# MYSQL

## 语法

### msyql语句执行顺序

代码写的顺序：

select ... from... where.... group by... having... order by..

代码的执行顺序：

from... where...group by... having.... select ... order by...

## 索引

* Mysql数据库的索引结构采用的B+tree 索引结构。

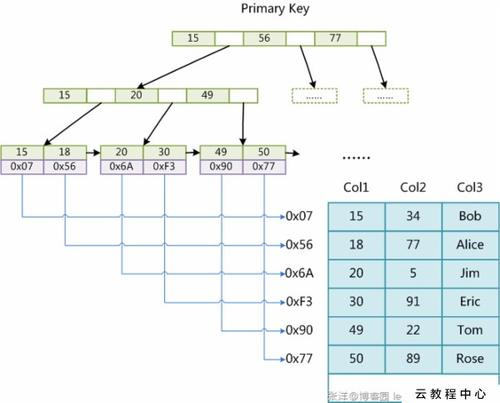
二叉树（存在单子树情况）->

平衡树（针对二叉树的缺点，每一个节点的左右子树的高度差不能超过1）->

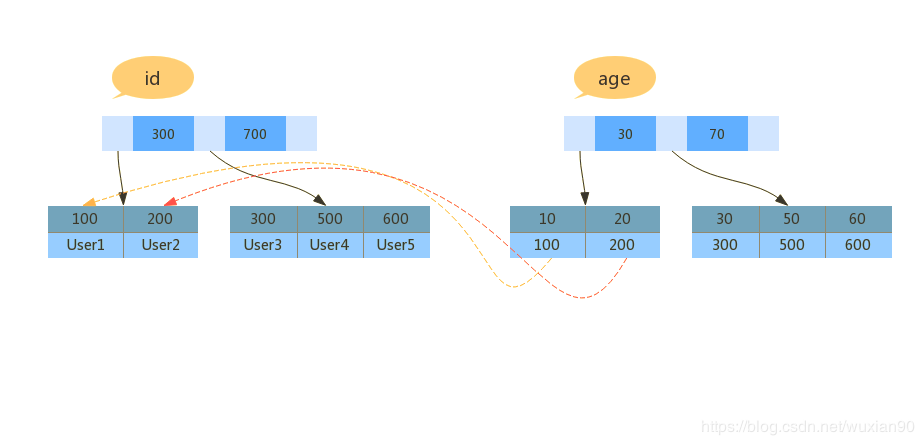
B-tree（多插树，所有的叶子节点都出现在同一层，并且叶子节点不包含所有关键字信息）-> B+tree（所有的叶子节点都出现在同一层，关键字信息都在叶子节点，并且叶子节点按照关键字大小顺序链接）

### B+tree结构的索引分两类： 聚集索引， 普通索引

聚集索引（即主键）：叶子节点存放表中所有行数据记录的信息。创建一张表的时候 一般要显示地为表创建一个主键（聚集索引）；如果不主动创建主键，InnoDB会选择第一个不包含有null值的唯一索引作为主键；如果连唯一索引都没有，InnoDB会默认为该表生成一个rowId作为主键。



### 聚集索引， 普通索引



从上图可以看出，有 2 个索引结构：主键ID 索引和普通索引。主键索引的叶子节点存储的是行数据的内容（聚簇索引），普通索引的叶子节点存储的是主键的值（非聚簇索引/二级索引）

主键索引和普通索引的区别

当我们使用主键索引查询记录时，查询语句如下所示。此时只需要一次主键索引树的查找即可返回数据行。

SELECT \* FROM t\_user WHERE id = 100;

如果使用普通索引，idx\_age 查询记录，如下所示。此时就会查找2 个索引树的结构。首先根据idx\_age 查找到记录的主键值为 100，然后再根据主键索引树查找到对应的记录行，这个过程称为回表。

SELECT \* FROM t\_user WHERE age = 10;

### 覆盖索引（注：非索引类型）

上面我们已经提到了 回表的概念了，也就是普通索引的查询，可能会再到主键索引上面再搜索一遍。但是如果我们执行如下语句：

SELECT id FROM t\_user WHERE age = 10;

此时，普通索引 idx\_age 的叶子节点上面，就已经包含了id 的value值了，此时就不需要回表了，这个就称之为“覆盖索引”（覆盖索引是一种优化查询的方式，不是索引的分类）。

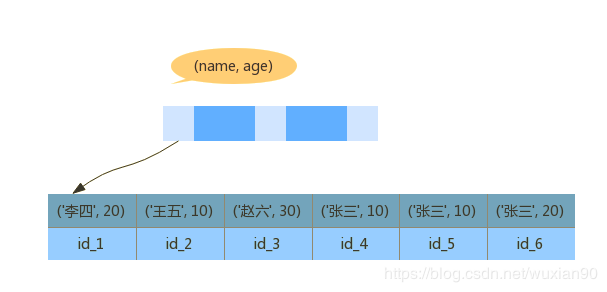
### 前缀索引

对Blob,text, 长varchar类型的列，给他们的前几个字符建立索引

Alter table table name add key(columNeme(前缀长度))

### 联合索引

我们创建索引时，也会经常创建如 idx\_name\_age (name, age) 这样的索引结构。并且还知道 WHERE 条件中 name = ? AND age = ? 和 name = ? 都可以使用到这个联合索引。下面我们来看一下其结构，看一下为什么是可以做到这一点的。



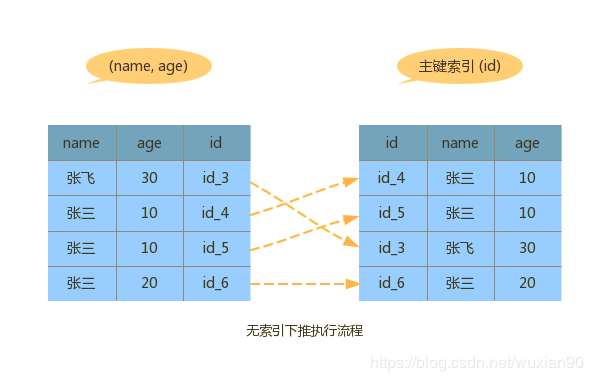
从上面结构可以看出，数据是按照 联合索引 从左到右的顺序进行排序的。由此看来，不论使用 name AND age 或者name 来查询，不论等值或者 左前缀模糊查询，都可以用到复合索引。这里面需要注意的是，只有左前缀的模糊匹配才可以使用此联合索引。因为从索引结构看来，符合左前缀的顺序排序。

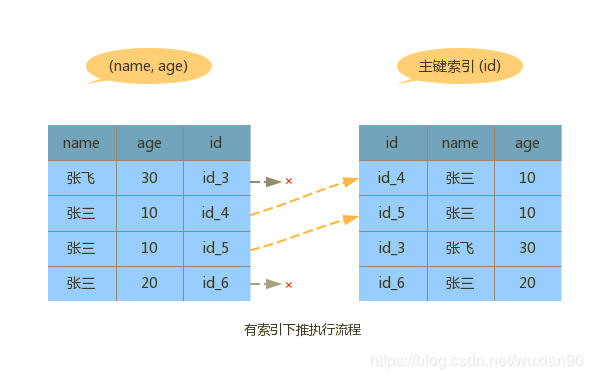
### 哈希索引

Hash算法为键哈希值-指向数据地址

### 索引下推

前面的部分我们知道，左前缀的模糊查询可以使用索引。还是上面的例子，索引(name, age) ，当我们 WHERE条件中使用 name LIKE ‘张%’ AND age = 10 时。MySQL 5.6 及以后的版本可以对查询做下推的优化，如下图所示：





从上图可以看出，当做了下推优化后，MySQL会隔断一些不满足条件的记录 进行回表操作，从一定程度上有了性能的提升。

部分内容参考：

<https://blog.csdn.net/fouy_yun/article/details/87889263>

## SQL性能优化

### MYSQL 分页查询优化

#### 分页原理：

Limit+偏移量实现分页:

select \* from t where colum =X order by Y limit n1,n

mysql innoDB分页工作原理:

innoDB先查询出n1+n条记录

然后舍掉前面的n1条记录

返回剩下的n条记录

#### 分页慢查询优化

数据量大500W以上分页查询性能差

尽量采用利用覆盖索引方式提高查询性能:

Select \* from t1 inner join( select id from t where xxx\_id = X limit n1,20) t2 on t1.id = t.id

注意：使用inner join而不能使用in 来嵌套子查询，是因为in的嵌套子查询不能使用limit

### Mysql优化方方面面

#### 硬盘优化：SSD

#### 配置参数优化

存储引擎缓存池的大小，数据库最大连接数，日志参数（全量日志关闭）等

#### 操作系统

I/O调度模式（deadline还是noop）,文件系统类型选择

#### 表设计和sql方面的优化：

* 库名，表名，字段名字母小写，采用’\_’下划线分割。
* 字段类型和大小 尽量遵循简单够用的原则。字段和索引数量不宜过多。
* 表的存储引擎选择InnoDB. Mysql8.0废弃了MyISAM.
* 表的字符集采用utf8,或者utf8mb4
* 显示的给表创建自增主键（int类型或者bigInt）.
* 很长的字符串列可以考虑创建前缀索引
* 索引字段不使用函数

原因：如果对字段做了函数计算，就用不上索引了，这是 MySQL 的规定。

* 金钱和日期，时间及IP都可以采用int类型
* 字段定义默认不允许为null

（1）所有使用NULL值的情况，都可以通过一个有意义的值的表示，这样有利于代码的可读性和可维护性，并能从约束上增强业务数据的规范性。

（2）NULL值到非NULL的更新无法做到原地更新，更容易发生索引分裂，从而影响性能。(null -> not null性能提升很小，除非确定它带来了问题，否则不要当成优先的优化措施)

（3）NULL值在timestamp类型下容易出问题，特别是没有启用参数explicit\_defaults\_for\_timestamp

（4）NOT IN、!= 等负向条件查询在有 NULL 值的情况下返回永远为空结果，查询容易出错

* 尽量不采用like 模糊查询
* Join列的类型和字符集保持一致（pg不一致易报错），超过三个表禁止用join(将复杂逻辑交给应用层).连接键字段要有索引。见例子1.1
* 利用覆盖索引优化查询—分页查询

#### show 命令查看慢sql

show processlist命令查看了当前正在执行的sql语句,同时可以查看用户的当前连接

查看慢日志

show variables like '%slow\_query\_log%';

show variables like 'long\_query\_time%';设置慢日志记录什么样的SQL，默认10s

log-queries-not-using-indexes：未使用索引的查询也被记录到慢查询日志中,一般也开启这个变量

show status查看mysql运行状态

show global status like '%Slow\_queries%';查看mysql中有多少条慢查询记录

例子1.1 查询 选修张三老师所教课程的学生中，成绩最高的学生姓名和成绩。

s(sid,sname), t(tid,tname), c(cid,cname), sct(sid,cid,tid,score)

优化前：

select sname, cname, score

from s,t,c,sct

where

sct.sid = s.sid and sct.cid = c.cid and sct.tid = t.tid and t.name = ‘张三’

stc.score = (select max(score) from sct where sct.tid =t.tid)

优化 拆分多个查询

select t.id，max（sct.score） from sct ，t where sct.tid = t.tid and t.tname = ‘张三’

select sct.sid from sct, t where t.tid = sct.tid and t.name = ‘张三’ and sct.score = 600;

select s.name sct.score , c.cnme from s,sct ,c where s.sid in(123,345) and s.sid = sct.sid and sct.cid = c.cid;

# DB事务

## 事务的四个特性

### 原子性

### 一致性

### 隔离性

### 持久性

## 隔离级别

事务的一致性依赖于事务的隔离级别

### 读未提交

### 读提交

### 可重复读

mysql默认的事务隔离级别 避免了脏读，幻读，不可重复读现象

### 串行

读-共享锁，写-排他锁。 并发能力弱。超时和锁竞争

# MYSQL语法

inner join ，inner join可以理解为“**有效的连接”，就是根据on后面的关联条件，两张表中都有的数据才会显示**

left join：理解为“主全显,后看on”(主表数据不受影响)，即主表全显示，连接后的表看on后面的选择条件，left join后面的条件，并不会影响左表的数据显示，左表数据会全部显示出来，连接的表如果没有数据，则全部显示为null

right join  right join理解为“主看on,后全显”(右表数据不受影响)，即右表数据全部显示，主表数据看on后面的选择条件