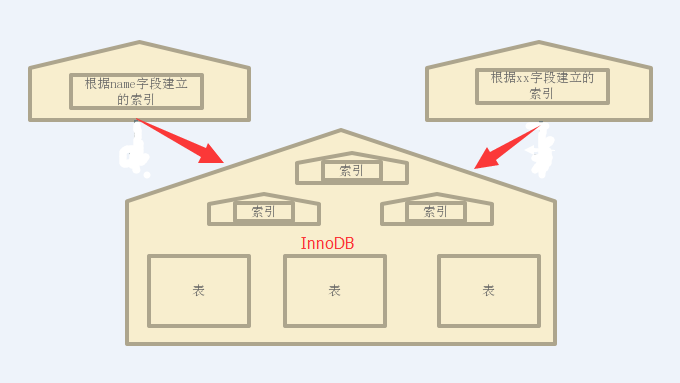
**InnoDb 中索引的建立和使用**

# 聚集索引

按照每张表的主键构造一颗B+树，同时叶子节点中存放的即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，聚集索引的这个特性也决定了索引组织表中数据也是索引的一部分。同B+树数据结构一样，么个数据页都通过一个双向链表来进行链接。

# 辅助索引

叶子节点并不包含行记录的全部数据。叶子结点除了包含键值以外，每个叶子结点中的所有行中还包含了一个书签。该书签用来告诉InnoDB存储引擎哪里可以找到与索引相对应的行数据。由于InnoDB存储引擎表是索引组织表，因此InnoDB存储引擎的辅助索引的书签就是相应行数据的聚集索引键。



InnoDB表也是一张索引表

# 建立索引

1. create index idx\_name on tb\_name (col\_name);
2. alter table tb\_name add index idx\_name (col\_name);

# 删除索引

drop index index\_name on tb\_name;

# 索引维护

如果索引所在字段发生了修改、删除、插入等操作，那么索引项就会发生变化，因此如果不能保证索引的有序，那么就不能索引的准确与效率，而索引的排序发生了变化的这个行为，我们称为索引维护

在insert/delete/update操作时，为了维护索引的排序，数据库会自动的完成索引项的维护，索引的排序，这些行为对用户是透明的，感觉不到的

在一个有索引的表中，创建它时，实际上还同时创建了索引排序的表，因此在DML中，插入等操作不再是普通的插入，MySQL将它封装成了一个事务，连着索引项的排序表一起操作

因此，我们应当严格控制表上的索引数量，否则容易影响数据库的性能

总结索引维护如下：

1. 索引维护由数据库自动完成
2. 插入/修改/删除每一个索引行都变成一个内部封装的事务
3. 索引越多，事务越大，代价越高
4. 索引越多，对表的插入和索引字段的修改就越慢

# Online DDL

在建立索引的时候，避免创建临时表，从而提高了效率，使用LOCK来对表添加锁，可选

## NONE

执行索引创建或删除操作时，对目标表不添加任何的锁，即事务仍然可以进行读写操作，不会收到阻塞，可以获得最大的并发度。

## 2.SHARE

和快速建立索引类似，执行索引创建或删除操作时，对目标表加上一个S锁，对于并发的读事务，依然可以执行，但是遇到写事务，就会发生等待操作，如果存储引擎不支持SHARE模式，会返回一个错误信息。

## 3.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方式运行得到的状态类似，但是不需要像COPY方式那样创建一张临时表。

## 4.DEFAULT

DEFAULT模式首先会当前操作是否可以使用NONE模式，若不能，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，也就是说DEFAULT会通过判断事务的最大并发性来判断执行DDL模式。

# MySQL中能够使用索引的典型场景

1.匹配全值（Match the full value）

2.匹配值得范围查询

3.匹配最左前缀

4.仅仅对索引进行查询

5.匹配列前缀

6.能过实现索引匹配部分精确而其他部分进行范围匹配

# 索引失效（存在索引但是不能使用索引的典型场景）

以%开头的like查询不能够使用B-TREE索引数据库中出现隐式转换的时候不会使用索引

复合索引的情况下，假如查询条件不包含索引的最左部分，即不满足最左原则

用or分割开的条件，如果or前的条件中的列有索引，而后面的列中没有索引，那么涉及到的索引都不会被用到。

# 两个简单实用的优化方法

定期分析和检查表：analyze table tablename 找出可能出现的问题

check table tablename 检查一个或多个表是否有错