderkey`,`l_linenumber`),  -- 两个列作为primary
  KEY `i_l_shipdate` (`l_shipDATE`),
  KEY `i_l_suppkey_partkey` (`l_partkey`,`l_suppkey`),
  KEY `i_l_partkey` (`l_partkey`),
  KEY `i_l_suppkey` (`l_suppkey`),
  KEY `i_l_receiptdate` (`l_receiptDATE`),
  KEY `i_l_orderkey` (`l_orderkey`),
  KEY `i_l_orderkey_quantity` (`l_orderkey`,`l_quantity`),
  KEY `i_l_commitdate` (`l_commitDATE`),
  CONSTRAINT `lineitem_ibfk_1` FOREIGN KEY (`l_orderkey`) REFERENCES `orders` (`o_orderkey`),
  CONSTRAINT `lineitem_ibfk_2` FOREIGN KEY (`l_partkey`, `l_suppkey`) REFERENCES `partsupp` (`ps_partkey`, `ps_suppkey`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql>  show index from lineitem\G  -- 省略其他输出，只看PRIMARY
*************************** 1. row ***************************
        Table: lineitem
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1  -- 索引中的顺序，该列的顺序为1
  Column_name: l_orderkey
    Collation: A
  Cardinality: 1416486  -- 约140W
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: lineitem
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 2  -- 索引中的顺序，该列的顺序为2
  Column_name: l_linenumber
    Collation: A
  Cardinality: 5882116  -- 约580W
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

对应当前例子
第一个索引（Seq_in_index = 1）的 `Cardinality` 的值表示 `当前列（l_orderkey）` 的不重复的值，
第二个索引（Seq_in_index = 2）的 `Cardinality` 的值表示 `前两列（l_orderkey）和（l_linenumber）` 不重复的值

```
--
-- SQL-1
--
mysql> select * from lineitem limit 10;
+------------+-----------+-----------+--------------+------------+-----------------+------------+------+--------------+--------------+------------+--------------+-------------------+------------+--------------------------------------+
| l_orderkey | l_partkey | l_suppkey | l_linenumber | l_quantity | l_extendedprice | l_discount | l_tax | l_returnflag | l_linestatus | l_shipDATE | l_commitDATE | l_receiptDATE | l_shipinstruct    | l_shipmode | l_comment                            |
+------------+-----------+-----------+--------------+------------+-----------------+------------+------+--------------+--------------+------------+--------------+-------------------+------------+--------------------------------------+
|          1 |    155190 |      7706 |            1 |         17 |        21168.23 |       0.04 |  0.02 | N            | O            | 1996-03-13 | 1996-02-12   | 1996-03-22    | DELIVER IN PERSON | TRUCK      | blithely regular ideas caj           |
|          1 |     67310 |      7311 |            2 |         36 |        45983.16 |       0.09 |  0.06 | N            | O            | 1996-04-12 | 1996-02-28   | 1996-04-20    | TAKE BACK RETURN  | MAIL       | slyly bold pinto beans detect s      |
|          1 |     63700 |      3701 |            3 |          8 |         13309.6 |        0.1 |  0.02 | N            | O            | 1996-01-29 | 1996-03-05   | 1996-01-31    | TAKE BACK RETURN  | REG AIR    | deposits wake furiously dogged,      |
|          1 |      2132 |      4633 |            4 |         28 |        28955.64 |       0.09 |  0.06 | N            | O            | 1996-04-21 | 1996-03-30   | 1996-05-16    | NONE              | AIR        | even ideas haggle. even, bold reque  |
|          1 |     24027 |      1534 |            5 |         24 |        22824.48 |        0.1 |  0.04 | N            | O            | 1996-03-30 | 1996-03-14   | 1996-04-01    | NONE              | FOB        | carefully final gr                   |
|          1 |     15635 |       638 |            6 |         32 |        49620.16 |       0.07 |  0.02 | N            | O            | 1996-01-30 | 1996-02-07   | 1996-02-03    | DELIVER IN PERSON | MAIL       | furiously regular accounts haggle bl |
|          2 |    106170 |      1191 |            1 |         38 |        44694.46 |          0 |  0.05 | N            | O            | 1997-01-28 | 1997-01-14   | 1997-02-02    | TAKE BACK RETURN  | RAIL       | carefully ironic platelets against t |
|          3 |      4297 |      1798 |            1 |         45 |        54058.05 |       0.06 |     0 | R            | F            | 1994-02-02 | 1994-01-04   | 1994-02-23    | NONE              | AIR        | blithely s                           |
|          3 |     19036 |      6540 |            2 |         49 |        46796.47 |        0.1 |     0 | R            | F            | 1993-11-09 | 1993-12-20   | 1993-11-24    | TAKE BACK RETURN  | RAIL       | final, regular pinto                 |
|          3 |    128449 |      3474 |            3 |         27 |        39890.88 |       0.06 |  0.07 | A            | F            | 1994-01-16 | 1993-11-22   | 1994-01-23    | DELIVER IN PERSON | SHIP       | carefully silent pinto beans boost fur |
+------------+-----------+-----------+--------------+------------+-----------------+------------+------+--------------+--------------+------------+--------------+-------------------+------------+--------------------------------------+
10 rows in set (0.00 sec)

--
-- SQL-2
--
mysql> select l_orderkey, l_linenumber from lineitem limit 10;
+------------+--------------+
| l_orderkey | l_linenumber |
+------------+--------------+
|     721220 |            2 |
|     842980 |            4 |
|     904677 |            1 |
|     990147 |            1 |
|    1054181 |            1 |
|    1111877 |            3 |
|    1332613 |            1 |
|    1552449 |            2 |
|    2167527 |            3 |
|    2184032 |            5 |
+------------+--------------+
10 rows in set (0.00 sec)

--- SQL-1和SQL-2其实都是在没有排序的情况下，取出前10条数据。但是结果不一样

--
-- SQL-3
--
mysql> select l_orderkey, l_linenumber from lineitem order by l_orderkey limit 10;  -- 和上面的sql相比，多了一个order by的操作
+------------+--------------+
| l_orderkey | l_linenumber |
+------------+--------------+
|          1 |            1 | -----
|          1 |            2 | -- 看orderkey为1，对应的linenumber有6条
|          1 |            3 | -- 这就是orderkey的Cardinality仅为140W
|          1 |            4 | -- 而(orderkey + linenumber)就有580W
|          1 |            5 |
|          1 |            6 | -----
|          2 |            1 |
|          3 |            1 |
|          3 |            2 |
|          3 |            3 |
+------------+--------------+
10 rows in set (0.01 sec)

--- SQL-3 和SQL-2 不同的原因是 他们走了不同的索引
mysql> explain select l_orderkey, l_linenumber from lineitem limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
   partitions: NULL
         type: index
possible_keys: NULL
          key: i_l_shipdate  -- 使用了shipdate进行了索引
      key_len: 4
          ref: NULL
         rows: 5882306
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.00 sec)


mysql> explain select l_orderkey, l_linenumber from lineitem order by l_orderkey limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
   partitions: NULL
         type: index
possible_keys: NULL
          key: i_l_orderkey  -- 使用了orderkey进行了查询
      key_len: 4
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from lineitem limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
   partitions: NULL
         type: ALL   -- SQL-1进行了全表扫描
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 5882306
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)

-- 所以，不使用order by取出的结果，可以理解为不是根据主键排序的结果。
```

> innodb_on_state = off
> 在MySQL5.5之前，执行 show create table 操作会触发采样，而5.5之后将该参数off后，需要主动执行 analyze table 才会去采样。采样不会锁表或者锁记录。

4. 复合索引

```
mysql>  show create table lineitem\G
*************************** 1. row ***************************
       Table: lineitem
Create Table: CREATE TABLE `lineitem` (
  `l_orderkey` int(11) NOT NULL,
  `l_partkey` int(11) DEFAULT NULL,
  `l_suppkey` int(11) DEFAULT NULL,
  `l_linenumber` int(11) NOT NULL,
  `l_quantity` double DEFAULT NULL,
  `l_extendedprice` double DEFAULT NULL,
  `l_discount` double DEFAULT NULL,
  `l_tax` double DEFAULT NULL,
  `l_returnflag` char(1) DEFAULT NULL,
  `l_linestatus` char(1) DEFAULT NULL,
  `l_shipDATE` date DEFAULT NULL,
  `l_commitDATE` date DEFAULT NULL,
  `l_receiptDATE` date DEFAULT NULL,
  `l_shipinstruct` char(25) DEFAULT NULL,
  `l_shipmode` char(10) DEFAULT NULL,
  `l_comment` varchar(44) DEFAULT NULL,
  PRIMARY KEY (`l_orderkey`,`l_linenumber`),  -- 两个列作为primary，这个就是复合索引
  KEY `i_l_shipdate` (`l_shipDATE`),
  KEY `i_l_suppkey_partkey` (`l_partkey`,`l_suppkey`),
  KEY `i_l_partkey` (`l_partkey`),
  KEY `i_l_suppkey` (`l_suppkey`),
  KEY `i_l_receiptdate` (`l_receiptDATE`),
  KEY `i_l_orderkey` (`l_orderkey`),
  KEY `i_l_orderkey_quantity` (`l_orderkey`,`l_quantity`),
  KEY `i_l_commitdate` (`l_commitDATE`),
  CONSTRAINT `lineitem_ibfk_1` FOREIGN KEY (`l_orderkey`) REFERENCES `orders` (`o_orderkey`),
  CONSTRAINT `lineitem_ibfk_2` FOREIGN KEY (`l_partkey`, `l_suppkey`) REFERENCES `partsupp` (`ps_partkey`, `ps_suppkey`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

--
-- 复合索引举例
--
mysql> create table test_index_2(a int, b int , c int);
Query OK, 0 rows affected (0.15 sec)

mysql> alter table test_index_2 add index  idx_mul_ab (a, b);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> insert into test_index_2 values
    -> (1,1,10),
    -> (1,2,9),
    -> (2,1,8),
    -> (2,4,15),
    -> (3,1,6),
    -> (3,2,17);
Query OK, 6 rows affected (0.04 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> select * from test_index_2 where a = 1;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 |    1 |   10 |
|    1 |    2 |    9 |
+------+------+------+
2 rows in set (0.00 sec)

mysql> explain select * from test_index_2 where a = 1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ref
possible_keys: idx_mul_ab
          key: idx_mul_ab   -- 走了索引
      key_len: 5
          ref: const
         rows: 2
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)

mysql> select * from test_index_2 where a = 1 and b = 2;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 |    2 |    9 |
+------+------+------+
1 row in set (0.00 sec)

mysql> explain select * from test_index_2 where a = 1 and b = 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ref   -- 此时也是走了索引
possible_keys: idx_mul_ab
          key: idx_mul_ab
      key_len: 10
          ref: const,const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from test_index_2 where b = 2\G  -- 只查询b
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ALL  -- 没有使用索引
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
     filtered: 16.67
        Extra: Using where
1 row in set, 1 warning (0.00 sec)


mysql> explain select * from test_index_2 where a=1 or b = 2\G  -- 使用or，要求结果集是并集
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ALL  -- 没有使用索引，因为b没有索引，所以b是走全表扫描的，既然走扫描，a的值也可以一起过滤
                    -- 就没有必要在去查一次 a 的索引了
possible_keys: idx_mul_ab
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
     filtered: 30.56
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

--
-- 特别的例子
--
---- 还是只使用b列去做范围查询，发现是走索引了
---- 注意查询的是 count(*)
mysql> explain select count(*) from test_index_2 where  b >1 and b < 3\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: index  -- 走了索引
possible_keys: NULL
          key: idx_mul_ab
      key_len: 10
          ref: NULL
         rows: 6
     filtered: 16.67
        Extra: Using where; Using index  -- 覆盖索引
1 row in set, 1 warning (0.00 sec)

-- 因为要求的是count(*)，要求所有的记录的和，
-- 那索引a是包含了全部的记录的，即扫描（a,b)的索引也是可以得到count(*)的

mysql> explain select * from test_index_2 where  b >1 and b < 3\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ALL  -- 查询 * 就没法使用（a，b）索引了，需要全表扫描b的值。
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
     filtered: 16.67
        Extra: Using where
1 row in set, 1 warning (0.00 sec)


mysql> explain select * from test_index_2 where  a = 1 and c = 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ref  -- 也是走索引的，先用走a索引得到结果集，再用c=10去过滤
possible_keys: idx_mul_ab
          key: idx_mul_ab
      key_len: 5
          ref: const
         rows: 2
     filtered: 16.67
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from test_index_2 where  b = 2 and c = 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
```

```
   partitions: NULL
         type: ALL  -- 而b和c是不行的，(b，c)不是有序的
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
     filtered: 16.67
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

五. information_schema (一)

```
mysql> use information_schema;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

mysql> show tables;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| OPTIMIZER_TRACE                       |
| PARAMETERS                            |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| PROFILING                             |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
| SESSION_VARIABLES                     |
| STATISTICS                            |
| TABLES                                |
| TABLESPACES                           |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
| INNODB_LOCKS                          |
| INNODB_TRX                            |
| INNODB_SYS_DATAFILES                  |
| INNODB_FT_CONFIG                      |
| INNODB_SYS_VIRTUAL                    |
| INNODB_CMP                            |
| INNODB_FT_BEING_DELETED               |
| INNODB_CMP_RESET                      |
| INNODB_CMP_PER_INDEX                  |
| INNODB_CMPMEM_RESET                   |
| INNODB_FT_DELETED                     |
| INNODB_BUFFER_PAGE_LRU                |
| INNODB_LOCK_WAITS                     |
| INNODB_TEMP_TABLE_INFO                |
| INNODB_SYS_INDEXES                    |
| INNODB_SYS_TABLES                     |
| INNODB_SYS_FIELDS                     |
| INNODB_CMP_PER_INDEX_RESET            |
| INNODB_BUFFER_PAGE                    |
| INNODB_FT_DEFAULT_STOPWORD            |
| INNODB_FT_INDEX_TABLE                 |
| INNODB_FT_INDEX_CACHE                 |
| INNODB_SYS_TABLESPACES                |
| INNODB_METRICS                        |
| INNODB_SYS_FOREIGN_COLS               |
| INNODB_CMPMEM                         |
| INNODB_BUFFER_POOL_STATS              |
| INNODB_SYS_COLUMNS                    |
| INNODB_SYS_FOREIGN                    |
| INNODB_SYS_TABLESTATS                 |
+---------------------------------------+
61 rows in set (0.00 sec)

-- information_schema 数据库相当于一个数据字典。保存了表的元信息。

mysql> select * from key_column_usage limit  3\G  -- 显示了哪个索引使用了哪个列
*************************** 1. row ***************************
           CONSTRAINT_CATALOG: def
            CONSTRAINT_SCHEMA: burn_test
              CONSTRAINT_NAME: PRIMARY
                TABLE_CATALOG: def
                 TABLE_SCHEMA: burn_test
                   TABLE_NAME: Orders -- 表名
                  COLUMN_NAME: order_id  -- 索引的名称
             ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
      REFERENCED_TABLE_SCHEMA: NULL
        REFERENCED_TABLE_NAME: NULL
       REFERENCED_COLUMN_NAME: NULL
*************************** 2. row ***************************
           CONSTRAINT_CATALOG: def
            CONSTRAINT_SCHEMA: burn_test
              CONSTRAINT_NAME: product_name
                TABLE_CATALOG: def
                 TABLE_SCHEMA: burn_test
                   TABLE_NAME: Orders_MV
                  COLUMN_NAME: product_name
             ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
      REFERENCED_TABLE_SCHEMA: NULL
        REFERENCED_TABLE_NAME: NULL
       REFERENCED_COLUMN_NAME: NULL
*************************** 3. row ***************************
           CONSTRAINT_CATALOG: def
            CONSTRAINT_SCHEMA: burn_test
              CONSTRAINT_NAME: child_ibfk_1
                TABLE_CATALOG: def
                 TABLE_SCHEMA: burn_test
                   TABLE_NAME: child
                  COLUMN_NAME: parent_id
             ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: 1
      REFERENCED_TABLE_SCHEMA: burn_test
        REFERENCED_TABLE_NAME: parent
       REFERENCED_COLUMN_NAME: id
3 rows in set (0.04 sec)

-- 作业:
-- 1:Cardinality 在那张表里面
-- 2:线上的索引是否可以优化
```

六. EXPLAIN （一）

> EXPLAIN 官方文档

- explain是解释SQL语句的执行计划，即显示该SQL语句怎么执行的
  - 使用 `explain` 的时候，也可以使用 `desc`
- 5.6 版本支持DML语句进行explain解释
- 5.6 版本开始支持 `JSON格式` 的输出

  注意：EXPLAIN查看的是执行计划，做SQL解析，不会去真的执行；且到5.7以后子查询也不会去执行。

- **参数extended**

```
mysql> explain extended  select * from test_index_2 where  b >1 and b < 3\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: test_index_2
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
     filtered: 16.67
        Extra: Using where
1 row in set, 2 warnings (0.00 sec)  有 warnings，这里相当于提供一个信息返回

mysql> show warnings\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1681
Message: 'EXTENDED' is deprecated and will be removed in a future release. -- 即将被弃用
*************************** 2. row ***************************  -- 显示真正的执行语句
  Level: Note
   Code: 1003
Message: /* select#1 */ select `burn_test`.`test_index_2`.`a` AS `a`,`burn_test`.`test_index_2`.`b` AS `b`,`burn_test`.`test_index_2`.`c` AS `c` from `burn_test`.`test_index_2` where ((`burn_test`.`test_index_2`.`b` > 1) and (`burn_test`.`test_index_2`.`b` < 3))
2 rows in set (0.00 sec)
```

- **参数FORMAT**
  - 使用 `FORMART=JSON` 不仅仅是为了格式化输出效果，还有其他有用的显示信息。
  - 且当5.6版本后，使用 `MySQL Workbench` ，可以使用 `visual Explain` 方式显示详细的图示信息。

```
mysql> explain format=json  select * from test_index_2 where  b >1 and b < 3\G
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "2.20"  -- 总成本
    },
    "table": {
      "table_name": "test_index_2",
      "access_type": "ALL",
      "rows_examined_per_scan": 6,
      "rows_produced_per_join": 1,
      "filtered": "16.67",
      "cost_info": {
        "read_cost": "2.00",
        "eval_cost": "0.20",
        "prefix_cost": "2.20",
        "data_read_per_join": "16"
      },
      "used_columns": [
        "a",
        "b",
        "c"
      ],
      "attached_condition": "((`burn_test`.`test_index_2`.`b` > 1) and (`burn_test`.`test_index_2`.`b` < 3))"
    }
  }
}
1 row in set, 1 warning (0.00 sec)
```