# SQL

## select

多个条件相与AND连接，相或用OR连接，取反用NOT

字符串可以用like，“%abc%”其中%可以匹配多个字符

字符串用单引号包围

查询结果可以指定别名

select 列1 别名1，列2 别名2 from table where id = 1；

## order by，排序，在where之后

查询结果按照某个字段排序

order by DESC表示逆序

还可以进一步排序

select \* from table where sex=‘m’ order by age DESC，id；

## 分页查询

LIMIT 5 OFFSET 10，最多查询五个，越过前十个。

## 聚合函数

select count(\*) from table;

count(\*) 也是一个列名，可以指定别名

select count(\*) sum from table;

其他常用聚合函数

sum、avg、max、min

## group by

分组

## 多表查询

可以使用表.列区分字段

select table1.id,table2.id from table1,table2;

也可以给表设置别名

select a.id,b.id from table1 a,table2 b;

## 连接查询

inner join

两个表的交集

left join

左表全部+右表匹配部分

right join

右表全部+左表匹配部分

full join

两个表的并集

## SQL关键字编写顺序

select from where group by having order by

## where和having区别：

where对全部数据进行筛选

having只对group by分组后的数据进行筛选

## 关键字的执行顺序

from->找到涉及到的表

where->筛选涉及的数据

group by->将数据按XX分组

聚合函数count/avg/sum->聚合函数计算

having->筛选计算后的数据

select->选择要呈现的数据

order by 按照升降序排列

# MySQL索引详解

## 索引的分类（多角度）：

数据结构角度：

B+树索引：B+树

Hash索引：Hash表

存储结构角度：

聚簇索引：叶子结点为数据

非聚簇索引：叶子结点无数据

逻辑角度：

主键、唯一、普通、复合

## B+树索引：

其底层数据结构是一棵B+树（特点：多叉树，普通结点不含数据，只包含索引字段值，叶子结点间通过指针连接。）

优点：范围查询效率高（通过叶子结点的指针）

## Hash索引：

类似于hash表，将索引字段作为key，记录的指针作为value。

优点：单条记录查询速度极快

缺点：数据乱序、范围查询满、只能查询=条件，复合索引只能全部使用，hash碰撞多时效率低

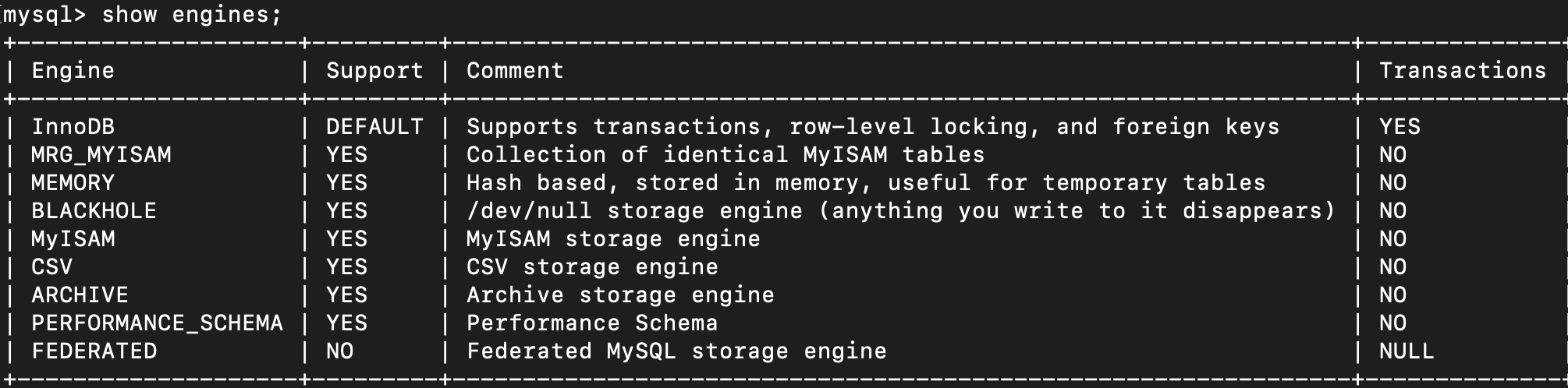
## 聚簇索引：B+树的叶子结点保存数据

## 非聚簇索引：B+树的叶子结点不保存数据

## 查看SQL执行计划

EXPLAIN关键字

## MySQL存储引擎



其中常用的是InnoDB和MyISAM

存储引擎决定一整套的数据库系统：包括存储机制、索引结构、加锁水平、事务执行情况等。

## InnoDB：事务、行锁、外键、全文索引

特点：并发性能高（行锁，锁粒度小）

支持事务

## MyISAM：表锁、无外键、无事务、

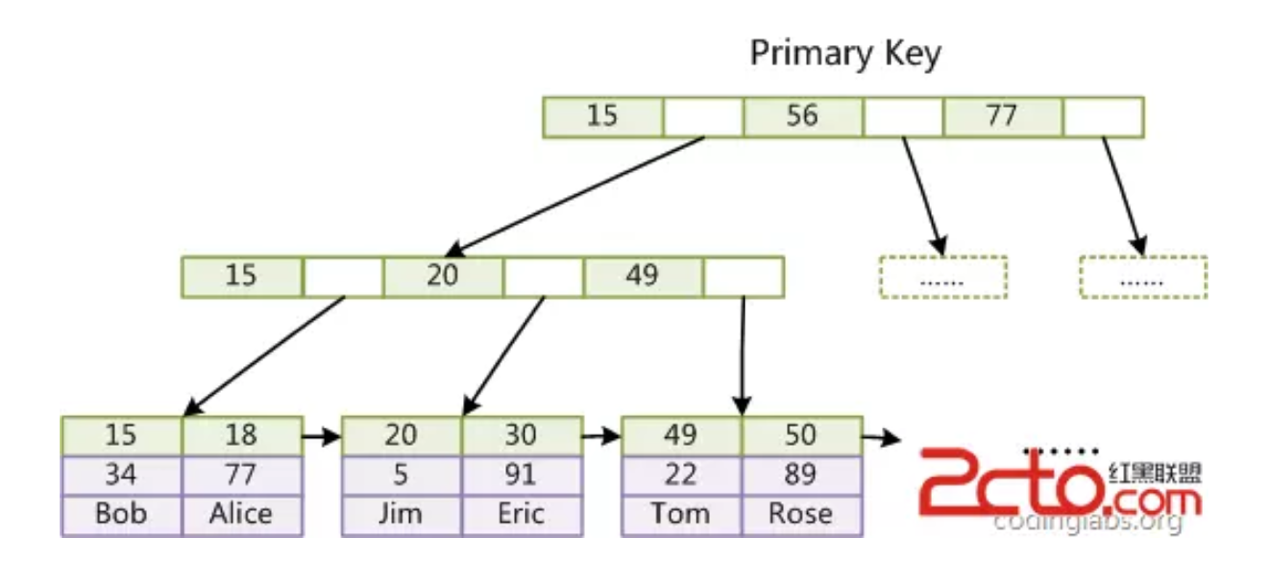
特点：读取更快（索引和数据分开，一次可以读取更多索引）

不支持事务、

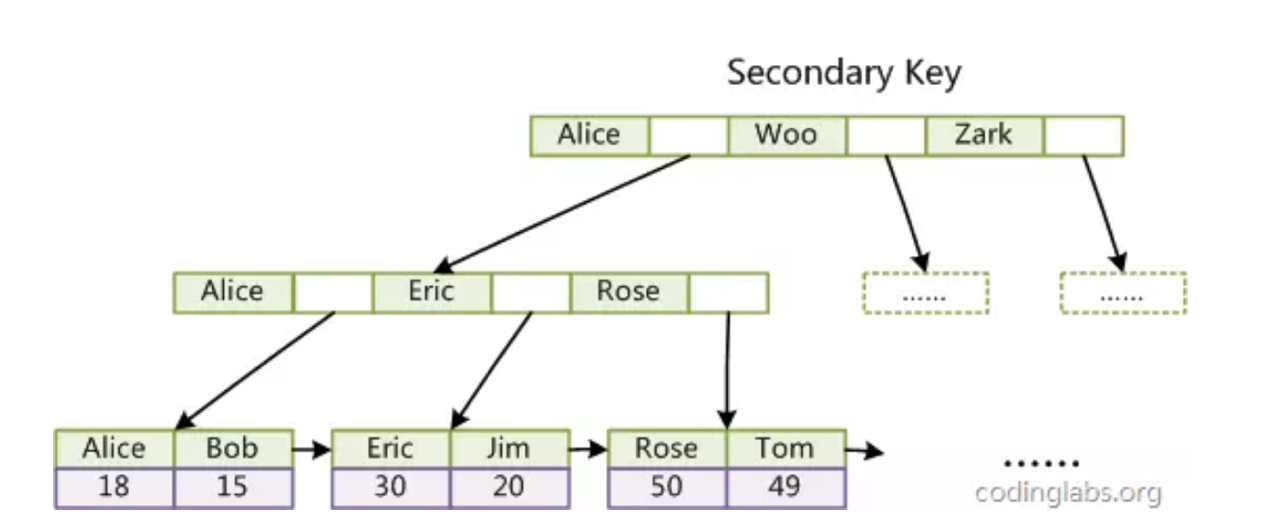
## InnoDB与MyISAM索引实现：

### InnoDB：

聚簇索引：普通结点保存索引关键字，叶子结点保存记录数据

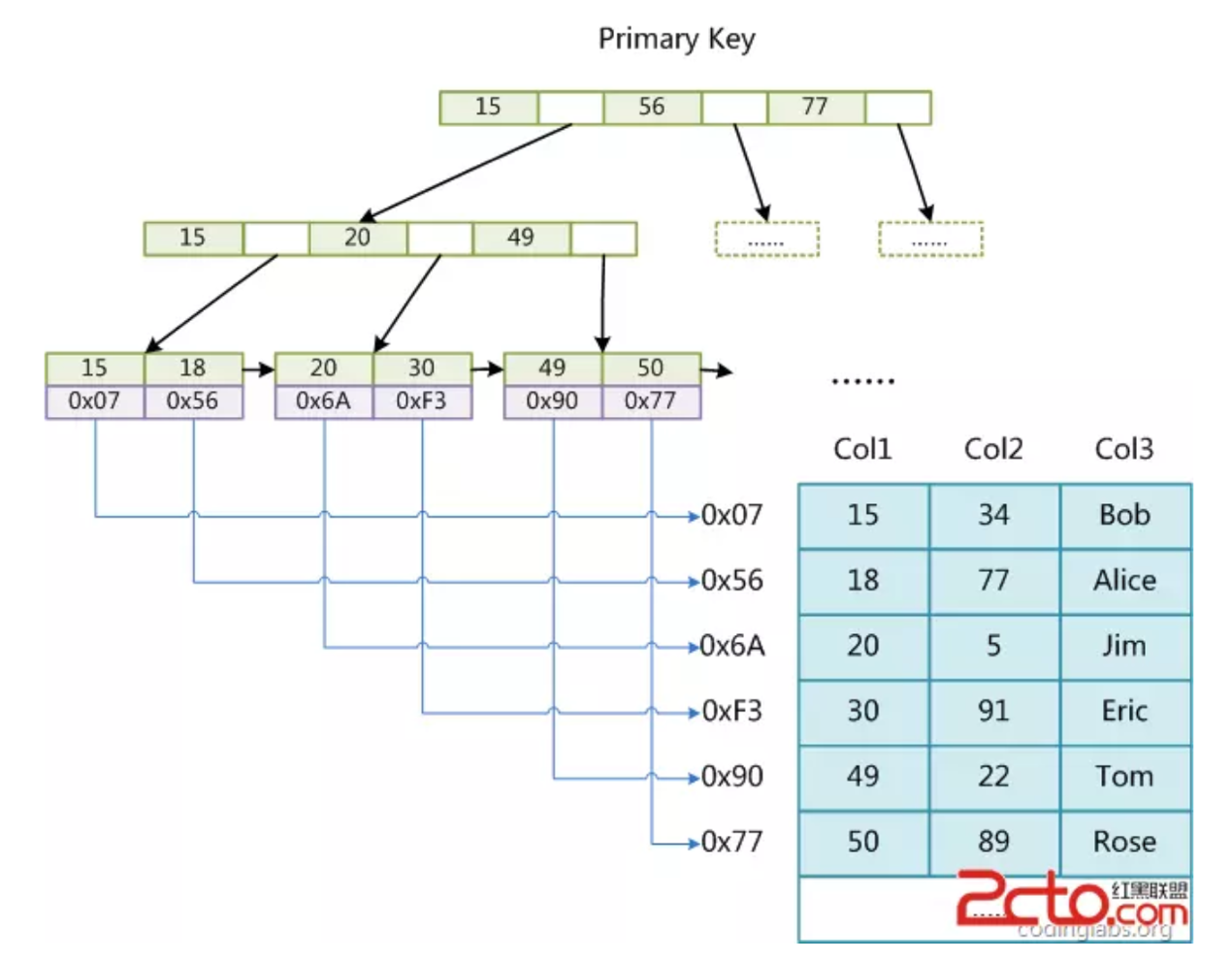


非聚簇索引：普通索引保存关键字，叶子结点保存主键值，然后去聚簇索引查询对应的记录



## MyISAM索引实现：天然非聚簇索引

索引结构：B+树保存索引，叶子结点保存数据在磁盘中的地址



## B+树深究：

为什么是B+树：

二叉搜索树——深度太深，意味着多次IO，速度太慢

B树——数据在页内，一次IO得到的索引少，IO多，叶子结点互相独立，范围查找困难

B+树：1、深度低，IO次数少

2、叶子结点间有指针，范围查询快

3、普通结点不包含数据，数据只存在叶子结点中

3、每个结点设计大小为一个磁盘数据页，IO次数少。

## 重点：为什么每个结点设计为一个磁盘数据页大小？

数据库的性能主要取决于IO的次数，因此设计上需要尽可能减少IO次数。

操作系统为了提高磁盘的读取性能，采用预加载，加载当前读取数据的临近数据，读取的单位是磁盘数据页。将结点设计为一个数据页大小，则保证了结点只需要一次IO就能读取（不这样的话有可能把一个结点分成多次读取）。另外，叶子结点不存储数据又使一个结点能尽可能多的存储关键字数目，进一步减少IO次数。

## 插入问题——页分裂：

由于一个结点大小固定，并且数据有序，那么当插入一个结点时，可能在原本的结点中已经没有空位了，那么就需要再新建一个结点，然后将原来结点的数据放在两个结点上，这个过程叫页分裂。但是此时会影响到父结点，可能引发父结点的分裂。

## 索引的优缺点：

优点：大幅度提高查询效率

缺点：1、占据空间大（索引树也需要空间）

2、更新慢（需要维护索引树）

3、索引创建、销毁需要时间

## 索引的注意事项

1. 适合高离散度的字段（否则难以命中）
2. 字段长度不宜过长（否则索引树空间过大）
3. 不宜建立太多的索引（空间过大）
4. 对经常作为查询条件的字段建立索引（使用频率高）
5. 经常更新的字段不适合做索引（更新要维护索引树）
6. 考虑左前缀问题

## 引擎选择：

有并发需求：InnoDB

更新频繁：InnoDB

其他：MyISAM

## 事务四个特性：ACID

A-原子性：事务是一个整体，全执行/不执行

C-一致性：更加类似于正确性，数据库从一个满足所有约束的状态转移到另一个满足所有约束的状态。（原子性、隔离性、持久性更像是手段，一致性更像是目的）

I-隔离性：多个事务之间互相影响的程度，四个隔离级别，不同的隔离级别对应不同的并发问题。

D-持久性：事务提交后对数据库的影响是永久的。

## 事务的隔离级别：

隔离级别： 问题

读未提交： 脏读 读B事务修改的数据，结果B回滚

读已提交： 不可重复读 A事务读取两次数据，但是次数据被B事务修改了，导致A事务两次读取结果不同。

重复读： 幻读

序列化： 无

基本语句：

建表：create table XXX(

`id` int primary key,

`name` varchar(16) not null

);

插入：insert into XXX(field1,field2) values(v1,v2);

删除：delete from XXX where …

更新：update ]XXX set xx=xx where …

修改表：

alter table XXX add `name` varchar(16) not null; 加字段

alter table XXX drop `name`; 删字段

alter table XXX add index nameIndex(`name`); 加索引

alter table XXX drop index indexName; 删索引

alter table XXX add constraint