索引

索引的目的

提高数据查询的效率

索引的数据模型

哈希表

- 特点：key-value的形式存储，用哈希函数计算key的数组位置，在这个位置上存储值
- 哈希冲突：拉出一个链表，存储冲突的数据
- 缺点：不适合范围查询，因为数据不是有序的
- 适合场景：只有等值查询的场景
- 时间复杂度：查询O(1)

有序数组

- 特点：有序的数组
- 缺点：数据插入效率低，如果在数组中间位置插入数据，需要移动后面的记录
- 适合场景：适用于静态存储引擎，不再修改的数据
- 时间复杂度：查询O(logn)

搜索树

- 特点：父节点左子树节点的值小于父节点的值，右子树节点的值大于父节点的值
- 时间复杂度：查询O(logn)，插入O(logn)
- 数据库存储不使用二叉树：因为等量的数据二叉树树高会很高，磁盘访问次数会变多；使用N叉树解决

InnoDB中的索引

B+树

每一个索引对应一颗B+树，所有的数据都是存储在B+树中

> B+树能够很好的配合磁盘读写特性，减少单次查询的磁盘访问次数。

主键索引

叶子节点存整行的数据，又被称为**聚簇索引**

非主键索引

叶子节点存的是主键的值，也被称为**普通索引，二级索引，辅助索引**

主键索引与非主键索引区别

如果非主键索引中的数据不满足查询条件，需要根据主键值去主键索引中查询其他字段数据；这个过程称为**回表**

页分裂

B+树需要维护索引的有序性，插入新数据的时候需要保持索引有序。
当一个数据页已经满了，此时要在这个数据页中插入一条数据；就需要申请一个新的数据页，然后把部分数据挪到新的数据页当中，这个过程称为**页分裂**

自增ID作索引的好处

- 性能方面：数据有序，每次插入都是追加操作，不会造成页分裂
- 空间方面：主键长度短，普通索引的叶子节点就越小，占用的空间也就越小

联合索引技巧

覆盖索引

如果查询条件使用的是联合索引，查询结果是联合索引字段或是主键，不需要回表操作，直接返回结果。
一次查询中，联合索引覆盖了查询所需的字段，称其为“覆盖索引”。

优点：避免了回表操作，减少了磁盘IO

最左前缀规则

利用索引的最左前缀，来定位记录。

联合索引的最左N个字段，也可以是字符串索引的最左M个字符

查询数据的时候，遵循最左匹配规则，根据联合索引字段顺序，来调整查询条件的顺序。

优点：多字段查询，通过调整字段顺序，充分利用联合索引，减少索引的维护。

索引下推

索引下推（Index Condition Pushdown），简称 ICP
MySQL5.6 引入的索引下推优化：在“仅能利用最左前缀索引的场景”下（而不是能利用全部联合索引）对于最左前缀索引中的其他联合索引字段加以利用，在遍历索引时，就用这些字段进行过滤。过滤会减少索引查出来的主键数，从而减少回表次数，提升整体的性能。
可以在最左前缀索引遍历过程中，对索引中包含的字段先进行判断，直接过滤掉不满足条件的记录，减少回表次数

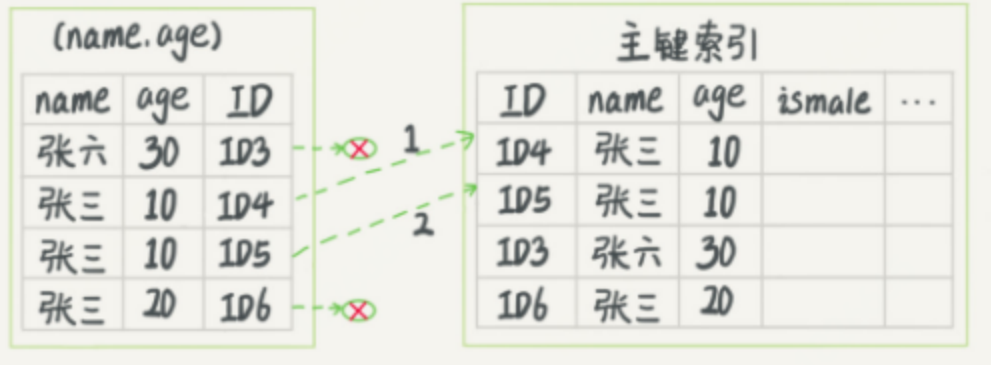
Explain查看执行计划的时候，Extra字段会出现 Using index condition，表示用到了最左前缀匹配，且会利用到索引下推。

举例：

```
mysql> select * from tuser where name like '张%' and age=10 and ismale=1;
```

在5.6之前，根据最左匹配原则，只能用name查询数据，找到ID=3的数据，然后开始挨个遍历去回表，需要回表4次

在5.6引入了索引下推优化，在索引内部就判断了age是否等于10，对应不等于10的记录，直接判断跳过，只需回表2次



普通索引与唯一索引区别

- 从数据查询角度：普通索引需要找到所有符合条件的数据，才会停止搜索；唯一索引找到一条符合条件的数据，就会停止搜索。但这种区别对性能损耗来说影响不大，因为数据加载以页为单位，大多数查询都在一个数据页内完成
- 从数据更新角度：
  - 如果更新的数据已经在内存中，那么两者没什么区别，都在内存中直接更新数据
  - 如果更新的数据不在内存中
    - 普通索引，会将数据的更新操作记录在change buffer中
    - 唯一索引，需要校验数据唯一性，所以会把数据页先加载到内存里去，比普通索引多了一次磁盘IO

Change Buffer

概念

记录数据变更操作的一块缓冲区，包含（insert，update，delete）变更

作用

- 将更新操作先记录在 change buffer ，减少读磁盘，提高语句执行的效率（减少的是对二级普通索引页的读磁盘操作）
- 数据读入内存需要占用 buffer pool，使用这种change buffer可以减少占用内存，提高内存利用率。（change buffer虽然还是要占用内存，但记录的是更新操作，相比数据页16kb来说，占用的内存还是很少的）

原理

- 当需要更新一个数据页时，如果数据页在内存中，则直接更新；如果不在内存中，在不影响数据一致性的前提下，InnoDB会将数据更新操作缓存到 change buffer中
- 什么时候将更新操作应用到数据页？
  - 后台线程定期merge
  - 访问这个数据页触发merge

使用场景

只有在普通索引中适用，适合**写多读少**的场景，数据写完后不会立马被访问到。例如：日志类、账单类系统。

如果一个业务场景写入后马上会做查询，如果将更新记录在change buffer，但之后由于马上要访问数据页，触发merge过程，这样随机访问IO的次数并不会减少，反而增加了chang buffer的维护代价。

实战

- `innodb_change_buffer_max_size`：设置change buffer内存大小，例如：50，表示占用buffer pool 50%；如果配置为0，表示关闭 chagng buffer