**Mysql**

1. mysql的基础语法

2. 建表原则

3. 多表查询 (多表子查询)

day 08 : mysql

1. 常见的数据库:

- 关系型数据库:

mysql : 小型.开源免费,大多数中小型互联网公司都在使用这个,sun公司收购了mysql (集群) --> aliSQL

Oracle : 中大型的数据库,服务收费的,安全稳定. 证券,银行,保险公司,金融, Oracle收购了sun公司, mysql6.x之后就开始收费了

sybase : powerdesinger 数据建模工具

SQLite : 运用在嵌入式系统中的小型数据库,android,IOS

mariadb: 开源免费数据库, 和mysql基本上一模一样,但是它是免费的,并且在不停的更新,是mysql开源版本的一个分支

SQL Server : 微软公司 大学里面教学使用比较多, 政府官网, 图形化界面做的很好

DB2 : IBM公司, 主要运用在银行系统, IBM主要是服务器软硬件解决方案

- 非关系型数据库: key/value

redis : 保存到内存

Mongodb : 保存到硬盘

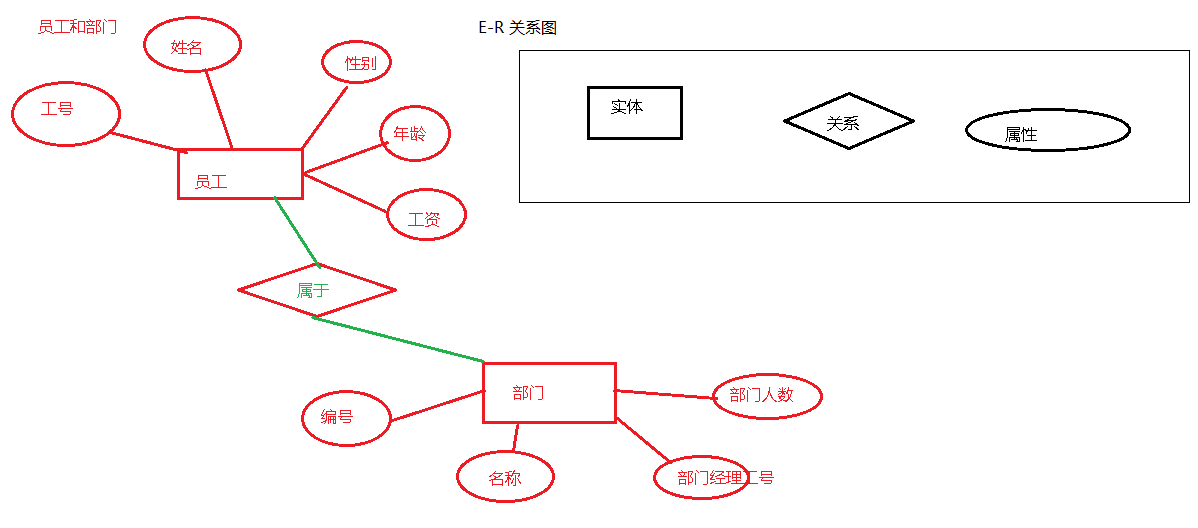
2. 关系型数据库: E-R关系图

椭圆 --> 属性

菱形 --> 属于

方形 --> 实体

------> 用实线连接



1. MYSQL的SQL语句 ---> SQL: Structured Query Language 结构化的查询语言

DDL: 数据定义语言 :

定义数据 库 或者 表 结构 : alter(修改) create(创建) drop(删除) truncate(截断)

DML: 数据操纵语言: ( M --> manage ? manipulate --> 这个有可能 (操纵；操作；巧妙地处理；篡改) )

用来 操作 表 中数据: insert(增) delete(删) update(改)

DQL: 数据查询语言:

用来 查询 表 中数据: select(选择) from(从哪里查) where(哪一条,判断条件)

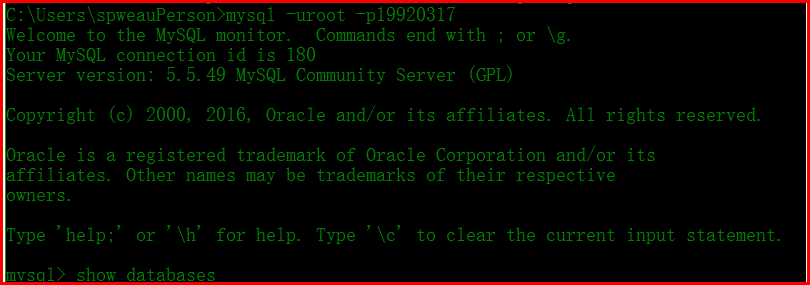
DCL: 数据控制语言:

用来控制用户权限: grant(授权) revoke(取消授权) --> DBA

4 . sql 语句 以 ; 结束

Cmd 启动 Mysql :

登录数据:  mysql  -uroot  -p   
 -u   username 用户名   
 -p   password 密码  
  
 ```sql  
 C:\Users\spweauPerson>mysql -uroot -p  
 Enter password: \*\*\*\*\*\*\*\*  
  
 ```



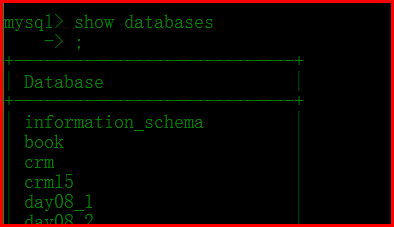
4.1 ----------------------- 数据库的操作 DDL

// 4.1.1 查询数据库

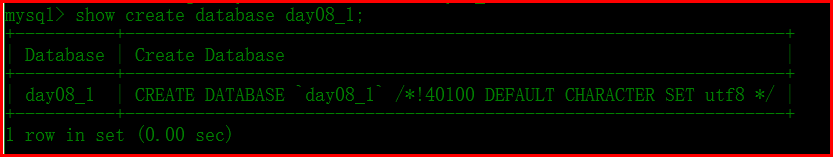
// mysql的原始模样  
 mysql> show databases ;

+--------------------+  
 | Database       |  
 +--------------------+  
 | information\_schema  | ----> 存放mysql需要的信息 禁止删除   
 | mysql        | ----> 存放用户相关信息   禁止删除   
 | performance\_schema  | ----> 数据库运行时的信息  禁止删除   
 | test         | ----> 空数据库 可删除   
 +--------------------+

mysql> show databases ; // 显示所有的数据库



mysql> show create database day08\_3; // 显示单个数据库 --> 显示详细信息



// 创建数据库 指定编码格式

create database databaseName character set 字符集 ; -- 一般采取默认的编码格式 (utf8)

// 创建数据库 指定编码格式 和 效验规则

create database 数据库的名字 character set 字符集 collate 校对规则;

校对规则: 实际上是字符的比较规则  
  a : 110   A:111  
  utf8\_general\_ci : 默认的规则 不区分大小写  a=A 成立  
  utf8\_bin   : 比较的是编码  区分大小写   a=A 不成立  
    
  字符在条件判断的时候,会使用校对规则

// 4.1.2 创建数据库

create database 数据库的名字 ;

create database day08\_1;

// 4.1.3 修改数据库 编码

alter database 数据库的名字 character set 字符集;

alter database day08\_2 character set utf8;

// 4.1.4 删除数据库

drop database 数据库名字;

mysql> drop database test;

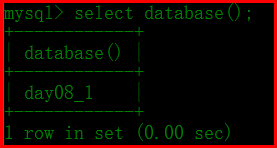
// 4.1.5 其他数据库命令

--切换数据库(选中数据库) --> use 数据库名字 ;

要对表进行操作,要先指定是哪一个数据库 , 所以 要先使用 use xxx,来确定是使用哪一个 库 ;



--(查询)当前选中是哪个数据库,当前正在使用哪个数据库 --> select database(); --> 注意后面有一个 ( ) ;



4.2 -- 表的操作 (CRUD) -----------> DML , DQL

4.2.1 表的创建

create table tablename (

列名 列的类型 [列的约束],

列名2 列的类型 [列的约束]

);

列的类型(常用的) :

java mysql

int int

String char(长度)/varchar(长度)

char(8) : 固定长度字符串 : hello , 占满5个, 不够8个用空格补齐

varchar(8) : 可变长度字符串 : hello 只占满5个, 不能超过最大长度

boolean boolean

double (只有double类型)

Date date : YYYY-MM-DD

time : hh:mm:ss

datetime: YYYY-MM-DD hh:mm:ss 默认值是null

timestamp:YYYY-MM-DD hh:mm:ss 时间戳,默认值是当前的时间 ,最大时间2038年

text: 大文本类型,存一本小说(最大2G?)

blob: binary large Object二进制大对象类型 大电影 java代码去存放和读取



--建一张员工表 employee ----> emp(工号,姓名,性别,年龄,工资)

create table emp(

empno int,

ename varchar(20),

gender varchar(2),

age int,

sal double

);

-- 列的约束 : 约束, 限制插入的字段必须复合规则

主键约束: primary key 不能重复, 不能为空 通常用在编号ID

唯一约束: unique 唯一约束: 不能重复,但是可以为空

非空约束: not null 不能为空

--

create table emp1(

empno int primary key, -- 主键约束 , 不能为空

ename varchar(20) unique, -- 唯一约束 , 不能重复 , 但是可以为空

gender varchar(2),

age int,

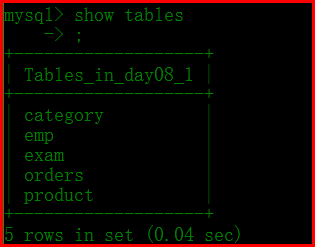
sal double not null -- 不能为空

);

4.2.2 表的语法

-- 查看所有的表

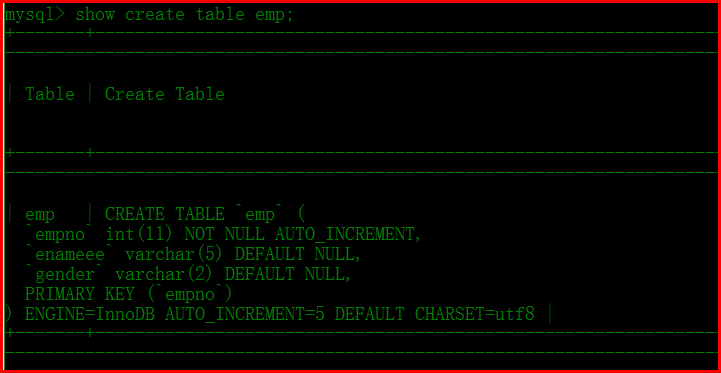
show tables;



-- 查看表的创建过程,建表语句

show create table 表名

show create table emp;



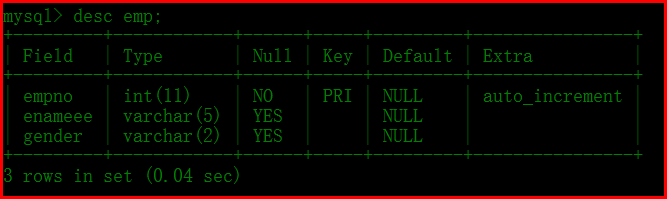
int 类型 默认是 11 个字节

-- 查看表结构

desc 表名

desc emp;

mysql> desc emp;



-- 修改表 : 添加列(add),修改列类型(modify),修改列名(change), 删除列(drop),修改表名(rename),修改表的字符集

添加列(add) alter table 表名 add 列名 列名类型

--添加手机号

alter table emp add phone int;

修改列名(change) alter table 表名 change 旧的列名 新列名 列的类型

--- 修改phone ---> mobile

alter table emp1 change phone mobile varchar(11);

-- change 可以修改列名的同时 修改列名类型

mysql> alter table emp change mobile phone int;

-- change 可以 只修改 列名类型

mysql> alter table emp change mobile mobile varchar(11);

-- 注意 修改表后面的类型 一定要添加上 , 否则 error

-- 如果只修改列名 不修改 类型 --> 只要最后的类型 与之前的保持一致即可

修改列名类型(modify) alter table 表名 modify 列名 列的类型

-- 修改手机号的类型 --- > varchar

alter table emp1 modify phone varchar(11);

删除列(drop) alter table 表名 drop 列名;

-- 删除mobile

alter table emp1 drop mobile;

修改表名(rename) rename table 旧表名 to 新表名

rename table emp1 to employee;

修改表的字符集 alter table employee character set 字符集 --> 一般不修改,使用默认的

alter table employee character set gbk;

-- show create table employee;

-- ENGINE=InnoDB DEFAULT CHARSET=utf8

-- 修改后

-- ENGINE=InnoDB DEFAULT CHARSET=gbk

-- 删除表 ---------------------------------------

drop table 表名

drop table employee;

4.2.3 表中数据的CRUD的操作

--创建一张表

create table emp(

empno int primary key auto\_increment,

ename varchar(20),

gender varchar(2)

);

--------------------------------- insert ----------------------------------

--指定列名插入:insert into 表名(列名1,列名2) values(值1,值2)

注意: 列名的数量和值的数量要一一对应

insert into emp(empno,ename) values(1,'zs');

insert into emp(empno,ename) values(null,'zs');

--不指定列名插入:必须按照顺序,把表中所有的列的数据都填上,如果不填必须填null

insert into emp values(null,'lisi','nan'); -- Data too long for column 'gender' at row

insert into emp values(null,'lisi','nv');

insert into emp values(3,'ww','nv'); -- 异常 Duplicate entry '3' for key 'PRIMARY'

insert into emp values(null,'李四','nv'); -- 插入的中文乱码

-- ERROR 1366 (HY000): Incorrect string value: '\xC0\xEE\xCB\xC4' for column 'ename' at row 1

主键 --> 一般不填 , 让其自动增长

---> 中文乱码的解决问题

安装数据库时 ,使用的字符集是 utf8

在cmd中输入的字符集是 : gbk

-- 临时解决方案

set names gbk; -- 会有问题 , 退出cmd , 在进入会看到乱码

mysql> insert into emp values(null,'李四','nv');

ERROR 1366 (HY000): Incorrect string value: '\xC0\xEE\xCB\xC4' for column 'ename' at row 1

mysql> set names gbk;

Query OK, 0 rows affected (0.00 sec)

mysql> insert into emp values(null,'李四','nv');

Query OK, 1 row affected (0.05 sec)

mysql> insert into emp values(null,'李四','男');

Query OK, 1 row affected (0.05 sec)

mysql> insert into emp values(null,'李四','男男'); -- 汉字 也默认是一个字符

Query OK, 1 row affected (0.10 sec)

mysql> select \* from emp;

+-----------+----------+-----------+

| empno | ename | gender |

+-----------+-----------+-----------+

| 1 | zs | NULL |

| 2 | zs | NULL |

| 6 | zs1 | NULL |

| 7 | lisi | nv |

| 8 | 李四 | nv |

| 9 | 李四 | 男 |

| 10 | 李四 | 男男 |

+----------+-------------+--------------+

-- 退出cmd , 在进入会看到乱码

mysql> select \* from emp; > 退出,在打开cmd --> 查看

+-------+--------+--------+

| empno | ename | gender |

+-------+--------+--------+

| 1 | zs | NULL |

| 2 | zs | NULL |

| 6 | zs1 | NULL |

| 7 | lisi | nv |

| 8 | 鏉庡洓 | nv |

| 9 | 鏉庡洓 | 鐢? |

| 10 | 鏉庡洓 | 鐢风敺 |

+-------+--------+--------+

7 rows in set (0.00 sec)

终身解决方案:

0. 先关闭mysql的服务

1. 找到mysql安装路径下的my.ini文件 D:\cz\Tool\MySQL

2. 找到文件中的第57行,将编码修改成gbk;

3. 重新打开cmd, 重新开启mysql的服务

-- 主键的插入 不可以插入已经存在的 , 但可以插入 不存在的主键

mysql> insert into emp(empno,ename) values(1,'zs');

Query OK, 1 row affected (0.06 sec)

mysql> select \* from emp;

+-----------+----------+-----------+

| empno | ename | gender |

+----------+-----------+-----------+

| 1 | zs | NULL |

+----------+----------+------------+

1 row in set (0.00 sec)

mysql> insert into emp(empno,ename) values(null,'zs');

Query OK, 1 row affected (0.06 sec)

mysql> select \* from emp;

+-------+-------+--------+

| empno | ename | gender |

+-------+-------+--------+

| 1 | zs | NULL |

| 2 | zs | NULL |

+-------+-------+--------+

2 rows in set (0.00 sec)

-- 插入已经存存在的主键

mysql> insert into emp(empno,ename) values(1,'zs1');

ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

-- 插入不存存存在的主键

mysql> insert into emp(empno,ename) values(6,'zs1');

Query OK, 1 row affected (0.07 sec)

mysql> select \* from emp;

+-------+-------+--------+

| empno | ename | gender |

+-------+-------+--------+

| 1 | zs | NULL |

| 2 | zs | NULL |

| 6 | zs1 | NULL |

+-------+-------+--------+

3 rows in set (0.00 sec)

--------------------------------- delete----------------------------------

-- 删除记录

delete from 表名 [ where 条件 ]

delete from emp where empno=5; -- 对空格不敏感

delete from emp; --逐条删除 所有的 记录

--面试概率非常大: delete 和 truncate 区别

delete truncate

DML DDL

逐条删除 先删除表,再重新建表

执行效率要高一些

删除的数据可以回滚 删除的数据找不回来

-- 重新插入empno -- 重新插入 empno

-- 其id不会从1开始 -- 其id 从1开始

-- truncate

mysql> truncate emp;

Query OK, 0 rows affected (0.05 sec)

mysql> select \* from emp;

Empty set (0.00 sec)

mysql> insert into emp values(null,'王五','男');

Query OK, 1 row affected (0.06 sec)

mysql> select \* from emp;

+-----------+-----------+-----------+

| empno | ename | gender |

+-----------+----------+-----------+

| 1 | 王五 | 男 |

+-----------+----------+------------+

1 row in set (0.00 sec)

--------------------------------update----------------------------------

-- 更新表记录

update 表名 set 列名=列的值,列名2=列的值2 [where 条件]

-- 初始

+-------+-------+--------+

| empno | ename | gender |

+-------+-------+--------+

| 1 | 王五 | 男 |

| 2 | 赵六 | 男 |

| 3 | 张三 | 男 |

| 4 | 李四 | 男 |

+-------+-------+--------+

update emp set ename='赵六' where ename='王五'; --> 将 王五 更新为 赵六

+-------+-------+--------+

| empno | ename | gender |

+-------+-------+--------+

| 1 | 赵六 | 男 |

| 2 | 赵六 | 男 |

| 3 | 张三 | 男 |

| 4 | 李四 | 男 |

+-------+-------+--------+

-- update emp set ename='赵六' -- 全部修改了

+-------+-------+--------+

| empno | ename | gender |

+-------+-------+--------+

| 1 | 赵六 | 男 |

| 2 | 赵六 | 男 |

| 3 | 赵六 | 男 |

| 4 | 赵六 | 男 |

+-------+-------+--------+

-- update emp set gender='女';

+-------+-------+--------+

| empno | ename | gender |

+-------+-------+--------+

| 1 | 赵六 | 女 |

| 2 | 赵六 | 女 |

| 3 | 赵六 | 女 |

| 4 | 赵六 | 女 |

+-------+-------+--------+

4.2.4 查询记录---------------------

--准备一张商品分类表(分类ID,分类名称,分类描述)

create table category(

cid int primary key auto\_increment,

cname varchar(10) not null,

cdesc varchar(100)

);

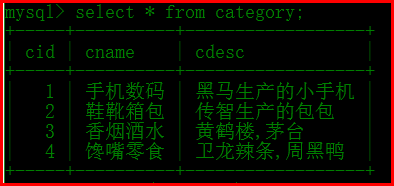
--数据的准备

insert into category values(null,'手机数码','黑马生产的小手机');

insert into category values(null,'鞋靴箱包','传智生产的包包');

insert into category values(null,'香烟酒水','黄鹤楼,茅台');

insert into category values(null,'馋嘴零食','卫龙辣条,周黑鸭');



--准备一张商品表(商品编号,商品名称,商品价格,商品描述,商品分类编号)

create table product(

pid int primary key auto\_increment,

pname varchar(10) not null,

price double,

pdesc varchar(20),

cno int

);

insert into product values(null,'锤子',2999,'可以砸榴莲的手机',1);

insert into product values(null,'小米',1599,'为发烧而生',1);

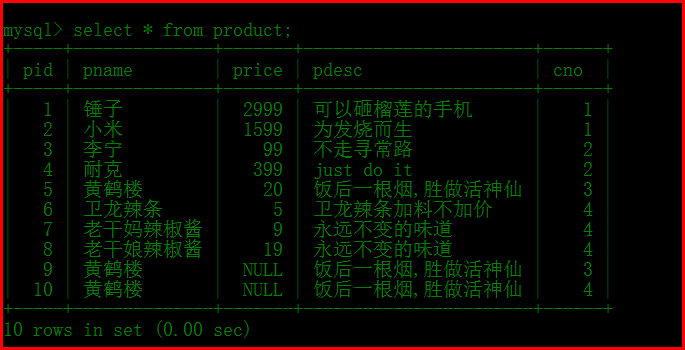
insert into product values(null,'李宁',99,'不走寻常路',2);

insert into product values(null,'耐克',399,'just do it',2);

insert into product values(null,'黄鹤楼',20,'饭后一根烟,胜做活神仙',3);

insert into product values(null,'卫龙辣条',5,'卫龙辣条加料不加价',4);

insert into product values(null,'黄鹤楼',null,'饭后一根烟,胜做活神仙',3);



查询语句的基本结构: select ... from ....

select 要显示的列名 from 从哪张表里面去查询数据 [where 条件:查询的是哪条记录/满足哪个条件]

select [distinct] [列名,列名] [\*] from 表名 [where 条件]

只是查询出结果,不会改变数据库里面原有的内容

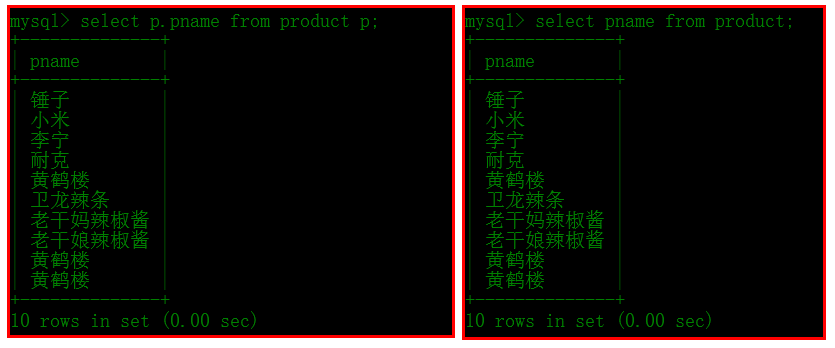
--简单查询:

---查询表中所有的商品信息: select \* from product;

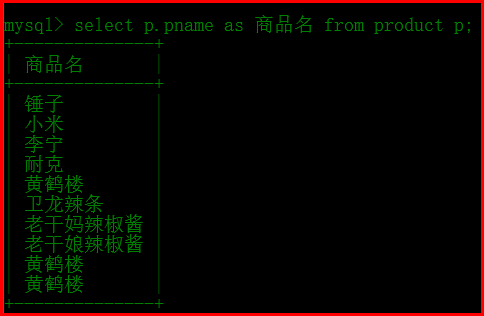
---查询商品名称和商品价格(多个字段 用 , 分隔): select pname,price from product;

---别名查询. as 的关键字 , as 关键字是可以省略

--表别名: 多表查询里面使用 ----- select p.pname from product p; == select pname from product ; 显示一样



--列别名: ----- select pname as 商品名称 from product; --> 显示的是 列 改变的 名称



-- : 注意 不能直接查找已经改名的表

mysql> select p from product p;

ERROR 1054 (42S22): Unknown column 'p' in 'field list'

-- 同理

mysql> select product from product ;

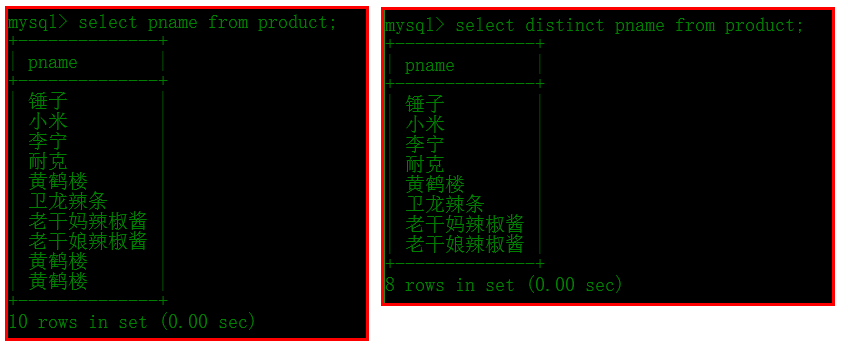
ERROR 1054 (42S22): Unknown column 'product' in 'field list'

-- 原因 select 选择的是 字段名 不可以 填写 表名

---去掉重复的值 distinct

--查询所有商品的名称

mysql> select distinct pname from product;



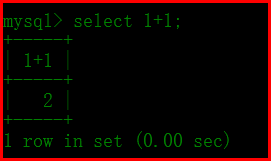
-- 多列去除重复的, 必须是每一列都一样才算重复 (是 & 的关系)

select distinct pdesc,cno from product; -- pdesc 和 cno 两列 的值都是 一样的 , 才能只显示1个

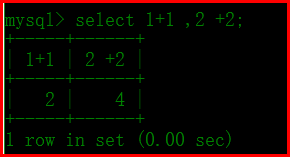
--select运算查询 : 仅仅在查询结果上做了运算 + - \* /

select 1+1;

mysql> select 1+1; -- 展现形式 很怪

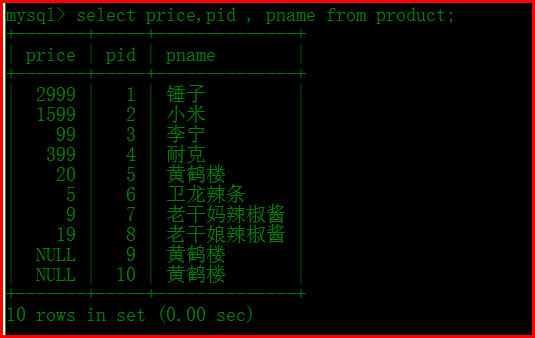


mysql> select 1+1 ,2 +2;

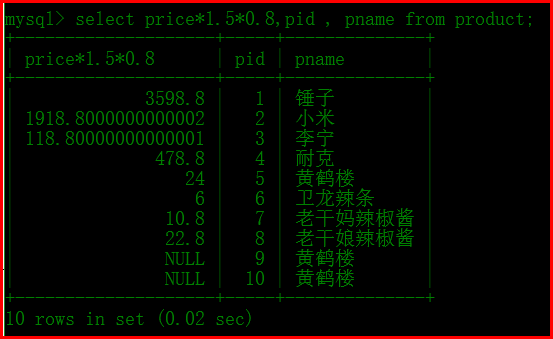


--查询商品的折后价格

mysql> select price,pid , pname from product;



mysql> select price\*1.5\*0.8,pid , pname from product; -- 注意一下精度问题



--条件查询 [where关键字] : 查询表中符合条件的数据 -----------------------------------------------------------------------------------------------------------

-- 关系/比较运算符: > >= = < <= != <>

!= : 不属于标准SQL

<> : (不等于)标准SQL

-- 逻辑运算符: and or not

-- 其它运算符:

is null : 是否为空

is not null : 是否不为空

between...and ...: 在某个区间内: 闭区间[10,20] 开区间(10,20) [) (]

in(集合) : 在集合内

--like: 模糊查询

% 匹配多个字符

\_ 匹配单个字符

--排序查询: order by 关键字/列名

asc : ascend 升序 默认升序

desc : descend 降序

-- 查找 黄鹤楼 价格 不为null

mysql> select pname,price from product where pname='黄鹤楼' and price is not null;

--查询商品价格>60元的所有商品信息

select \* from product where price > 60;

-- where 后的条件写法

--查询商品价格不等于99的所有商品

select \* from product where price !=99;

select \* from product where price <>99;

--查询出商品价格不为null的所有商品

select \* from product where price != null; ----> 错误的演示

null : 代表的是未知的,不可预知的,不确定

select \* from product where price is not null;

-- 查询商品价格在10 到 100之间

select \* from product where price >=10 and price <=100;

select \* from product where price between 10 and 100;

--查询出商品价格 小于35 或者商品价格 大于900

select \* from product where price <35 or price >900;

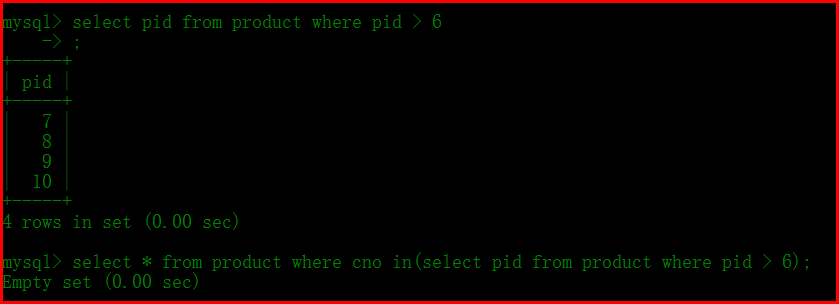
-- in 判断值是否在某个集合/范围

--查询出商品分类ID在 1,3,4里面的所有商品

select \* from product where cno in(1,3,4);

-- in 里面只接受确定值 --> 也就是说,没有办法 进行动态 查询

select \* from product where cno in(select pid from product where pid > 6); --> 没有



-- 查询出名字中带有辣的所有商品: where pname like '%辣%';

select \* from product where pname like '%辣%';

-- 查询出名字第三个字是辣字的所有商品信息

select \* from product where pname like '\_\_辣%';

-- 查询所有商品,按照价格进行排序

select \* from product order by price; -- 默认是 asc-升序

--查询所有的商品,按价格进行降序排序

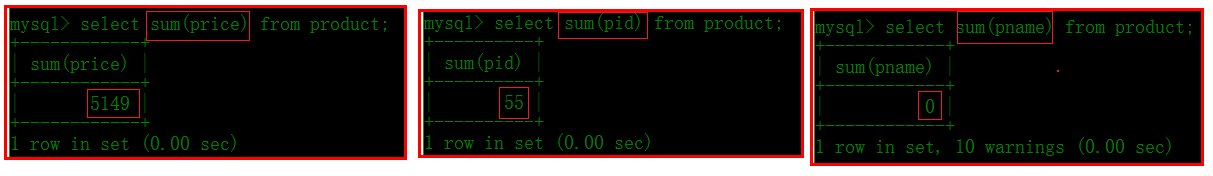
select \* from product order by price desc;

-- 聚合函数: 统计数据 --> 注意字段

sum() 求和 avg()求平均值 count()统计个数 max()最大值 min()最小值

--1.获得所有商品价格的总和: --> 注意字段

select sum(price) from product;



--2.获得所有商品的平均价格:

select avg(price) from product;

--3.获得所有商品的个数:

select count(pid) from product;

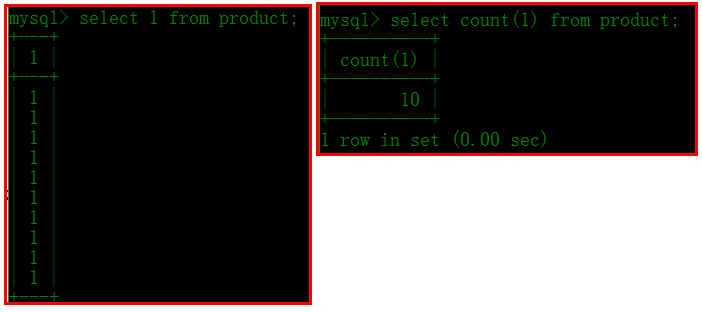
select count(\*) from product;

select count(1) from product;

select 1 from product; -- 在product里面 (添加了 --> 只是打印出来了 , 对表的内部实际上没有改变) 字段是 1 内容全是1 的数据

mysql> select 1 from product;

select count(1) from product; -- SQL优化, count(常量) 高效一点



-- 注意 : where 条件后面不能接聚合函数 , having 后面可以加

where 是在分组之前执行 ,having分组之后执行

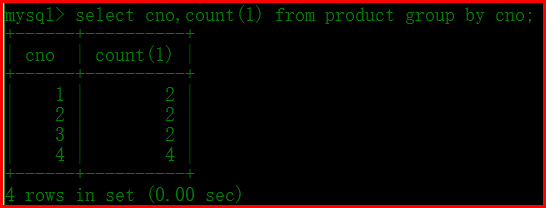
--分组: group by

select 分组的条件,分组之后的操作 from 表名 group by 分组条件 [having 条件过滤];

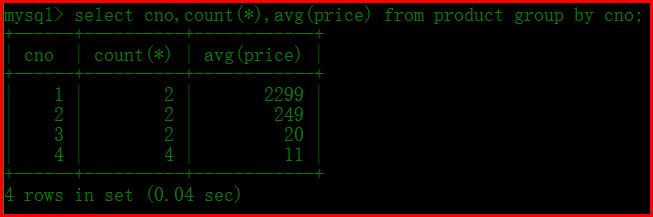
select后控制的是要显示的内容, 只写分组的条件和分组之后的操作

--1.根据cno字段分组,分组后统计商品的个数

mysql> select cno,count(1) from product group by cno;



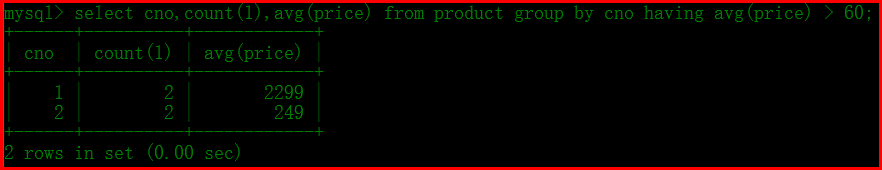
mysql> select cno,count(\*),avg(price) from product group by cno;



--分组之后的条件过滤: having

根据cno分组,分组统计每组商品的平均价格 并且 商品平均价格 > 60

mysql> select cno,count(\*),avg(price) from product group by cno having avg(price) > 60;



**-- SQL编写顺序**

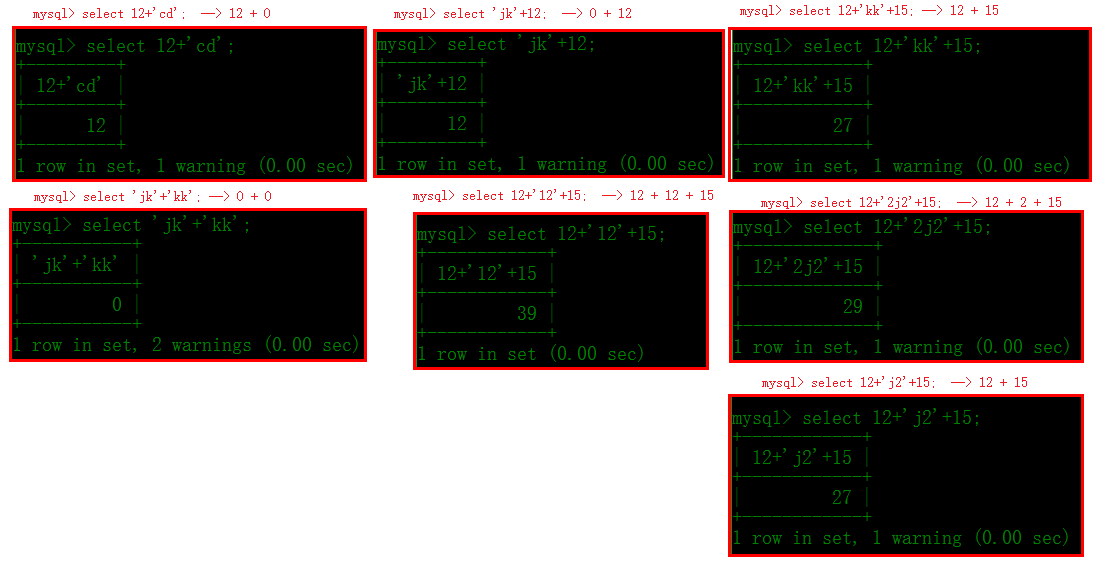
**S..F..W..G..H..O select | from | where | group by | having | order by**

**-- SQL执行顺序**

**F..W..G..H..S..O**

-- 扩展:

-- select运算查询 : 仅仅在查询结果上做了运算 + - \* /



parseInt (ahd123) --> 123

parseInt (123 2hhs 123) --> 1232

day 09 : mysql -- MYSQL数据库的多表查询 及 建表的操作 -----------------------------------------------------------------------------

-- 五大约束:

主键约束(primary key) : 检查列的值, 非空并且唯一

唯一约束(unique) : 不能重复,必须是唯一

非空约束(not null) : 不能为空

检查约束(?) : mysql里面不支持,在插入数据时,按照指定规则检查数据 check(条件)

外键约束 : 主要是用来约束多张表之间的关系, 一张表中的记录必须是存在于另外一张表, 子表中的记录必须是存在于父表/主表中

--创建一个新的数据库 day09\_1;

create database day09\_1;

--创建分类表 cid分类的ID cname分类名称 cdesc分类描述

create table category(

cid int primary key auto\_increment,

cname varchar(20),

cdesc varchar(30)

);

insert into category values(null,'手机数码','黑马生产的小手机');

insert into category values(null,'鞋靴箱包','传智生产的包包');

insert into category values(null,'香烟酒水','黄鹤楼,茅台');

insert into category values(null,'馋嘴零食','卫龙辣条,周黑鸭');

--创建一个商品 product pid pname price pdesc cno

create table product(

pid int primary key auto\_increment,

pname varchar(10) unique,

price double,

pdesc varchar(30),

cno int

);

insert into product values(null,'华为手机',500,'支持国产',1);

-- 插入的 垃圾 数据

insert into product values(null,'华为荣耀',600,'支持国产',11);

-- 给子表product添加外键约束

**alter table product add foreign key(cno) references category(cid);**

-- **这个时候添加不成功,因为表中本身存在垃圾数据**

mysql> alter table product add foreign key(cno) references category(cid);

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`day09`.`#sql-5dd0\_3`, CONSTRAINT `#sql-5dd0\_3\_ibfk\_1` FOREIGN KEY (`cno`) REFERENCES `category` (`cid`))

mysql> select \* from product;

+-----------+-----------------+-------------+---------------+-------------+

| pid | pname | price | pdesc | cno |

+-----------+-----------------+------------+----------------+-------------+

| 1 | 华为手机 | 500 | 支持国产 | 1 |

| 2 | 华为荣耀 | 600 | 支持国产 | 11 |

+-----------+----------------+-------------+-----------------+-------------+ -- cno == 11 --> 垃圾数据

-- --> 添加外键之前的

--------------------+

| product | CREATE TABLE `product` (

`pid` int(11) NOT NULL AUTO\_INCREMENT,

`pname` varchar(10) DEFAULT NULL,

`price` double DEFAULT NULL,

`pdesc` varchar(30) DEFAULT NULL,

`cno` int(11) DEFAULT NULL,

PRIMARY KEY (`pid`),

UNIQUE KEY `pname` (`pname`),

KEY `cno` (`cno`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8 |

+---------+---------------------------------------------

-- --> 添加外键之后的

---------------------+

| product | CREATE TABLE `product` (

`pid` int(11) NOT NULL AUTO\_INCREMENT,

`pname` varchar(10) DEFAULT NULL,

`price` double DEFAULT NULL,

`pdesc` varchar(30) DEFAULT NULL,

`cno` int(11) DEFAULT NULL,

PRIMARY KEY (`pid`),

UNIQUE KEY `pname` (`pname`),

KEY `cno` (`cno`),

**CONSTRAINT `product\_ibfk\_1` FOREIGN KEY (`cno`) REFERENCES `category` (`cid`)** --> 增加量

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8 |

-- +---------+--------------------------------------------------

-- 添加外键约束之后

-- 1. 插入数据 : 先往主表中插入,再往子表中插入

-- 2. 删除数据: 先删除子表中的记录, 再删除主表中的记录

-- 2.1 删除主表的 如果cid中删除的数值 是 cno中没有的就不报错

-- 2.2 删除主表的 如果cid中删除的数值 是 cno 有的就报错 -- 这个时候就要先删除子表的数据

-- 删除外键约束 ( 外键约束: 其实是在 子表里面系统自动创建了 一个 将两个表关联的 字段 product\_ibfk\_1)

**alter table product drop foreign key product\_ibfk\_1;**

多表之间的关系如何来维护 --> 外键约束:foreign key --> 一般建表 , 是不需要外键 约束的 , 是位了 数据库的性能 , 判断 是在 代码层进行判断 的

多表之间的建表原则

- 一对多: 学生和班级, 商品分类和商品

- 建表原则: 在多的一方添加一个键,然后让它作为外键指向一的一方

- 订单(order (key --> orderID) orderItem(多个商品 --> 根据orderID 来判断) )

- 多对多: 学生和选课

- 建表原则:创建一张中间表,将多对多的关系,拆分成一对多的关系, 中间表至少要包含两个外键,这两个外键分别指向各自原来的表

- 一对一: 公民和身份证号

- 建表原则:

1. 将两张表的主键建立连接,
2. 将两张合并成一张表

3. 将一对一的关系,拆分成一对多的关系 (添加的外键 也必须是唯一的)

- 合并表,拆表

---- 多表查询讲解

- 交叉连接查询

笛卡尔积: 两张表记录的成积 , 并没有什么意义; select \* from tablename1 tablename2; --> 但是这个笛卡尔积是多表查询的基础 (所需要的数据是从这个笛卡尔积中筛选出来的)

--------------

- 内连接查询:

--隐式内连接:

1.等值内连接: select \* from product p,category c where p.cno = c.cid;

-- select \* from product p,category c where p.cno = c.cid; -- 用到了 别名

-- select \* from product ,category where product.cno = category.cid; --可以不用别名

2.不等值内连接:

-- select \* from emp,dept where emp.deptno != dept.deptno;

-- 显式内连接:

select \* from product p inner join category c on p.cno = c.cid;

**表A inner join 表B on 条件**

inner关键字可以省略

等价于 select \* from product p **join** category c **on** p.cno = c.cid;

--------------

- 外连接查询:

-- 左外连接: 以左表为基础,将左表中所有的记录都查询出来,如果没有对应的记录,用null值填充

mysql> Select \* from product p **left outer join** category c **on** p.cno = c.cid;

+------------+-------------------------+-------------+------------------------------------------+-------------+-----------+-------------------+------------------------------+

| pid | pname | price | pdesc | cno | cid | cname | cdesc |

+------------+--------------------------+------------+------------------------------------------+-------------+------------+-------------------+-----------------------------+

| 1 | 华为手机 | 500 | 支持国产 | 1 | 1 | 手机数码 | 黑马生产的小手机 |

| 2 | 锤子 | 2999 | 可以砸榴莲的手机 | 1 | 1 | 手机数码 | 黑马生产的小手机 |

| 3 | 小米 | 1599 | 为发烧而生 | 1 | 1 | 手机数码 | 黑马生产的小手机 |

| 4 | 李宁 | 99 | 不走寻常路 | 2 | 2 | 鞋靴箱包 | 传智生产的包包 |

| 5 | 耐克 | 399 | just do it | 2 | 2 | 鞋靴箱包 | 传智生产的包包 |

| 6 | 黄鹤楼 | 20 | 饭后一根烟 | 3 | 3 | 香烟酒水 | 黄鹤楼,茅台 |

| 7 | 卫龙辣条 | 5 | 卫龙辣条加料不加价 | 4 | 4 | 馋嘴零食 | 卫龙辣条,周黑鸭 |

| 8 | 老干妈辣椒酱 | 9 | 永远不变的味道 | 4 | 4 | 馋嘴零食 | 卫龙辣条,周黑鸭 |

| 9 | 老干娘辣椒酱 | NULL | 永远不变的味道 | 4 | 4 | 馋嘴零食 | 卫龙辣条,周黑鸭 |

| 10 | 热干面 | 10 | 大武汉热干面 | NULL | NULL | NULL | NULL | -- 如果右表没有对应的记录,用null值填充

+-------------+------------------------+-------------+-----------------------------------------+----------------+------------+-------------------+-----------------------------+

-- 右外连接:以右表为基础,将右表中的所有记录都查询出来,如果没有对应的记录,用null值填充

mysql> select \* from product p right outer join category c on p.cno= c.cid;

+------+--------------+-------+--------------------+------+-----+----------+------------------+

| pid | pname | price | pdesc | cno | cid | cname | cdesc |

+------+--------------+-------+--------------------+------+-----+----------+------------------+

| 1 | 华为手机 | 500 | 支持国产 | 1 | 1 | 手机数码 | 黑马生产的小手机 |

| 2 | 锤子 | 2999 | 可以砸榴莲的手机 | 1 | 1 | 手机数码 | 黑马生产的小手机 |

| 3 | 小米 | 1599 | 为发烧而生 | 1 | 1 | 手机数码 | 黑马生产的小手机 |

| 4 | 李宁 | 99 | 不走寻常路 | 2 | 2 | 鞋靴箱包 | 传智生产的包包 |

| 5 | 耐克 | 399 | just do it | 2 | 2 | 鞋靴箱包 | 传智生产的包包 |

| 6 | 黄鹤楼 | 20 | 饭后一根烟 | 3 | 3 | 香烟酒水 | 黄鹤楼,茅台 |

| 7 | 卫龙辣条 | 5 | 卫龙辣条加料不加价 | 4 | 4 | 馋嘴零食 | 卫龙辣条,周黑鸭 |

| 8 | 老干妈辣椒酱 | 9 | 永远不变的味道 | 4 | 4 | 馋嘴零食 | 卫龙辣条,周黑鸭 |

| 9 | 老干娘辣椒酱 | NULL | 永远不变的味道 | 4 | 4 | 馋嘴零食 | 卫龙辣条,周黑鸭 |

| NULL | NULL | NULL | NULL | NULL | 5 | 电脑办公 | 联想电脑外星人 | -- 如果没有对应的记录,用null值填充

+------+--------------+-------+--------------------+------+-----+----------+------------------+

----------

---分页查询 : limit

limit : 起始索引 , 查询几条 ; 索引是0开始

select \* from product limit (2-1)\*3,3; -- 第二页 的数据

-- 前面的数据 是从第几条开始查 , 后面的数据是查几条

-- (页数-1)\*每页显示数量 -- 页数是从1开始 --> (pageNum - 1 ) \* count , count ;

练习题 :

1. 按照商品分类的名称统计商品的个数:

-- 分组 查询

select c.cname, count(1) from product p, category c where p.cno = c.cid group by c.cname;

--------------------------------------------------------------------------------------------------------

子查询 , 多表查询

--------------------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------------

多表查询练习 :

--员工信息表

CREATE TABLE emp(

empno INT, -- /\*员工编号\*/

ename VARCHAR(50), -- /\*员工姓名\*/

job VARCHAR(50), -- /\*员工工作\*/

mgr INT, -- /\*员工经理编号\*/

hiredate DATE, -- /\*员工入职日期\*/

sal DECIMAL(7,2), -- /\*员工工资\*/

comm DECIMAL(7,2), -- /\*员工奖金\*/

deptno INT -- /\*员工所处的部门编号\*/

);

insert into emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,null,20);

insert into emp values(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30);

insert into emp values(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30);

insert into emp values(7566,'JONES','MANAGER',7839,'1981-04-02',2975,null,20);

insert into emp values(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30);

insert into emp values(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,null,30);

insert into emp values(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,null,10);

insert into emp values(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,null,20);

insert into emp values(7839,'KING','PRESIDENT',null,'1981-11-17',5000,null,10);

insert into emp values(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30);

insert into emp values(7876,'ADAMS','CLERK',7788,'1987-05-23',1100,null,20);

insert into emp values(7900,'JAMES','CLERK',7698,'1981-12-03',950,null,30);

insert into emp values(7902,'FORD','ANALYST',7566,'1981-12-03',3000,null,20);

insert into emp values(7934,'MILLER','CLERK',7782,'1982-01-23',1300,null,10);

insert into emp values(7981,'MILLER','CLERK',7788,'1992-01-23',2600,500,20);

--部门表

CREATE TABLE dept(

deptno INT, -- /\*部门的编号\*/

dname varchar(14), --/\*部门的名称\*/

loc varchar(13) --/\*部门所处的位置\*/

);

insert into dept values(10, 'ACCOUNTING', 'NEW YORK');

insert into dept values(20, 'RESEARCH', 'DALLAS');

insert into dept values(30, 'SALES', 'CHICAGO');

insert into dept values(40, 'OPERATIONS', 'BOSTON');

--所有员工的信息

select \* from emp;

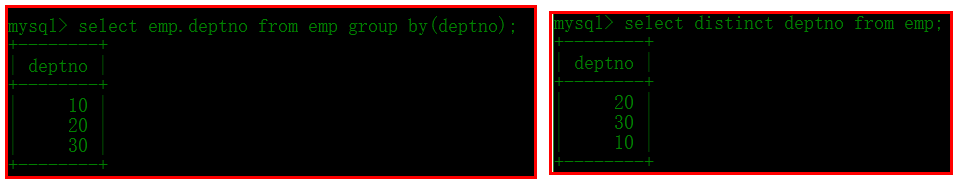
--薪资大于等于1000并且小于等于2000的员工信息

select \* from emp where sal >= 1000 and sal <=2000;

select \* from emp where sal between 1000 and 2000;

--从员工表中查询出所有的部门编号

去重方法 --> 分组去重 distinct 去重



--查询出名字以A开头的员工的信息

select \* from emp where ename like 'A%';

--查询出名字第二个字母是L的员工信息

select \* from emp where ename like '\_L%';

--查询出没有奖金的员工信息 (这个应该出了 null 还有 0.00 ??? -->看需求)

select \* from emp where comm is null; -- 注意error: comm != nulll

--所有员工的平均工资

select avg(sal) from emp;

--所有员工的工资总和

select SUM(sal) from emp;

--所有员工的数量

select COUNT(1) from emp;

--最高工资

select MAX(sal) from emp;

--最少工资

select MIN(sal) from emp;

--最高工资的员工信息 --> 子查询 (多表查询无法做到,因为不能直接获取到另一张表的参考数据,需要根据另一张表的数据计算得到参考数据)

select \* from emp where sal = (select MAX(sal) from emp);

--最低工资的员工信息

select \* from emp where sal = (select MIN(sal) from emp);

--每个部门的平均工资

select deptno,avg(sal) from emp group by deptno;

-- 单行单列子查询( **> < >= <= != <>**  ) 查询出来的结果只有一个值

-- 查询出高于10号部门的平均工资的员工信息 (可以包含10号部门员工 的工资)

-- 1. 10号部门的平均工资 2916

select avg(sal) from emp where deptno = 10;

-- 2. 得出员工信息

select \* from emp where sal > (select avg(sal) from emp where deptno = 10);

-- 多行子查询 (  **in not in any all**  ) >any >all

>**any** : 任一 : 100 ,200 , 300

-- 查询出比10号部门任一员工薪资高的员工信息

-- 1. 查询出10号部门所有的薪水

select sal from emp where deptno = 10;

-- 2. 得出结果

select \* from emp where sal > any(select sal from emp where deptno = 10) and deptno <> 10;

-- 多列子查询(实际使用较少) in

-- 和10号部门同名同工作的员工信息

-- 1. 查询10号部门的姓名和工作 信息

select ename,job from emp where deptno =10;

-- 2.得出结果

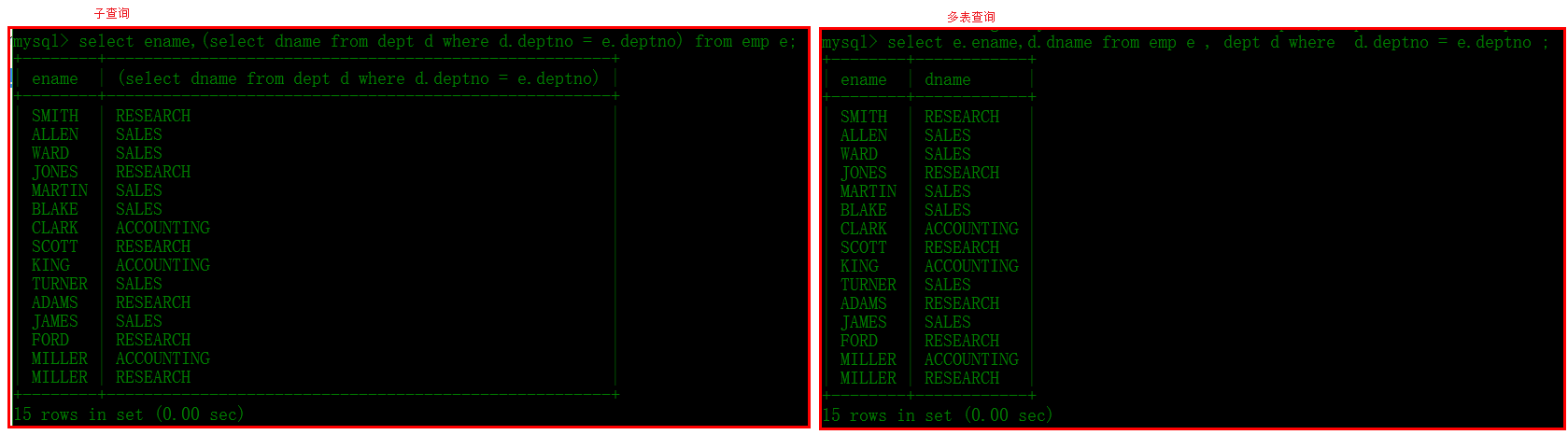
select \* from emp where (ename,job) **in** (select ename,job from emp where deptno =10) and deptno != 10;

--------------------------------------------------------------------- 子查询的位置

-- **Select接子查询**

-- 获取员工的名字和部门的名字

select ename,(select dname from dept d where d.deptno = e.deptno) from emp e;



-- **from后面接子查询**

-- 查询emp表中经理信息

-- 1. 首先查询出所有经理的ID

select distinct mgr from emp;

-- 2.得出结果

select e.\* from (select distinct mgr from emp) t\_manager, emp e where t\_manager.mgr = e.empno;

-- where 接子查询

-- 薪资高于10号部门平均工资的所有员工信息

1. 找出10号部门的平均工资 2916

select avg(sal) from emp where deptno = 10;

2. 得出结果:

select \* from emp where sal > (select avg(sal) from emp where deptno = 10);

-- having后面接子查询

-- 有哪些部门的平均工资高于30号 部门的平均工资

1. 30号部门的平均工资 1566

select avg(sal) from emp where deptno=30;

2. 统计每个部门的平均工资

select deptno,avg(sal) from emp group by deptno;

3. 得出结果 f g h s

**select deptno,avg(sal) from emp group by deptno having avg(sal) > (select avg(sal) from emp where deptno=30);**

--------------------------------------------------------------------- 练习

多表查询 也叫表关联查询 , 一般表关联查询的效率 比 子查询 效率高 ,

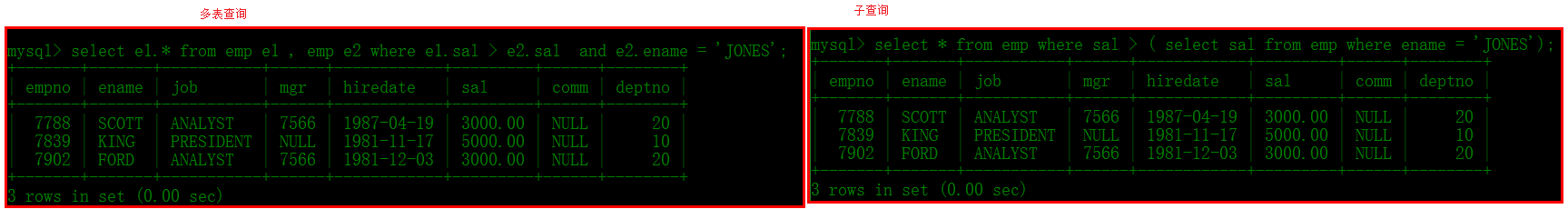
-- 工资>JONES工资 的 员工 信息

1.查询JONES的工资 2975

select sal from emp where ename = 'JONES';

2. 得出结果

select \* from emp where sal > ( select sal from emp where ename = 'JONES');



-- 查询与SCOTT同一个部门的员工

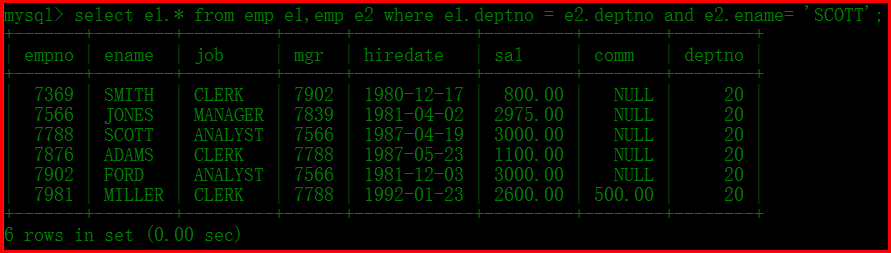
1. 查询出scott的部门编号 20

select deptno from emp where ename = 'SCOTT';

2.得出结果

select \* from emp where deptno = (select deptno from emp where ename = 'SCOTT');

表关联查询



-- 工资高于30号部门所有人的员工信息

1.查询30号最高的工资 2850

select MAX(sal) from emp where deptno =30;

2.得出结果

select \* from emp where sal > (select MAX(sal) from emp where deptno =30);

-- 查询工作和工资与MARTIN完全相同的员工信息

1. 查出MARTIN 的工作和工资

select job,sal from emp where ename = 'MARTIN';

2. 得出结果

select \* from emp where (job,sal) **in** (select job,sal from emp where ename = 'MARTIN') and ename != 'MARTIN';

-- 有两个以上直接下属的员工信息

1.分组统计mgr的数量

select mgr from emp;

select mgr,COUNT(1) from emp group by mgr ;

2.过滤出mgr出现次数大于2的记录

select mgr,COUNT(1) from emp group by mgr having COUNT(1) > 2;

3. 得出结果

select \* from emp where empno in (select mgr from emp group by mgr having COUNT(1) >2);

-- 查询员工编号为7788的员工名称,员工工资,部门名称,部门地址 --> 不是子查询了 , 表连接查询

1.查询所有员工的员工名称,员工工资,部门名称,部门地址

select e.ename,e.sal,d.dname,d.loc from emp e,dept d where e.deptno = d.deptno;

2. 得出结果

select e.ename,e.sal,d.dname,d.loc

from emp e,dept d

where e.deptno = d.deptno and e.empno = 7788;

-- 1. **查询出高于本部门平均工资的员工信息**

1.分组统计出所有部门的平均工资

select deptno,avg(sal) avgsal from emp group by deptno;

2.将emp表和上面的结果连接

select \* from emp e, (select deptno,avg(sal) avgsal from emp group by deptno) e2 where e.deptno = e2.deptno;

3.得出结果

**select \* from emp e, (select deptno,avg(sal) avgsal from emp group by deptno) e2 where e.deptno = e2.deptno and e.sal > e2.avgsal;**

-- 2. 列出达拉斯工作的人中,比纽约平均工资高的人

1. 纽约的平均工资

1. 找出纽约的部门编号 10

select deptno from dept where loc = 'NEW YORK';

2. 计算纽约的平均工资 2916

select avg(sal) from emp where deptno = (select deptno from dept where loc = 'NEW YORK');

2. 达拉斯对应的部门编号 20

select deptno from dept where loc = 'DALLAS';

3. 找出达拉斯工作的人

select \* from emp where deptno = (select deptno from dept where loc = 'DALLAS');

4. 得出结果

**select \***

**from emp**

**where**

**deptno = (select deptno from dept where loc = 'DALLAS')**

**and**

**sal > (select avg(sal) from emp where deptno = (select deptno from dept where loc = 'NEW YORK'));**

-- 2. 查询7369员工编号,姓名,经理编号和经理姓名

1. 员工编号,姓名,经理编号和经理姓名 (多表连接)

select e1.empno,e1.ename,e1.mgr,t\_manager.ename from emp e1, emp t\_manager where e1.mgr = t\_manager.empno;

2. 得出结果

select e1.empno,e1.ename,e1.mgr,t\_manager.ename from emp e1, emp t\_manager where e1.mgr = t\_manager.empno and e1.empno = 7369;

-- 3. 查询出各个部门薪水最高的员工所有信息

1.分组统计每个部门最高工资

select deptno , MAX(sal) maxsal from emp group by deptno;

2. 连接emp

select \* from emp e1,( select deptno , MAX(sal) maxsal from emp group by deptno) e2 where e1.deptno = e2.deptno;

3.得出结果

**select \* from emp e1,( select deptno , MAX(sal) maxsal from emp group by deptno) e2 where e1.deptno = e2.deptno and e1.sal = e2.maxsal;**

------------------------------------------- 练习depart2

1. 表关联查询

-- 查询员工编号,员工姓名,员工对应的 经理编号,经理姓名

select \* from emp e1 ,emp m1 where e1.mgr = m1.empno;

select e1.empno,e1.ename,e1.mgr,m1.ename

from emp e1 ,emp m1

where e1.mgr = m1.empno;

-- 查询员工编号,员工姓名,员工部门名称, 经理编号,经理姓名

select e1.empno,e1.ename,d1.dname,e1.mgr,m1.ename

from emp e1 ,emp m1,dept d1

where e1.mgr = m1.empno and e1.deptno = d1.deptno;

-- 查询员工编号,员工姓名,员工部门名称, 经理编号,经理姓名,经理的部门名称

select e1.empno,e1.ename,d1.dname,e1.mgr,m1.ename ,d2.dname

from emp e1 ,emp m1,dept d1,dept d2

where e1.mgr = m1.empno and e1.deptno = d1.deptno and m1.deptno = d2.deptno;

2. 子查询 : 查询语句嵌套查询语句,主要用来解决一些复杂查询需求

-- 统计 薪资 大于 ( (薪资最高的员工 所在部门 的平均工资) 和 (薪资最低的员工 所在部门 的平均工资) 的平均工资 ) 的员工信息

-- sal > 10 100 20 80 (100+80)/2

-- sal > 90

-- 1. 薪资最高的员工

select MAX(sal) from emp; -- 5000

-- 2. 薪资最高员工所在部门 10

select deptno from emp where sal = (select MAX(sal) from emp);

-- 3. 10号部门的平均工资 2916

select avg(sal) from emp where deptno = (select deptno from emp where sal = (select MAX(sal) from emp));

-- 1. 薪资最低的员工

select MIN(sal) from emp; -- 800

-- 2. 薪资最低员工所在部门 20

select deptno from emp where sal = (select MIN(sal) from emp);

-- 3. 20号部门的平均工资 2245

select avg(sal) from emp where deptno = (select deptno from emp where sal = (select MIN(sal) from emp));

-- 10号和20号的平均工资的平均工资 2580

select (2916+2245)/2

-- 子查询 --2581

select (

(select avg(sal) from emp where deptno = (select deptno from emp where sal = (select MAX(sal) from emp)))

+

(select avg(sal) from emp where deptno = (select deptno from emp where sal = (select MIN(sal) from emp)))

)/2;

-- 薪资 大于 2581

select \* from emp where sal > (select (

(select avg(sal) from emp where deptno = (select deptno from emp where sal = (select MAX(sal) from emp)))

+

(select avg(sal) from emp where deptno = (select deptno from emp where sal = (select MIN(sal) from emp)))

)/2);

-- 有两个以上直接下属的员工信息 经理 有两个以上直接下属 ????

-- 分组: 按经理编号进行,只要mgr出现两次以上就是满足条件的

-- 1. 分组统计mgr出现的次数

select mgr,COUNT(\*) from emp group by mgr;

-- 2. 次数大于2的

select mgr,COUNT(\*) from emp group by mgr having COUNT(\*)>2;

-- 3. 员工信息

select mgr from emp group by mgr having COUNT(\*)>2;

-- 4.只要员工编号 在 (7698,7839)

select \* from emp where empno in(select mgr from emp group by mgr having COUNT(\*)>2);

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

-- 多表之间的关系如何来维护 ? 由谁来维护

-- 添加外键约束: alter table product add **foreign key(cno) references** category(cid);

-- 建表原则

-- 一对多: 在多的一方的添加一个外键,指向一的一方

-- 多对多: 增加一张中间表,将多对多的关系,拆成一对多的关系, 中间表至少包含两个外键,分别指向它们原来的表的主键

-- 一对一: 1.将两张表的主键进行关联

-- 2.将两张表合并成一张表

-- 3.将一对一的关系,当作一对多的关系进行处理, 在任一一张表中添加一个外键指向另外一张表, 这个外键必须是唯一约束

-- 建库原则: 一个项目通常都是新建一个数据库

-- 商城案例 :用户表 订单表 中间表 商品表 商品分类

-- 多表查询:

-- 内联接: product category

隐式内链接: select \* from product p , category c where p.cno = c.cid;

显示内链接: select \* from product p inner join category c on p.cno = c.cid; --> 一般不用,一般使用隐式内连接

-- 外连接:

左外连接: select \* from product p left outer join category c on p.cno = c.cid;

右外连接: select \* from product p right outer join category c on p.cno = c.cid;

-- 子查询 : 查询语句中嵌套查询语句

-- 按照行列的划分

: > >= = < <= != <>

多行单列: in >any >all

多行多列: in

-- 按照出现的位置进行划分:

select 后面接子查询

from 后面子查询, 将查询结果当作一张表来处理, 必须得取别名

where 后面接子查询

having 后面接子查询

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

#### 面试题  
  
```sql  
CREATE TABLE test(  
  name CHAR(20),  
  kecheng CHAR(20),  
  fenshu CHAR(20)  
);  
INSERT INTO test VALUES('张三','语文',81),  
('张三','数学',75),  
('李四','语文',76),  
('李四','数学',90),  
('王五','语文',81),  
('王五','数学',82);  
  
--请用一条Sql语句查处每门分数都大于80的学生  
  
-- 找到有几个人  
select name from test group by name  
  
-- 每个人的 全部的分数  
  
select fendshu from test where name  ???  (select name from test group by name);  
  
select name from test where 80 <  all(select fenshu from test group by name);  
  
```  
  
方法一：  
  
　　思路：如果能获得一张表，由学生姓名，语文成绩，数学成绩，英语成绩的表，剩下的就是在WHERE条件中筛选及可以获得想要的结果。  
  
　　具体办法：通过自连接的办法，以“姓名”为连接条件，自连接三次，便可以获得包含又姓名和三门课程成绩的数据行。虽然可以得到想要的数据列。但会有很多冗余重复列！  
  
　　点评：此方法是根据题目，依题解题，中规中矩！ 不过多张表连接非常耗费时间。而且SQL语句也比较复杂，需要注意事项很多。  
  
　　  
  
复制代码  
复制代码  
SELECT  D.name  FROM (  
SELECT  S.name,S.score AS ITEM1,S1.score AS ITEM2,S2.score AS ITEM3  
FROM Student S   
inner join Student S1 on S.name = S1.name and S.course <> S1.course  
inner join Student S2 on S.name = S2.name and S.course <> S2.course  
WHERE S.score>=80 and S1.score>=80 and S2.score>=80  
) D   
GROUP BY D.name   
复制代码  
复制代码  
　　易错点：内表的 score字段必须要取别名，否则会报错 。  
  
   
  
　　方法二：  
  
　　思路：采用逆向思维想想。。。。。。求三门成绩都大于80的人，也可以是使先查出有成绩小于80 的人，再除去这些人不就是三门成绩都大于80的人了么？  以前学过的数学逻辑逆向思维还真是有用的阿！！  
  
　　具体办法：先扫描表，查出有成绩小于80的人的姓名，然后再次扫描表，用not in 或not exists 方法。  
  
　　点评：此方法采用逆向思维，能快速写出高效且简单的 SQL语句。  
  
复制代码  
复制代码  
//not in   
SELECT DISTINCT A.name FROM Student A   
WHERE A.name not in(  
SELECT Distinct S.name FROM Student S WHERE S.score <80)  
  
//not exists  
  
SELECT DISTINCT A.name From Student A    
where not exists (SELECT 1 From Student S Where  S.score <80 AND S.name =A.name)  
  
/\*exists 详解  
取出 外表第一条数据 ，然后与内表  根据连接条件 ，  
形成一条或多条数据，判断这些生成的数据中是否存在  
或者是不存在符合where条件的 。结果为ture的那条外表  
记录旧被查询出来！  
  
实例过程： 取出外表的第一条记录，   
和内表通过姓名条件连接，这时候产生2两记录，  
根据 not exists是判断不存在。 条件是 score<80 .  
而这两条记录存在一条记录小于80，所以于not exists 不符合，  
该条记录不被查出。  
\*/

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

1. mysql的序列
2. Mysql的索引
3. Mysql简单函数

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------

## 第7章：优化

**目录**

[7.1. 优化概述](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "optimize-overview)

[7.2. 优化SELECT语句和其它查询](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "query-speed)

[7.3. 锁定事宜](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "locking-issues)

[7.4. 优化数据库结构](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "optimizing-database-structure)

[7.5. 优化MySQL服务器](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "optimizing-the-server)

[7.6. 磁盘事宜](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "disk-issues)

优化是一个复杂的任务，因为最终要求了解整个待优化的系统。尽管可以进行局部优化而不需要了解系统或应用程序，为了优化得更好，你必须知道更多的信息。

本章解释并给出不同的优化MySQL的方法示例。但要记住总有一些其它方法使系统更快，尽管需要更多的工作。

## 7.1. 优化概述

[7.1.1. MySQL设计局限与折衷](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "design-limitations)

[7.1.2. 为可移植性设计应用程序](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "portability)

[7.1.3. 我们已将MySQL用在何处？](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "internal-use)

[7.1.4. MySQL基准套件](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "mysql-benchmarks)

[7.1.5. 使用自己的基准](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "custom-benchmarks)

使一个系统更快的最重要因素当然是基本设计。此外，还需要知道系统正做什么样的事情，以及瓶颈是什么。

最常见的系统瓶颈是：

* 磁盘搜索。需要花时间从磁盘上找到一个数据，用在现代磁盘的平均时间通常小于10ms，因此理论上我们能够每秒大约搜索1000次。这个时间在新磁盘上提高不大并且很难为一个表进行优化。**优化它的方法是将数据分布在多个磁盘上。**
* 磁盘读/写。当磁盘放入正确位置后，我们需要从中读取数据。对于现代的磁盘，一个磁盘至少传输10-20Mb/s的吞吐。这比搜索要容易优化，因为你能**从多个磁盘并行地读**。
* CPU周期。我们将数据读入内存后，需要对它进行处理以获得我们需要的结果。表相对于内存较小是最常见的限制因素。但是对于小表，速度通常不成问题。
* 内存带宽。当CPU需要的数据超出CPU缓存时，主缓存带宽就成为内存的一个瓶颈。这在大多数系统正是一个不常见的瓶颈但是你应该知道它。

### 7.1.1. MySQL设计局限与折衷

当使用MyISAM存储引擎时，MySQL使用极快速的表锁定，以便允许多次读或一次写。使用该存储引擎的最大问题出现在同一个表中进行混合稳定数据流更新与慢速选择。如果这只是某些表的问题，你可以使用另一个存储引擎。参见[第15章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html)*[存储引擎和表类型](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \o "Chapter 15. Storage Engines and Table Types)*。

MySQL可以使用事务表和非事务表。为了更容易地让非事务表顺利工作(如果出现问题不能回滚)，MySQL采用下述规则。请注意这些规则只适用于不运行在严格模式下或为INSERT或UPDATE使用IGNORE规定程序时。

·         所有列有默认值。请注意当运行在严格SQL模式(包括TRADITIONAL SQL模式)时，必须为NOT NULL列指定默认值。

·         如果向列内插入不合适的或超出范围的值，MySQL将该列设定为“最好的可能的值”，而不是报告错误。对于数字值，为0、可能的最小值或最大值。对于字符串，为空字符串或列内可以保存的字符串。请注意当运行在严格模式或TRADITIONAL SQL模式时该行为不 适用。

·         所有表达式的计算结果返回一个表示错误状况的信号。例如，1/0返回NULL。(使用ERROR\_FOR\_DIVISION\_BY\_ZERO SQL模式可以更改该行为)。

如果正使用非事务表，不应该使用MySQL来检查列的内容。一般情况，最安全的(通常是最快的)方法径是让应用程序确保只向数据库传递合法值。

相关详细信息参见[1.8.6节，“MySQL处理约束的方式”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/introduction.html" \l "constraints" \o "1.8.6. How MySQL Deals with Constraints)和[13.2.4节，“INSERT语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "insert" \o "13.2.4. INSERT Syntax)或[5.3.2节，“SQL服务器模式”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-sql-mode" \o "5.3.2. The Server SQL Mode)。

### 7.1.2. 为可移植性设计应用程序

因为不同SQL服务器实现了标准SQL的不同部分，需要花功夫来编写可移植的SQL应用程序。对很简单的选择/插入，很容易实现移植，但是需要的功能越多则越困难。如果想要应用程序对很多数据库系统都快，它变得更难！

为了使一个复杂应用程序可移植，你需要选择它应该工作的SQL服务器，并确定这些服务器支持什么特性。

所有数据库都有一些弱点。这就是它们不同的设计折衷导致的不同行为。

可以使用**MySQL**的**crash-me**程序来找出能用于数据库服务器选择的函数、类型和限制。**crash-me**并不能找出所有的特性，但是其广度仍然很合理，可以进行大约450个测试。

**crash-me**可以提供的一种类型的信息的例子：如果想要使用Informix或DB2，不应该使用超过18个字符的列名。

**crash-me**程序和**MySQL**基准程序是独立于数据库的。通过观察它们是如何编写的，编可以知道必须为编写独立于数据库的应用程序做什么。基准本身可在**MySQL**源码分发的“sql-bench”目录下找到。它们用DBI数据库接口以Perl写成。使用DBI本身即可以解决部分移植性问题，因为它提供与数据库无关的的存取方法。

关于**crash-me**结果，访问[http://dev.mysql.com/tech-resources/crash-me.php](http://dev.mysql.com/tech-resources/crash-me.php" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)。到[http://dev.mysql.com/tech-resources/benchmarks/](http://dev.mysql.com/tech-resources/benchmarks/" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)看这个基准的结果。

如果你为数据库的独立性而努力，需要很好地了解每个SQL服务器的瓶颈。例如，**MySQL**在检索和更新MyISAM表记录方面很快，但是在同一个表上混合慢速读者和写者方面有一个问题。另一方面，当你试图访问最近更新了(直到它们被刷新到磁盘上)的行时，在Oracle中有一个很大的问题。事务数据库总的来说在从记录文件表中生成总结表方面不是很好，因为在这种情况下，行锁定几乎没有用。

为了使应用程序“确实”独立于数据库，需要定义一个容易扩展的接口，用它可操纵数据。因为C++在大多数系统上可以适用，使用数据库的一个C++ 类接口是有意义的。

如果你使用某个数据库特定的功能(例如**MySQL专用的**REPLACE语句)，应该为SQL服务器编码一个方法以实现同样的功能。尽管慢些，但确允许其它服务器执行同样的任务。

用**MySQL**，可以使用/\*! \*/语法把**MySQL**特定的关键词加到查询中。在/\*\*/中的代码将被其它大多数SQL服务器视为注释(并被忽略)。

如果高性能真的比准确性更重要，就像在一些web应用程序那样，一种可行的方法是创建一个应用层，缓存所有的结果以便得到更高的性能。通过只是让旧的结果在短时间后‘过期’，能保持缓存合理地刷新。这在极高负载的情况下是相当不错的，在此情况下，能动态地增加缓存并且设定较高的过期时限直到一切恢复正常。

在这种情况下，表创建信息应该包含缓存初始大小和表刷新频率等信息。

实施应用程序缓存的一种方法是使用MySQL查询缓存。启用查询缓存后，服务器可以确定是否可以重新使用查询结果。这样简化了你的应用程序。参见[5.13节，“MySQL查询高速缓冲”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "query-cache" \o "5.13. The MySQL Query Cache)。

### 7.1.3. 我们已将MySQL用在何处？

该节描述了Mysql的早期应用程序。

在MySQL最初开发期间，MySQL的功能适合大多数客户。MySQL为瑞典的一些最大的零售商处理数据仓库。

我们从所有商店得到所有红利卡交易的每周总结，并且我们期望为所有店主提供有用的信息以帮助他们得出他们的广告战如何影响他们的顾客。

数据是相当巨量的(大约每月7百万宗交易总结)并且我们保存4-10年来的数据需要呈现给用户。我们每周从顾客那里得到请求，他们想要“立刻”访问来自该数据的新报告。

我们通过每月将所有信息存储在压缩的“交易”表中来解决它。我们有一套简单的宏/脚本用来生成来自交易表的不同条件( 产品组、顾客id，商店...)的总结表。报告是由一个进行语法分析网页的小perl脚本动态生成的网页，在脚本中执行SQL语句并且插入结果。我们很想使用PHP或**mod\_perl**，但是那时它们还不可用。

对图形数据，我们用C语言编写了一个简单的工具，它能基于那些结果处理SQL查询结果并生成GIF图形。该工具也从分析Web网页的perl脚本中动态地执行。

在大多数情况下，一个新的报告通过简单地复制一个现有脚本并且修改其中的SQL查询来完成。在一些情况下，我们将需要把更多的列加到一个现有的总结表中或产生一个新的，但是这也相当简单，因为我们在磁盘上保存所有交易表。（目前我们大约有50G的交易表和200G的其它顾客数据)。

我们也让我们的顾客直接用ODBC访问总结表以便高级用户能自己用这些数据进行试验。

该系统工作得很好，我们可以毫无问题地用很适度的Sun Ultra SPARC工作站硬件(2x200MHz)来处理数据。该系统被逐步移植到了Linux中。

### 7.1.4. MySQL基准套件

本节应该包含**MySQL**基准套件(和**crash-me**)的技术描述，但是该描述还没写成。目前，你可以通过在MySQL源码分发中的“sql-bench”目录下的代码和结果了解基准套件是如何工作的。

通过基准用户可以了解一个给定的SQL实现在哪方面执行得很好或很糟糕。

注意，这个基准是单线程的，它可以测量操作执行的最小时间。我们计划将来在基准套件中添加多线程测试。

要使用基准套件，必须满足下面的要求：

·         基准套件随MySQL源码分发提供。可以从[http://dev.mysql.com/downloads/](http://dev.mysql.com/downloads/" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)下载分发，或者使用当前的开发源码树(参见[2.8.3节，“从开发源码树安装”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/installing.html" \l "installing-source-tree" \o "2.8.3. Installing from the Development Source Tree))。

·         基准脚本用Perl编写而成，使用Perl DBI模块访问数据库服务器，因此必须安装DBI。还需要为每个待测试的服务器提供服务器专用DBD驱动程序。例如，要测试MySQL、PostgreSQL和DB2，必须安装DBD::mysql、DBD::Pg和DBD::DB2模块。参见[2.13节，“Perl安装注意事项”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/installing.html" \l "perl-support" \o "2.13. Perl Installation Notes)。

获得MySQL源码分发后，可以在sql-bench目录找到基准套件。要运行基准测试，应构建MySQL，然后进入sql-bench目录并执行run-all-tests脚本：

shell> **cd sql-bench**

shell> **perl run-all-tests --server=*server\_name***

*server\_name*是一个支持的服务器。要获得所有选项和支持的服务器，调用命令：

shell> **perl run-all-tests --help**

**crash-me**脚本也位于sql-bench目录。**crash-me**尝试通过实际运行查询确定数据库支持的特性以及其功能和限制。例如，它确定：

·         支持什么列类型

·         支持多少索引

·         支持什么函数

·         查询可以多大

·         VARCHAR列可以多大

可以从[http://dev.mysql.com/tech-resources/crash-me.php](http://dev.mysql.com/tech-resources/crash-me.php" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)发现许多不同数据库服务器的**crash-me**的结果。关于基准测试结果的详细信息，访问[http://dev.mysql.com/tech-resources/benchmarks/](http://dev.mysql.com/tech-resources/benchmarks/" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)。

### 7.1.5. 使用自己的基准

一定要测试应用程序和数据库，以发现瓶颈在哪儿。通过修正它(或通过用一个“哑模块”代替瓶颈)，可以很容易地确定下一个瓶颈。即使你的应用程序的整体性能目前可以接受，至少应该对每个瓶颈做一个计划，如果某天确实需要更好的性能，应知道如何解决它。

关于一些可移植的基准程序的例子，参见MySQL基准套件。请参见[7.1.4节，“MySQL基准套件”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "mysql-benchmarks" \o "7.1.4. The MySQL Benchmark Suite)。可以利用这个套件的任何程序并且根据你的需要修改它。通过这样做，可以尝试不同的问题的解决方案并测试哪一个是最好的解决方案。

另一个免费基准套件是开放源码数据库基准套件，参见[http://osdb.sourceforge.net/](http://osdb.sourceforge.net/" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)。

在系统负载繁重时出现一些问题是很普遍的，并且很多客户已经与我们联系了，他们在生产系统中有一个(测试)系统并且有负载问题。大多数情况下，性能问题经证明是与基本数据库设计有关的问题(例如，表扫描在高负载时表现不好)或操作系统或库问题。如果系统已经不在生产系统中，它们大多数将**很**容易修正。

为了避免这样的问题，应该把工作重点放在在可能最坏的负载下测试你的整个应用程序。你可以使用Super Smack。该工具可以从[http://jeremy.zawodny.com/mysql/super-smack/](http://jeremy.zawodny.com/mysql/super-smack/" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)获得。正如它的名字所建议，它可以根据你的需要提供合理的系统，因此确保只用于你的开发系统。

## 7.2. 优化SELECT语句和其它查询

[7.2.1. EXPLAIN语法（获取SELECT相关信息）](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "explain)

[7.2.2. 估计查询性能](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "estimating-performance)

[7.2.3. SELECT查询的速度](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "select-speed)

[7.2.4. MySQL怎样优化WHERE子句](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "where-optimizations)

[7.2.5. 范围优化](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "range-optimization)

[7.2.6. 索引合并优化](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-optimization)

[7.2.7. MySQL如何优化IS NULL](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "is-null-optimization)

[7.2.8. MySQL如何优化DISTINCT](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "distinct-optimization)

[7.2.9. MySQL如何优化LEFT JOIN和RIGHT JOIN](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "left-join-optimization)

[7.2.10. MySQL如何优化嵌套Join](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "nested-joins)

[7.2.11. MySQL如何简化外部联合](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "outer-join-simplification)

[7.2.12. MySQL如何优化ORDER BY](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "order-by-optimization)

[7.2.13. MySQL如何优化GROUP BY](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "group-by-optimization)

[7.2.14. MySQL如何优化LIMIT](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "limit-optimization)

[7.2.15. 如何避免表扫描](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "how-to-avoid-table-scan)

[7.2.16. INSERT语句的速度](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "insert-speed)

[7.2.17. UPDATE语句的速度](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "update-speed)

[7.2.18. DELETE语句的速度](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "delete-speed)

[7.2.19. 其它优化技巧](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "tips)

首先，影响所有语句的一个因素是：你的许可设置得越复杂，所需要的开销越多。

执行GRANT语句时使用简单的许可，当客户执行语句时，可以使MySQL降低许可检查开销。例如，如果未授予任何表级或列级权限，服务器不需要检查tables\_priv和columns\_priv表的内容。同样地，如果不对任何 账户进行限制，服务器不需要对资源进行统计。如果查询量很高，可以花一些时间使用简化的授权结构来降低许可检查开销。

如果你的问题是与具体MySQL表达式或函数有关，可以使用**mysql**客户程序所带的BENCHMARK()函数执行定时测试。其语法为BENCHMARK(*loop\_count*,*expression*)。例如：

mysql> **SELECT BENCHMARK(1000000,1+1)；**

+------------------------+

| BENCHMARK(1000000,1+1) |

+------------------------+

|                      0 |

+------------------------+

1 row in set (0.32 sec)

上面结果在PentiumII 400MHz系统上获得。它显示**MySQL**在该系统上在0.32秒内可以执行1,000,000个简单的+表达式运算。

所有**MySQL**函数应该被高度优化，但是总有可能有一些例外。BENCHMARK()是一个找出是否查询有问题的优秀的工具。

### 7.2.1. EXPLAIN语法（获取SELECT相关信息）

EXPLAIN *tbl\_name*

或：

EXPLAIN [EXTENDED] SELECT *select\_options*

EXPLAIN语句可以用作DESCRIBE的一个同义词，或获得关于MySQL如何执行SELECT语句的信息：

·         EXPLAIN *tbl\_name*是DESCRIBE *tbl\_name*或SHOW COLUMNS FROM *tbl\_name*的一个同义词。

·         如果在SELECT语句前放上关键词EXPLAIN，**MySQL**将解释它如何处理SELECT，提供有关表如何联接和联接的次序。

该节解释EXPLAIN的第2个用法。

借助于EXPLAIN，可以知道什么时候必须为表加入索引以得到一个使用索引来寻找记录的更快的SELECT。

如果由于使用不正确的索引出现了问题，应运行ANALYZE TABLE更新表的统计（例如关键字集的势），这样会影响优化器进行的选择。参见[13.5.2.1节，“ANALYZE TABLE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "analyze-table" \o "13.5.2.1. ANALYZE TABLE Syntax)。

还可以知道优化器是否以一个最佳次序联接表。为了强制优化器让一个SELECT语句按照表命名顺序的联接次序，语句应以STRAIGHT\_JOIN而不只是SELECT开头。

EXPLAIN为用于SELECT语句中的每个表返回一行信息。表以它们在处理查询过程中将被MySQL读入的顺序被列出。**MySQL**用一遍扫描多次联接（single-sweep multi-join）的方式解决所有联接。这意味着**MySQL**从第一个表中读一行，然后找到在第二个表中的一个匹配行，然后在第3个表中等等。当所有的表处理完后，它输出选中的列并且返回表清单直到找到一个有更多的匹配行的表。从该表读入下一行并继续处理下一个表。

当使用EXTENDED关键字时，EXPLAIN产生附加信息，可以用SHOW WARNINGS浏览。该信息显示优化器限定SELECT语句中的表和列名，重写并且执行优化规则后SELECT语句是什么样子，并且还可能包括优化过程的其它注解。

EXPLAIN的每个输出行提供一个表的相关信息，并且每个行包括下面的列：

·         id

SELECT识别符。这是SELECT的查询序列号。

·         select\_type

SELECT类型，可以为以下任何一种：

o        SIMPLE

简单SELECT(不使用UNION或子查询)

o        PRIMARY

最外面的SELECT

o        UNION

UNION中的第二个或后面的SELECT语句

o        DEPENDENT UNION

UNION中的第二个或后面的SELECT语句，取决于外面的查询

o        UNION RESULT

UNION的结果。

o        SUBQUERY

子查询中的第一个SELECT

o        DEPENDENT SUBQUERY

子查询中的第一个SELECT，取决于外面的查询

o        DERIVED

导出表的SELECT(FROM子句的子查询)

·         table

输出的行所引用的表。

·         type

联接类型。下面给出各种联接类型，按照从最佳类型到最坏类型进行排序：

o        system

表仅有一行(=系统表)。这是const联接类型的一个特例。

o        const

表最多有一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被优化器剩余部分认为是常数。const表很快，因为它们只读取一次！

const用于用常数值比较PRIMARY KEY或UNIQUE索引的所有部分时。在下面的查询中，*tbl\_name*可以用于const表：

SELECT \* from *tbl\_name* WHERE *primary\_key*=1；

SELECT \* from *tbl\_name*

WHERE *primary\_key\_part1*=1和 *primary\_key\_part2*=2；

o        eq\_ref

对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了const类型。它用在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY。

eq\_ref可以用于使用= 操作符比较的带索引的列。比较值可以为常量或一个使用在该表前面所读取的表的列的表达式。

在下面的例子中，MySQL可以使用eq\_ref联接来处理*ref\_tables*：

SELECT \* FROM *ref\_table*,*other\_table*

  WHERE *ref\_table*.*key\_column*=*other\_table*.*column*;

SELECT \* FROM *ref\_table*,*other\_table*

  WHERE *ref\_table*.*key\_column\_part1*=*other\_table*.*column*

    AND *ref\_table*.*key\_column\_part2*=1;

o        ref

对于每个来自于前面的表的行组合，所有有匹配索引值的行将从这张表中读取。如果联接只使用键的最左边的前缀，或如果键不是UNIQUE或PRIMARY KEY（换句话说，如果联接不能基于关键字选择单个行的话），则使用ref。如果使用的键仅仅匹配少量行，该联接类型是不错的。

ref可以用于使用=或<=>操作符的带索引的列。

在下面的例子中，MySQL可以使用ref联接来处理*ref\_tables*：

SELECT \* FROM *ref\_table* WHERE *key\_column*=*expr*;

SELECT \* FROM *ref\_table*,*other\_table*

  WHERE *ref\_table*.*key\_column*=*other\_table*.*column*;

SELECT \* FROM *ref\_table*,*other\_table*

  WHERE *ref\_table*.*key\_column\_part1*=*other\_table*.*column*

    AND *ref\_table*.*key\_column\_part2*=1;

o        ref\_or\_null

该联接类型如同ref，但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。

在下面的例子中，MySQL可以使用ref\_or\_null联接来处理*ref\_tables*：

SELECT \* FROM *ref\_table*

WHERE *key\_column*=*expr* OR *key\_column* IS NULL;

参见[7.2.7节，“MySQL如何优化IS NULL”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "is-null-optimization" \o "7.2.7. How MySQL Optimizes IS NULL)。

o        index\_merge

该联接类型表示使用了索引合并优化方法。在这种情况下，key列包含了使用的索引的清单，key\_len包含了使用的索引的最长的关键元素。详细信息参见[7.2.6节，“索引合并优化”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-optimization" \o "7.2.6. Index Merge Optimization)。

o        unique\_subquery

该类型替换了下面形式的IN子查询的ref：

*value* IN (SELECT *primary\_key* FROM *single\_table* WHERE *some\_expr*)

unique\_subquery是一个索引查找函数，可以完全替换子查询，效率更高。

o        index\_subquery

该联接类型类似于unique\_subquery。可以替换IN子查询，但只适合下列形式的子查询中的非唯一索引：

*value* IN (SELECT *key\_column* FROM *single\_table* WHERE *some\_expr*)

o        range

只检索给定范围的行，使用一个索引来选择行。key列显示使用了哪个索引。key\_len包含所使用索引的最长关键元素。在该类型中ref列为NULL。

当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN或者IN操作符，用常量比较关键字列时，可以使用range：

SELECT \* FROM *tbl\_name*

WHERE *key\_column* = 10;

SELECT \* FROM *tbl\_name*

WHERE *key\_column* BETWEEN 10 and 20;

SELECT \* FROM *tbl\_name*

WHERE *key\_column* IN (10,20,30);

SELECT \* FROM *tbl\_name*

WHERE *key\_part1*= 10 AND *key\_part2 IN (10,20,30);*

o        index

该联接类型与ALL相同，除了只有索引树被扫描。这通常比ALL快，因为索引文件通常比数据文件小。

当查询只使用作为单索引一部分的列时，MySQL可以使用该联接类型。

o        ALL

对于每个来自于先前的表的行组合，进行完整的表扫描。如果表是第一个没标记const的表，这通常不好，并且通常在它情况下***很***差。通常可以增加更多的索引而不要使用ALL，使得行能基于前面的表中的常数值或列值被检索出。

·         possible\_keys

possible\_keys列指出**MySQL**能使用哪个索引在该表中找到行。注意，该列完全独立于EXPLAIN输出所示的表的次序。这意味着在possible\_keys中的某些键实际上不能按生成的表次序使用。

如果该列是NULL，则没有相关的索引。在这种情况下，可以通过检查WHERE子句看是否它引用某些列或适合索引的列来提高你的查询性能。如果是这样，创造一个适当的索引并且再次用EXPLAIN检查查询。参见[13.1.2节，“ALTER TABLE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "alter-table" \o "13.1.2. ALTER TABLE Syntax)。

为了看清一张表有什么索引，使用SHOW INDEX FROM *tbl\_name*。

·         key

key列显示**MySQL**实际决定使用的键（索引）。如果没有选择索引，键是NULL。要想强制MySQL使用或忽视possible\_keys列中的索引，在查询中使用FORCE INDEX、USE INDEX或者IGNORE INDEX。参见[13.2.7节，“SELECT语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "select" \o "13.2.7. SELECT Syntax)。

对于MyISAM和BDB表，运行ANALYZE TABLE可以帮助优化器选择更好的索引。对于MyISAM表，可以使用**myisamchk --analyze**。参见[13.5.2.1节，“ANALYZE TABLE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "analyze-table" \o "13.5.2.1. ANALYZE TABLE Syntax)和[5.9.4节，“表维护和崩溃恢复”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "table-maintenance" \o "5.9.4. Table Maintenance and Crash Recovery)。

·         key\_len

key\_len列显示**MySQL**决定使用的键长度。如果键是NULL，则长度为NULL。注意通过key\_len值我们可以确定MySQL将实际使用一个多部关键字的几个部分。

·         ref

ref列显示使用哪个列或常数与key一起从表中选择行。

·         rows

rows列显示**MySQL**认为它执行查询时必须检查的行数。

·         Extra

该列包含MySQL解决查询的详细信息。下面解释了该列可以显示的不同的文本字符串：

o        Distinct

MySQL发现第1个匹配行后，停止为当前的行组合搜索更多的行。

o        Not exists

MySQL能够对查询进行LEFT JOIN优化，发现1个匹配LEFT JOIN标准的行后，不再为前面的的行组合在该表内检查更多的行。

下面是一个可以这样优化的查询类型的例子：

SELECT \* 从t1 LEFT JOIN t2 ON t1.id=t2.id

  WHERE t2.id IS NULL；

假定t2.id定义为NOT NULL。在这种情况下，MySQL使用t1.id的值扫描t1并查找t2中的行。如果MySQL在t2中发现一个匹配的行，它知道t2.id绝不会为NULL，并且不再扫描t2内有相同的id值的行。换句话说，对于t1的每个行，MySQL只需要在t2中查找一次，无论t2内实际有多少匹配的行。

o        range checked for each record (index map: #)

MySQL没有发现好的可以使用的索引，但发现如果来自前面的表的列值已知，可能部分索引可以使用。对前面的表的每个行组合，MySQL检查是否可以使用range或index\_merge访问方法来索取行。关于适用性标准的描述参见[7.2.5节，“范围优化”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "range-optimization" \o "7.2.5. Range Optimization)和[7.2.6节，“索引合并优化”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-optimization" \o "7.2.6. Index Merge Optimization)，不同的是前面表的所有列值已知并且认为是常量。

这并不很快，但比执行没有索引的联接要快得多。

o        Using filesort

MySQL需要额外的一次传递，以找出如何按排序顺序检索行。通过根据联接类型浏览所有行并为所有匹配WHERE子句的行保存排序关键字和行的指针来完成排序。然后关键字被排序，并按排序顺序检索行。参见[7.2.12节，“MySQL如何优化ORDER BY”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "order-by-optimization" \o "7.2.12. How MySQL Optimizes ORDER BY)。

o        Using index

从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。当查询只使用作为单一索引一部分的列时，可以使用该策略。

o        Using temporary

为了解决查询，MySQL需要创建一个临时表来容纳结果。典型情况如查询包含可以按不同情况列出列的GROUP BY和ORDER BY子句时。

o        Using where

WHERE子句用于限制哪一个行匹配下一个表或发送到客户。除非你专门从表中索取或检查所有行，如果Extra值不为Using where并且表联接类型为ALL或index，查询可能会有一些错误。

如果想要使查询尽可能快，应找出Using filesort 和Using temporary的Extra值。

o        Using sort\_union(...), Using union(...), Using intersect(...)

这些函数说明如何为index\_merge联接类型合并索引扫描。详细信息参见[7.2.6节，“索引合并优化”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-optimization" \o "7.2.6. Index Merge Optimization)。

o        Using index for group-by

类似于访问表的Using index方式，Using index for group-by表示MySQL发现了一个索引，可以用来查询GROUP BY或DISTINCT查询的所有列，而不要额外搜索硬盘访问实际的表。并且，按最有效的方式使用索引，以便对于每个组，只读取少量索引条目。详情参见[7.2.13节，“MySQL如何优化GROUP BY”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "group-by-optimization" \o "7.2.13. How MySQL Optimizes GROUP BY)。

通过相乘EXPLAIN输出的rows列的所有值，你能得到一个关于一个联接如何的提示。这应该粗略地告诉你**MySQL**必须检查多少行以执行查询。当你使用max\_join\_size变量限制查询时，也用这个乘积来确定执行哪个多表SELECT语句。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

下列例子显示出一个多表JOIN如何能使用EXPLAIN提供的信息逐步被优化。

假定你有下面所示的SELECT语句，计划使用EXPLAIN来检查它：

EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,

               tt.ProjectReference, tt.EstimatedShipDate,

               tt.ActualShipDate, tt.ClientID,

               tt.ServiceCodes, tt.RepetitiveID,

               tt.CurrentProcess, tt.CurrentDPPerson,

               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,

               et\_1.COUNTRY, do.CUSTNAME

        FROM tt, et, et AS et\_1, do

        WHERE tt.SubmitTime IS NULL

          AND tt.ActualPC = et.EMPLOYID

          AND tt.AssignedPC = et\_1.EMPLOYID

          AND tt.ClientID = do.CUSTNMBR;

对于这个例子，假定：

·         被比较的列声明如下：

|  |  |  |
| --- | --- | --- |
| **表** | **列** | **列类型** |
| tt | ActualPC | CHAR(10) |
| tt | AssignedPC | CHAR(10) |
| tt | ClientID | CHAR(10) |
| et | EMPLOYID | CHAR(15) |
| do | CUSTNMBR | CHAR(15) |

·         表有下面的索引：

|  |  |
| --- | --- |
| **表** | **索引** |
| tt | ActualPC |
| tt | AssignedPC |
| tt | ClientID |
| et | EMPLOYID(主键) |
| do | CUSTNMBR(主键) |

·         tt.ActualPC值不是均匀分布的。

开始，在进行优化前，EXPLAIN语句产生下列信息：

table type possible\_keys key  key\_len ref  rows  Extra

et    ALL  PRIMARY       NULL NULL    NULL 74

do    ALL  PRIMARY       NULL NULL    NULL 2135

et\_1  ALL  PRIMARY       NULL NULL    NULL 74

tt    ALL  AssignedPC,   NULL NULL    NULL 3872

           ClientID,

           ActualPC

      range checked for each record (key map: 35)

因为type对每张表是ALL，这个输出显示**MySQL**正在对所有表产生一个笛卡尔乘积；即每一个行的组合！这将花相当长的时间，因为必须检查每张表的行数的乘积！对于一个实例，这是74 \* 2135 \* 74 \* 3872 = 45,268,558,720行。如果表更大，你只能想象它将花多长时间……

这里的一个问题是**MySQL**能更高效地在声明具有相同类型和尺寸的列上使用索引。在本文中，VARCHAR和CHAR是相同的，除非它们声明为不同的长度。因为tt.ActualPC被声明为CHAR(10)并且et.EMPLOYID被声明为CHAR(15)，长度不匹配。

为了修正在列长度上的不同，使用ALTER TABLE将ActualPC的长度从10个字符变为15个字符：

mysql> **ALTER TABLE tt MODIFY ActualPC VARCHAR(15);**

现在tt.ActualPC和et.EMPLOYID都是VARCHAR(15)，再执行EXPLAIN语句产生这个结果：

table type   possible\_keys key     key\_len ref         rows    Extra

tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using

             ClientID,                                         where

             ActualPC

do    ALL    PRIMARY       NULL    NULL    NULL        2135

      range checked for each record (key map: 1)

et\_1  ALL    PRIMARY       NULL    NULL    NULL        74

      range checked for each record (key map: 1)

et    eq\_ref PRIMARY       PRIMARY 15      tt.ActualPC 1

这不是完美的，但是好一些了：rows值的乘积少了一个因子74。这个版本在几秒内执行完。

第2种方法能消除tt.AssignedPC = et\_1.EMPLOYID和tt.ClientID = do.CUSTNMBR比较的列的长度失配问题：

mysql> **ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),**

    ->                **MODIFY ClientID   VARCHAR(15);**

EXPLAIN产生的输出显示在下面：

table type   possible\_keys key      key\_len ref           rows Extra

et    ALL    PRIMARY       NULL     NULL    NULL          74

tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using

             ClientID,                                         where

             ActualPC

et\_1  eq\_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1

do    eq\_ref PRIMARY       PRIMARY  15      tt.ClientID   1

这几乎很好了。

剩下的问题是，默认情况，**MySQL**假设在tt.ActualPC列的值是均匀分布的，并且对tt表不是这样。幸好，很容易告诉**MySQL**来分析关键字分布：

mysql> **ANALYZE TABLE tt；**

现在联接是“完美”的了，而且EXPLAIN产生这个结果：

table type   possible\_keys key     key\_len ref           rows Extra

tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using

             ClientID,                                        where

             ActualPC

et    eq\_ref PRIMARY       PRIMARY 15      tt.ActualPC   1

et\_1  eq\_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1

do    eq\_ref PRIMARY       PRIMARY 15      tt.ClientID   1

注意在从EXPLAIN输出的rows列是一个来自**MySQL**联接优化器的“教育猜测”。你应该检查数字是否接近事实。如果不是，可以通过在SELECT语句里面使用STRAIGHT\_JOIN并且试着在FROM子句以不同的次序列出表，可能得到更好的性能。

### 7.2.2. 估计查询性能

在大多数情况下，可以通过计算磁盘搜索来估计性能。对小的表，通常能在1次磁盘搜索中找到行(因为索引可能被缓存)。对更大的表，可以使用B－树索引进行估计，将需要log(*row\_count*)/log(*index\_block\_length*/3 \* 2/(*index\_length* + *data\_pointer\_length*))+1次搜索才能找到行。

在**MySQL**中，索引块通常是1024个字节，数据指针通常是4个字节，这对于有一个长度为3(中等整数)的索引的500,000行的表，通过公式可以计算出log(500,000)/log(1024/3\*2/(3+4))+1= 4次搜索。

上面的索引需要大约500,000 \* 7 \* 3/2 = 5.2MB，(假设典型情况下索引缓存区填充率为2/3)，可以将大部分索引保存在内存中，仅需要1-2调用从OS读数据来找出行。

然而对于写，将需要4次搜索请求(如上)来找到在哪儿存放新索引，并且通常需要2次搜索来更新这个索引并且写入行。

注意，上述讨论并不意味着应用程序的性能将缓慢地以log*N* 退化！当表格变得更大时，所有内容缓存到OS或SQL服务器后，将仅仅或多或少地更慢。在数据变得太大不能缓存后，将逐渐变得更慢，直到应用程序只能进行磁盘搜索(以log*N*增加)。为了避免这个问题，随数据增加而增加 键高速缓冲区大小。对于MyISAM表, 由key\_buffer\_size系统变量控制 键高速缓冲区大小。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

### 7.2.3. SELECT查询的速度

总的来说，要想使一个较慢速SELECT ... WHERE更快，应首先检查是否能增加一个索引。不同表之间的引用通常通过索引来完成。你可以使用EXPLAIN语句来确定SELECT语句使用哪些索引。参见[7.4.5节，“MySQL如何使用索引”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "mysql-indexes" \o "7.4.5. How MySQL Uses Indexes)和[7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "explain" \o "7.2.1. EXPLAIN Syntax (Get Information About a SELECT))。

下面是一些加速对MyISAM表的查询的一般建议：

·         为了帮助**MySQL**更好地优化查询，在一个装载数据后的表上运行ANALYZE TABLE或**myisamchk --analyze**。这样为每一个索引更新指出有相同值的行的平均行数的值（当然，如果只有一个索引，这总是1。）MySQL使用该方法来决定当你联接两个基于非常量表达式的表时选择哪个索引。你可以使用SHOW INDEX FROM *tbl\_name*并检查Cardinality值来检查表分析结果。**myisamchk --description --verbose**可以显示索引分布信息。

·         要想根据一个索引排序一个索引和数据，使用**myisamchk --sort-index --sort-records=1**（如果你想要在索引1上排序）。如果只有一个索引，想要根据该索引的次序读取所有的记录，这是使查询更快的一个好方法。但是请注意，第一次对一个大表按照这种方法排序时将花很长时间！

### 7.2.4. MySQL怎样优化WHERE子句

该节讨论为处理WHERE子句而进行的优化。例子中使用了SELECT语句，但相同的优化也适用DELETE和UPDATE语句中的WHERE子句。

请注意对MySQL优化器的工作在不断进行中，因此该节并不完善。MySQL执行了大量的优化，本文中所列的并不详尽。

下面列出了MySQL执行的部分优化：

·         去除不必要的括号：

·                        ((a AND b) AND c OR (((a AND b) AND (c AND d))))

·                -> (a AND b AND c) OR (a AND b AND c AND d)

·         常量重叠：

·                   (a<b AND b=c) AND a=5

·                -> b>5 AND b=c AND a=5

·         去除常量条件(由于常量重叠需要)：

·                   (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)

·                -> B=5 OR B=6

·         索引使用的常数表达式仅计算一次。

* 对于MyISAM和HEAP表，在一个单个表上的没有一个WHERE的COUNT(\*)直接从表中检索信息。当仅使用一个表时，对NOT NULL表达式也这样做。
* 无效常数表达式的早期检测。**MySQL**快速检测某些SELECT语句是不可能的并且不返回行。
* 如果不使用GROUP BY或分组函数(COUNT()、MIN()……)，HAVING与WHERE合并。
* 对于联接内的每个表，构造一个更简单的WHERE以便更快地对表进行WHERE计算并且也尽快跳过记录。
* 所有常数的表在查询中比其它表先读出。常数表为：
  + 空表或只有1行的表。
  + 与在一个PRIMARY KEY或UNIQUE索引的WHERE子句一起使用的表，这里所有的索引部分使用常数表达式并且索引部分被定义为NOT NULL。

下列的所有表用作常数表：

mysql> SELECT \* FROM t WHERE primary\_key=1;

mysql> SELECT \* FROM t1,t2

           WHERE t1.primary\_key=1 AND t2.primary\_key=t1.id;

* 尝试所有可能性便可以找到表联接的最好联接组合。如果所有在ORDER BY和GROUP BY的列来自同一个表，那么当联接时，该表首先被选中。
* 如果有一个ORDER BY子句和不同的GROUP BY子句，或如果ORDER BY或GROUP BY包含联接队列中的第一个表之外的其它表的列，则创建一个临时表。
* 如果使用SQL\_SMALL\_RESULT，**MySQL**使用内存中的一个临时表。
* 每个表的索引被查询，并且使用最好的索引，除非优化器认为使用表扫描更有效。是否使用扫描取决于是否最好的索引跨越超过30%的表。优化器更加复杂，其估计基于其它因素，例如表大小、行数和I/O块大小，因此固定比例不再决定选择使用索引还是扫描。
* 在一些情况下，**MySQL**能从索引中读出行，甚至不查询数据文件。如果索引使用的所有列是数值类，那么只使用索引树来进行查询。
* 输出每个记录前，跳过不匹配HAVING子句的行。

下面是一些快速查询的例子：

SELECT COUNT(\*) FROM *tbl\_name*;

SELECT MIN(*key\_part1*),MAX(*key\_part1*) FROM *tbl\_name*;

SELECT MAX(*key\_part2*) FROM *tbl\_name*

    WHERE *key\_part1*=*constant*;

SELECT ... FROM *tbl\_name*

    ORDER BY *key\_part1*,*key\_part2*,... LIMIT 10;

SELECT ... FROM *tbl\_name*

    ORDER BY *key\_part1* DESC, *key\_part2* DESC, ... LIMIT 10;

下列查询仅使用索引树就可以解决(假设索引的列为数值型)：

SELECT *key\_part1*,*key\_part2* FROM *tbl\_name* WHERE *key\_part1*=*val*;

SELECT COUNT(\*) FROM *tbl\_name*

    WHERE *key\_part1*=*val1* AND *key\_part2*=*val2*;

SELECT *key\_part2* FROM *tbl\_name* GROUP BY *key\_part1*;

下列查询使用索引按排序顺序检索行，不用另外的排序：

SELECT ... FROM *tbl\_name*

    ORDER BY *key\_part1*,*key\_part2*,... ;

SELECT ... FROM *tbl\_name*

    ORDER BY *key\_part1* DESC, *key\_part2* DESC, ... ;

### 7.2.5. 范围优化

[7.2.5.1. 单元素索引的范围访问方法](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "range-access-single-part)

[7.2.5.2. 多元素索引的范围访问方法](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "range-access-multi-part)

range访问方法使用单一索引来搜索包含在一个或几个索引值距离内的表记录的子集。可以用于单部分或多元素索引。后面的章节将详细描述如何从WHERE子句提取区间。

#### 7.2.5.1. 单元素索引的范围访问方法

对于单元素索引，可以用WHERE子句中的相应条件很方便地表示索引值区间，因此我们称为范围条件而不是“区间”。

单元素索引范围条件的定义如下：

·         对于BTREE和HASH索引，当使用=、<=>、IN、IS NULL或者IS NOT NULL操作符时，关键元素与常量值的比较关系对应一个范围条件。

·         对于BTREE索引，当使用>、<、>=、<=、BETWEEN、!=或者<>，或者LIKE '*pattern*'(其中 '*pattern*'不以通配符开始)操作符时，关键元素与常量值的比较关系对应一个范围条件。

·         对于所有类型的索引，多个范围条件结合OR或AND则产生一个范围条件。

前面描述的“常量值”系指：

·         查询字符串中的常量

·         同一联接中的const或system表中的列

·         无关联子查询的结果

·         完全从前面类型的子表达式组成的表达式

下面是一些WHERE子句中有范围条件的查询的例子：

SELECT \* FROM t1

    WHERE *key\_col* > 1

    AND *key\_col* < 10;

SELECT \* FROM t1

    WHERE *key\_col* = 1

    OR *key\_col* IN (15,18,20);

SELECT \* FROM t1

    WHERE *key\_col* LIKE 'ab%'

    OR *key\_col* BETWEEN 'bar' AND 'foo';

请注意在常量传播阶段部分非常量值可以转换为常数。

MySQL尝试为每个可能的索引从WHERE子句提取范围条件。在提取过程中，不能用于构成范围条件的条件被放弃，产生重叠范围的条件组合到一起，并且产生空范围的条件被删除。

例如，考虑下面的语句，其中key1是有索引的列，nonkey没有索引：

SELECT \* FROM t1 WHERE

   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR

   (key1 < 'bar' AND nonkey = 4) OR

   (key1 < 'uux' AND key1 > 'z');

key1的提取过程如下：

1.    用原始WHERE子句开始：

2.    (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR

3.     (key1 < 'bar' AND nonkey = 4) OR

4.     (key1 < 'uux' AND key1 > 'z')

5.    删除nonkey = 4和key1 LIKE '%b'，因为它们不能用于范围扫描。删除它们的正确途径是用TRUE替换它们，以便进行范围扫描时不会丢失匹配的记录。用TRUE替换它们后，可以得到：

6.            (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR

7.            (key1 < 'bar' AND TRUE) OR

8.            (key1 < 'uux' AND key1 > 'z')

9.    取消总是为true或false的条件：

·         (key1 LIKE 'abcde%' OR TRUE)总是true

·         (key1 < 'uux' AND key1 > 'z')总是false

用常量替换这些条件，我们得到：

(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)

删除不必要的TRUE和FALSE常量，我们得到

(key1 < 'abc') OR (key1 < 'bar')

10.将重叠区间组合成一个产生用于范围扫描的最终条件：

11.        (key1 < 'bar')

总的来说(如前面的例子所述)，用于范围扫描的条件比WHERE子句限制少。MySQL再执行检查以过滤掉满足范围条件但不完全满足WHERE子句的行。

范围条件提取算法可以处理嵌套的任意深度的AND/OR结构，并且其输出不依赖条件在WHERE子句中出现的顺序。

#### 7.2.5.2. 多元素索引的范围访问方法

多元素索引的范围条件是单元素索引的范围条件的扩展。多元素索引的范围条件将索引记录限制到一个或几个关键元组内。使用索引的顺序，通过一系列关键元组来定义关键元组区间。

例如，考虑定义为key1(*key\_part1*, *key\_part2*, *key\_part3*)的多元素索引，以及下面的按关键字顺序所列的关键元组：

*key\_part1*  *key\_part2*  *key\_part3*

  NULL       1          'abc'

  NULL       1          'xyz'

  NULL       2          'foo'

   1         1          'abc'

   1         1          'xyz'

   1         2          'abc'

   2         1          'aaa'

条件*key\_part1* = 1定义了下面的范围：

(1，-inf，-inf) <= (*key\_part1*，*key\_part2*，*key\_part3*) < (1，+inf，+inf)

范围包括前面数据集中的第4、5和6个元组，可以用于范围访问方法。

通过对比，条件*key\_part3* = 'abc'不定义单一的区间，不能用于范围访问方法。

下面更加详细地描述了范围条件如何用于多元素索引中。

·         对于HASH索引，可以使用包含相同值的每个区间。这说明区间只能由下面形式的条件产生：

·                     *key\_part1* *cmp* *const1*

·                 AND *key\_part2* *cmp* *const2*

·                 AND ...

·                AND *key\_partN* *cmp* *constN*;

这里，*const1*，*const2*，...为常量，*cmp*是=、<=>或者IS NULL比较操作符之一，条件包括所有索引部分。(也就是说，有*N* 个条件，每一个对应*N*-元素索引的每个部分）。

关于常量的定义，参见[7.2.5.1节，“单元素索引的范围访问方法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "range-access-single-part" \o "7.2.5.1. Range Access Method for Single-Part Indexes)。

例如，下面为三元素HASH索引的范围条件：

*key\_part1* = 1 AND *key\_part2* IS NULL AND *key\_part3* = 'foo'

·         对于BTREE索引，区间可以对结合AND的条件有用，其中每个条件用一个常量值通过=、<=>、IS NULL、>、<、>=、<=、!=、<>、BETWEEN或者LIKE '*pattern*' (其中'*pattern*'不以通配符开头)比较一个关键元素。区间可以足够长以确定一个包含所有匹配条件(或如果使用<>或!=，为两个区间)的记录的单一的关键元组。例如，对于条件：

·                  *key\_part1* = 'foo' AND *key\_part2* >= 10 AND *key\_part3* > 10

单一区间为：

('foo'，10，10)

   < (*key\_part1*，*key\_part2*，*key\_part3*)

      < ('foo'，+inf，+inf)

创建的区间可以比原条件包含更多的记录。例如，前面的区间包括值('foo'，11，0)，不满足原条件。

·         如果包含区间内的一系列记录的条件结合使用OR，则形成包括一系列包含在区间并集的记录的一个条件。如果条件结合使用了AND，则形成包括一系列包含在区间交集内的记录的一个条件。例如，对于两部分索引的条件：

·                (*key\_part1* = 1 AND *key\_part2* < 2)

·                OR (*key\_part1* > 5)

区间为：

(1, -inf) < (*key\_part1*, *key\_part2*) < (1, 2)

(5, -inf) < (*key\_part1*, *key\_part2*)

在该例子中，第1行的区间左侧的约束使用了一个关键元素，右侧约束使用了两个关键元素。第2行的区间只使用了一个关键元素。EXPLAIN输出的key\_len列表示所使用关键字前缀的最大长度。

在某些情况中，key\_len可以表示使用的关键元素，但可能不是你所期望的。假定*key\_part1*和*key\_part2*可以为NULL。则key\_len列显示下面条件的两个关键元素的长度：

*key\_part1* >= 1 AND *key\_part2* < 2

但实际上，该条件可以变换为：

*key\_part1* >= 1 AND *key\_part2* IS NOT NULL

[7.2.5.1节，“单元素索引的范围访问方法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "range-access-single-part" \o "7.2.5.1. Range Access Method for Single-Part Indexes)描述了如何进行优化以结合或删除单元素索引范围条件的区间。多元素索引范围条件的区间的步骤类似。

### 7.2.6. 索引合并优化

[7.2.6.1. 索引合并交集访问算法](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-intersection)

[7.2.6.2. 索引合并并集访问算法](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-union)

[7.2.6.3. 索引合并排序并集访问算法](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-merge-sort-union)

索引合并方法用于通过range扫描搜索行并将结果合成一个。合并会产生并集、交集或者正在进行的扫描的交集的并集。

在EXPLAIN输出中，该方法表现为type列内的index\_merge。在这种情况下，key列包含一列使用的索引，key\_len包含这些索引的最长的关键元素。

例如：

SELECT \* FROM *tbl\_name* WHERE *key\_part1* = 10 OR *key\_part2* = 20;

SELECT \* FROM *tbl\_name*

    WHERE (*key\_part1* = 10 OR *key\_part2* = 20) AND *non\_key\_part*=30;

SELECT \* FROM t1, t2

    WHERE (t1.*key1* IN (1,2) OR t1.*key2* LIKE '*value*%')

    AND t2.*key1*=t1.*some\_col*;

SELECT \* FROM t1, t2

    WHERE t1.*key1*=1

    AND (t2.*key1*=t1.*some\_col* OR t2.*key2*=t1.*some\_col2*);

索引合并方法有几种访问算法 (参见EXPLAIN输出的Extra字段)：

·         交集

·         联合

·         排序并集

后面几节更加详细地描述了这些方法。

**注释：**索引合并优化算法具有以下几个已知缺陷：

·         如果可以对某些关键字进行范围扫描，则不考虑索引合并。例如，下面的查询：

·                SELECT \* FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

对于该查询，可以有两个方案：

1.    使用(goodkey1 < 10 OR goodkey2 < 20)条件进行索引合并扫描。

2.    使用badkey < 30条件进行范围扫描。

然而，优化器只考虑第2个方案。如果这不是你想要的，你可以通过使用IGNORE INDEX或FORCE INDEX让优化器考虑index\_merge。下面的查询使用索引合并执行：

SELECT \* FROM t1 FORCE INDEX(goodkey1,goodkey2)

WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

SELECT \* FROM t1 IGNORE INDEX(badkey)

WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

·         如果查询有一个复杂的WHERE子句，有较深的AND/OR嵌套关系，MySQL不选择该优选方案，通过下面的识别法则尝试分布各条件：

·                (*x* AND *y*) OR *z* = (*x* OR *z*) AND (*y* OR *z*)

·                (*x* OR *y*) AND *z* = (*x* AND *z*) OR (*y* AND *z*)

index\_merge访问方法的不同变量之间的选择和其它访问方法基于各适用选项的成本估计。

#### 7.2.6.1. 索引合并交集访问算法

该访问算法可以用于当WHERE子句结合AND被转换为不同的关键字的几个范围条件，每个条件为下面之一：

·         以这种形式，即索引有确切的*N*部分(即包括了所有索引部分)：

·                *key\_part1*=*const1* AND *key\_part2*=*const2* ... AND *key\_partN*=*constN*

·         任何InnoDB或BDB表的主键的范围条件。

下面是一些例子：

SELECT \* FROM *innodb\_table* WHERE *primary\_key* < 10 AND *key\_col1*=20;

SELECT \* FROM *tbl\_name*

WHERE (*key1\_part1*=1 AND *key1\_part2*=2) AND *key2*=2;

索引合并交集算法同时对所有使用的索引进行扫描，并产生从合并的索引扫描接收的行序列的交集。

如果使用的索引包括查询中使用的所有列，所有表记录均不搜索，并且在这种情况下EXPLAIN的输出包含Extra字段中的Using index。下面是一个此类查询的例子：

SELECT COUNT(\*) FROM t1 WHERE key1=1 AND key2=1;

如果使用的索引未包括查询中使用的所有列，只有满足所有使用的关键字的范围条件才搜索所有记录。

如果某个合并条件是InnoDB或BDB表的主键的一个条件，不用于记录查询，但用于过滤使用其它条件搜索的记录。

#### 7.2.6.2. 索引合并并集访问算法

该算法的适用标准类似于索引合并方法交集算法的标准。算法可以用于当WHERE子句结合OR被转换为不同的关键字的几个范围条件的时候，每个条件为下面之一：

·         以这种形式，即索引有确切的*N*部分(即包括了所有索引部分)：

·                *key\_part1*=*const1* AND *key\_part2*=*const2* ... AND *key\_partN*=*constN*

·         任何InnoDB或BDB表的主键的范围条件。

·         索引合并方法交集算法适用的一个条件。

下面是一些例子：

SELECT \* FROM t1 WHERE *key1*=1 OR *key2*=2 OR *key3*=3;

SELECT \* FROM *innodb\_table* WHERE (*key1*=1 AND *key2*=2) OR

  (*key3*='foo' AND *key4*='bar') AND *key5*=5;

#### 7.2.6.3. 索引合并排序并集访问算法

该访问算法可以用于当WHERE子句结合OR被转换为不同的关键字的几个范围条件，但索引合并方法联合算法并不适用的时候。

下面是一些例子：

SELECT \* FROM *tbl\_name* WHERE *key\_col1* < 10 OR *key\_col2* < 20;

SELECT \* FROM *tbl\_name*

     WHERE (*key\_col1* > 10 OR *key\_col2* = 20) AND *nonkey\_col*=30;

排序联合算法和联合算法的区别是排序联合算法必须先索取所有记录的行ID，然后在返回记录前对它们进行排序。

### 7.2.7. MySQL如何优化IS NULL

MySQL可以对可以结合*col\_name* = *constant\_value*使用的*col\_name* IS NULL进行相同的优化。例如，MySQL可以使用索引和范围用IS NULL搜索NULL。

SELECT \* FROM *tbl\_name* WHERE *key\_col* IS NULL;

SELECT \* FROM *tbl\_name* WHERE *key\_col* <=> NULL;

SELECT \* FROM *tbl\_name*

    WHERE *key\_col*=*const1* OR *key\_col*=*const2* OR *key\_col* IS NULL;

如果WHERE子句包括声明为NOT NULL的列的*col\_name* IS NULL条件，表达式则优化。当列会产生NULL时，不会进行优化；例如，如果来自LEFT JOIN右侧的表。

MySQL也可以优化组合*col\_name* = *expr* AND *col\_name* IS NULL，这是解决子查询的一种常用形式。当使用优化时EXPLAIN显示ref\_or\_null。

该优化可以为任何关键元素处理IS NULL。

下面是一些优化的查询例子，假定表t2的列a和b有一个索引：

SELECT \* FROM t1 WHERE t1.a=*expr* OR t1.a IS NULL;

SELECT \* FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT \* FROM t1, t2

    WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT \* FROM t1, t2

    WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT \* FROM t1, t2

    WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)

    OR (t1.a=t2.a AND t2.a IS NULL AND ...);

ref\_or\_null首先读取参考关键字，然后单独搜索NULL关键字的行。

请注意该优化只可以处理一个IS NULL。在后面的查询中，MySQL只对表达式(t1.a=t2.a AND t2.a IS NULL)使用关键字查询，不能使用b的关键元素：

SELECT \* FROM t1, t2

     WHERE (t1.a=t2.a AND t2.a IS NULL)

     OR (t1.b=t2.b AND t2.b IS NULL);

### 7.2.8. MySQL如何优化DISTINCT

在许多情况下结合ORDER BY的DISTINCT需要一个临时表。

请注意因为DISTINCT可能使用GROUP BY，必须清楚MySQL如何使用所选定列的一部分的ORDER BY或HAVING子句中的列。参见[12.10.3节，“具有隐含字段的GROUP BY”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/functions.html" \l "group-by-hidden-fields" \o "12.10.3. GROUP BY with Hidden Fields)。

在大多数情况下，DISTINCT子句可以视为GROUP BY的特殊情况。例如，下面的两个查询是等效的：

SELECT DISTINCT c1, c2, c3 FROM t1 WHERE c1 > *const*;

SELECT c1, c2, c3 FROM t1 WHERE c1 > *const* GROUP BY c1, c2, c3;

由于这个等效性，适用于GROUP BY查询的优化也适用于有DISTINCT子句的查询。这样，关于DISTINCT查询的优化的更详细的情况，参见[7.2.13节，“MySQL如何优化GROUP BY”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "group-by-optimization" \o "7.2.13. How MySQL Optimizes GROUP BY)。

结合LIMIT *row\_count*和DISTINCT后，MySQL发现唯一的*row\_count*行后立即停止。

如果不使用查询中命名的所有表的列，MySQL发现第1个匹配后立即停止扫描未使用的表。在下面的情况中，假定t1在t2之前使用(可以用EXPLAIN检查)，发现t2中的第1行后，MySQL不再(为t1中的任何行)读t2：

SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;

### 7.2.9. MySQL如何优化LEFT JOIN和RIGHT JOIN

在MySQL中，*A* LEFT JOIN *B* join\_condition执行过程如下：

·         根据表*A*和*A*依赖的所有表设置表B。

·         根据LEFT JOIN条件中使用的所有表(除了*B*)设置表*A*。

·         LEFT JOIN条件用于确定如何从表*B*搜索行。(换句话说，不使用WHERE子句中的任何条件）。

·         可以对所有标准联接进行优化，只是只有从它所依赖的所有表读取的表例外。如果出现循环依赖关系，MySQL提示出现一个错误。

·         进行所有标准WHERE优化。

·         如果*A*中有一行匹配WHERE子句，但*B*中没有一行匹配ON条件，则生成另一个*B*行，其中所有列设置为NULL。

·         如果使用LEFT JOIN找出在某些表中不存在的行，并且进行了下面的测试：WHERE部分的*col\_name* IS NULL，其中*col\_name*是一个声明为 NOT NULL的列，MySQL找到匹配LEFT JOIN条件的一个行后停止(为具体的关键字组合)搜索其它行。

RIGHT JOIN的执行类似LEFT JOIN，只是表的角色反过来。

联接优化器计算表应联接的顺序。LEFT JOIN和STRAIGHT\_JOIN强制的表读顺序可以帮助联接优化器更快地工作，因为检查的表交换更少。请注意这说明如果执行下面类型的查询，MySQL进行全扫描b，因为LEFT JOIN强制它在d之前读取：

SELECT \*

    FROM a,b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)

    WHERE b.key=d.key;

在这种情况下修复时用a的相反顺序，b列于FROM子句中：

SELECT \*

    FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)

    WHERE b.key=d.key;

MySQL可以进行下面的LEFT JOIN优化：如果对于产生的NULL行，WHERE条件总为假，LEFT JOIN变为普通联接。

例如，在下面的查询中如果t2.column1为NULL，WHERE 子句将为false：

SELECT \* FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;

因此，可以安全地将查询转换为普通联接：

SELECT \* FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;

这样可以更快，因为如果可以使查询更佳，MySQL可以在表t1之前使用表t2。为了强制使用表顺序，使用STRAIGHT\_JOIN。

### 7.2.10. MySQL如何优化嵌套Join

表示联接的语法允许嵌套联接。下面的讨论引用了[13.2.7.1节，“JOIN语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "join" \o "13.2.7.1. JOIN Syntax)中描述的联接语法。

同SQL标准比较，*table\_factor*语法已经扩展了。后者只接受*table\_reference*，而不是括号内所列的。

*table\_reference*项列表内的每个逗号等价于内部联接，这是一个保留扩展名。例如：

SELECT \* FROM t1 LEFT JOIN (t2, t3, t4)

                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

等价于：

SELECT \* FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)

                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

在MySQL中，CROSS JOIN语法上等价于INNER JOIN (它们可以彼此代替。在标准SQL中，它们不等价。INNER JOIN结合ON子句使用；CROSS JOIN 用于其它地方。

总的来说，在只包含内部联接操作的联接表达式中可以忽略括号。删除括号并将操作组合到左侧后，联接表达式：

t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)

   ON t1.a=t2.a

转换为表达式：

(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3

    ON t2.b=t3.b OR t2.b IS NULL

但是这两个表达式不等效。要说明这点，假定表t1、t2和t3有下面的状态：

·         表t1包含行{1}、{2}

·         表t2包含行{1,101}

·         表t3包含行{101}

在这种情况下，第1个表达式返回包括行{1,1,101,101}、{2,NULL,NULL,NULL}的结果，第2个表达式返回行{1,1,101,101}、{2,NULL,NULL,101}：

mysql> **SELECT \***

    -> **FROM t1**

    ->      **LEFT JOIN**

    ->      **(t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)**

    ->      **ON t1.a=t2.a;**

+------+------+------+------+

| a    | a    | b    | b    |

+------+------+------+------+

|    1 |    1 |  101 |  101 |

|    2 | NULL | NULL | NULL |

+------+------+------+------+

mysql> **SELECT \***

    -> **FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)**

    ->      **LEFT JOIN t3**

    ->      **ON t2.b=t3.b OR t2.b IS NULL;**

+------+------+------+------+

| a    | a    | b    | b    |

+------+------+------+------+

|    1 |    1 |  101 |  101 |

|    2 | NULL | NULL |  101 |

+------+------+------+------+

在下面的例子中，外面的联接操作结合内部联接操作使用：

t1 LEFT JOIN (t2，t3) ON t1.a=t2.a

该表达式不能转换为下面的表达式：

t1 LEFT JOIN t2 ON t1.a=t2.a，t3.

对于给定的表状态，第1个表达式返回行{1,1,101,101}、{2,NULL,NULL,NULL}，第2个表达式返回行{1,1,101,101}、{2,NULL,NULL,101}：

mysql> **SELECT \***

    -> **FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;**

+------+------+------+------+

| a    | a    | b    | b    |

+------+------+------+------+

|    1 |    1 |  101 |  101 |

|    2 | NULL | NULL | NULL |

+------+------+------+------+

mysql> **SELECT \***

    -> **FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;**

+------+------+------+------+

| a    | a    | b    | b    |

+------+------+------+------+

|    1 |    1 |  101 |  101 |

|    2 | NULL | NULL |  101 |

+------+------+------+------+

因此，如果我们忽略联接表达式中的括号连同外面的联接操作符，我们会改变原表达式的结果。

更确切地说，我们不能忽视左外联接操作的右操作数和右联接操作的左操作数中的括号。换句话说，我们不能忽视外联接操作中的内表达式中的括号。可以忽视其它操作数中的括号(外部表的操作数)。

对于任何表t1、t2、t3和属性t2.b和t3.b的任何条件P，下面的表达式：

(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)

等价于表达式

t1，t2 LEFT JOIN t3 ON P(t2.b,t3.b)

如果联接表达式(*join\_table*)中的联接操作的执行顺序不是从左到右，我们则应讨论嵌套的联接。这样，下面的查询：

SELECT \* FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a

  WHERE t1.a > 1

SELECT \* FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a

  WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1

联接表：

t2 LEFT JOIN t3 ON t2.b=t3.b

t2, t3

认为是嵌套的。第1个查询结合左联接操作则形成嵌套的联接，而在第二个查询中结合内联接操作形成嵌套联接。

在第1个查询中，括号可以忽略：联接表达式的语法结构与联接操作的执行顺序相同。但对于第2个查询，括号不能省略，尽管如果没有括号，这里的联接表达式解释不清楚。(在外部扩展语法中，需要第2个查询的(t2，t3)的括号，尽管从理论上对查询分析时不需要括号：这些查询的语法结构将仍然不清楚，因为LEFT JOIN和ON将充当表达式(t2,t3)的左、右界定符的角色)。

前面的例子说明了这些点：

·         对于只包含内联接(而非外联接)的联接表达式，可以删除括号。你可以移除括号并从左到右评估(或实际上，你可以按任何顺序评估表)。

·         总的来说，对外联接却不是这样。去除括号可能会更改结果。

·         总的来说，对外联接和内联接的结合，也不是这样。去除括号可能会更改结果。

含嵌套外联接的查询按含内联接的查询的相同的管道方式执行。更确切地说，利用了嵌套环联接算法。让我们回忆嵌套环联接执行查询时采用什么算法。

假定我们有一个如下形式的表T1、T2、T3的联接查询：

SELECT \* FROM T1 INNER JOIN T2 ON P1(T1,T2)

                 INNER JOIN T3 ON P2(T2,T3)

  WHERE P(T1,T2,T3).

这里，P1(T1,T2)和P2(T3,T3)是一些联接条件(表达式)，其中P(t1,t2,t3)是表T1、T2、T3的列的一个条件。

嵌套环联接算法将按下面的方式执行该查询：

FOR each row t1 in T1 {

  FOR each row t2 in T2 such that P1(t1,t2) {

    FOR each row t3 in T3 such that P2(t2,t3) {

      IF P(t1,t2,t3) {

         t:=t1||t2||t3; OUTPUT t;

      }

    }

  }

}

符号t1||t2||t3表示“连接行t1、t2和t3的列组成的行”。在下面的一些例子中，出现行名的NULL表示NULL用于行的每个列。例如，t1||t2||NULL表示“连接行t1和t2的列以及t3的每个列的NULL组成的行”。

现在让我们考虑带嵌套的外联接的查询：

SELECT \* FROM T1 LEFT JOIN

              (T2 LEFT JOIN T3 ON P2(T2,T3))

              ON P1(T1,T2)

  WHERE P(T1,T2,T3)。

对于该查询我们修改嵌套环模式可以得到：

FOR each row t1 in T1 {

  BOOL f1:=FALSE;

  FOR each row t2 in T2 such that P1(t1,t2) {

    BOOL f2:=FALSE;

    FOR each row t3 in T3 such that P2(t2,t3) {

      IF P(t1,t2,t3) {

        t:=t1||t2||t3; OUTPUT t;

      }

      f2=TRUE;

      f1=TRUE;

    }

    IF (!f2) {

      IF P(t1,t2,NULL) {

        t:=t1||t2||NULL; OUTPUT t;

      }

      f1=TRUE;

    }

  }

  IF (!f1) {

    IF P(t1,NULL,NULL) {

      t:=t1||NULL||NULL; OUTPUT t;

    }

  }

}

总的来说，对于外联接操作中的第一个内表的嵌套环，引入了一个标志，在环之前关闭并且在环之后打开。如果对于外部表的当前行，如果匹配表示内操作数的表，则标志打开。如果在循环结尾处标志仍然关闭，则对于外部表的当前行，没有发现匹配。在这种情况下，对于内表的列，应使用NULL值补充行。结果行被传递到输出进行最终检查或传递到下一个嵌套环，但只能在行满足所有嵌入式外联接的联接条件时。

在我们的例子中，嵌入了下面表达式表示的外联接表：

(T2 LEFT JOIN T3 ON P2(T2,T3))

请注意对于有内联接的查询，优化器可以选择不同的嵌套环顺序，例如：

FOR each row t3 in T3 {

  FOR each row t2 in T2 such that P2(t2,t3) {

    FOR each row t1 in T1 such that P1(t1,t2) {

      IF P(t1,t2,t3) {

         t:=t1||t2||t3; OUTPUT t;

      }

    }

  }

}

对于有外联接的查询，优化器可以只选择这样的顺序：外表的环优先于内表的环。这样，对于有外联接的查询，只可能有一种嵌套顺序。在下面的查询中，优化器将评估两个不同的嵌套：

SELECT \* T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)

  WHERE P(T1,T2,T3)

嵌套为：

FOR each row t1 in T1 {

  BOOL f1:=FALSE;

  FOR each row t2 in T2 such that P1(t1,t2) {

    FOR each row t3 in T3 such that P2(t1,t3) {

      IF P(t1,t2,t3) {

        t:=t1||t2||t3; OUTPUT t;

      }

      f1:=TRUE

    }

  }

  IF (!f1) {

    IF P(t1,NULL,NULL) {

      t:=t1||NULL||NULL; OUTPUT t;

    }

  }

}

和

FOR each row t1 in T1 {

  BOOL f1:=FALSE;

  FOR each row t3 in T3 such that P2(t1,t3) {

    FOR each row t2 in T2 such that P1(t1,t2) {

      IF P(t1,t2,t3) {

        t:=t1||t2||t3; OUTPUT t;

      }

      f1:=TRUE

    }

  }

  IF (!f1) {

    IF P(t1,NULL,NULL) {

      t:=t1||NULL||NULL; OUTPUT t;

    }

  }

}

在两个嵌套中，必须在外环中处理T1，因为它用于外联接中。T2和T3用于内联接中，因此联接必须在内环中处理。但是，因为该联接是一个内联接，T2和T3可以以任何顺序处理。

当讨论内联接嵌套环的算法时，我们忽略了部分详情，可能对查询执行的性能的影响会很大。我们没有提及所谓的“下推”条件。假定可以用连接公式表示我们的WHERE条件P(T1,T2,T3)：

P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3)。

在这种情况下，MySQL实际使用了下面的嵌套环方案来执行带内联接得到查询：

FOR each row t1 in T1 such that C1(t1) {

  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2)  {

    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {

      IF P(t1,t2,t3) {

         t:=t1||t2||t3; OUTPUT t;

      }

    }

  }

}

你会看见每个连接 C1(T1)，C2(T2)，C3(T3)被从最内部的环内推出到可以对它进行评估的最外的环中。如果C1(T1)是一个限制性很强的条件，下推条件可以大大降低从表T1传递到内环的行数。结果是查询大大加速。

对于有外联接的查询，只有查出外表的当前的行可以匹配内表后，才可以检查WHERE条件。这样，对内嵌套环下推的条件不能直接用于带外联接的查询。这里我们必须引入有条件下推前提，由遇到匹配后打开的标志保护。

对于带下面的外联接的例子

P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)

使用受保护的下推条件的嵌套环方案看起来应为：

FOR each row t1 in T1 such that C1(t1) {

  BOOL f1:=FALSE;

  FOR each row t2 in T2

      such that P1(t1,t2) AND (f1?C2(t2):TRUE) {

    BOOL f2:=FALSE;

    FOR each row t3 in T3

        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {

      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {

        t:=t1||t2||t3; OUTPUT t;

      }

      f2=TRUE;

      f1=TRUE;

    }

    IF (!f2) {

      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {

        t:=t1||t2||NULL; OUTPUT t;

      }

      f1=TRUE;

    }

  }

  IF (!f1 && P(t1,NULL,NULL)) {

      t:=t1||NULL||NULL; OUTPUT t;

  }

}

总的来说，可以从联接条件（例如P1(T1,T2)和P(T2,T3)）提取下推前提。在这种情况下，下推前提也受一个标志保护，防止检查由相应外联接操作所产生的NULL-补充的行的断言。

请注意如果从判断式的WHERE条件推导出，根据从一个内表到相同嵌套联接的另一个表的关键字进行的访问被禁止。(在这种情况下，我们可以使用有条件关键字访问，但是该技术还未用于MySQL 5.1中）。

### 7.2.11. MySQL如何简化外部联合

在许多情况下，一个查询的FROM子句的表的表达式可以简化。

在分析阶段，带右外联接操作的查询被转换为只包含左联接操作的等效查询。总的来说，根据以下原则进行转换：

(T1, ...) RIGHT JOIN (T2,...) ON P(T1,...,T2,...) =

(T2, ...) LEFT JOIN (T1,...) ON P(T1,...,T2,...)

所有T1 INNER JOIN T2 ON P(T1,T2)形式的内联接表达式被替换为T1,T2、P(T1,T2)，并根据WHERE条件(或嵌入连接的联接条件，如果有)联接为一个连接。

当优化器为用外联接操作的联接查询评估方案时，它只考虑在访问内表之前访问外表的操作的方案。优化器选项受到限制，因为只有这样的方案允许我们用嵌套环机制执行带外联接操作的查询。

假定我们有一个下列形式的查询：

SELECT \* T1 LEFT JOIN T2 ON P1(T1,T2)

  WHERE P(T1,T2) AND R(T2)

R(T2)大大减少了表T2中匹配的行数。如果我们这样执行查询，优化器将不会有其它选择，只能在访问表T2之前访问表T1，从而导致执行方案非常低。

幸运的是，如果WHERE条件拒绝null，MySQL可以将此类查询转换为没有外联接操作的查询。如果为该操作构建的NULL补充的行评估为FALSE或UNKNOWN，则该条件称为对于某个外联接操作拒绝null。

因此，对于该外联接：

T1 LEFT JOIN T2 ON T1.A=T2.A

类似下面的条件为拒绝null：

T2.B IS NOT NULL,

T2.B > 3,

T2.C <= T1.C,

T2.B < 2 OR T2.C > 1

类似下面的条件不为拒绝null：

T2.B IS NULL,

T1.B < 3 OR T2.B IS NOT NULL,

T1.B < 3 OR T2.B > 3

检查一个外联接操作的条件是否拒绝null的总原则很简单。以下情况下为拒绝null的条件：

·         形式为A IS NOT NULL，其中A是任何内表的一个属性

·         包含内表引用的判断式，当某个参量为NULL时评估为UNKNOWN

·         包含用于连接的拒绝null的条件的联合

·         拒绝null的条件的逻辑和

一个条件可以对于一个查询中的一个外联接操作为拒绝null的而对于另一个不为拒绝null的。在下面的查询中：

SELECT \* FROM T1 LEFT JOIN T2 ON T2.A=T1.A

                 LEFT JOIN T3 ON T3.B=T1.B

  WHERE T3.C > 0

WHERE条件对于第2个外联接操作为拒绝null的但对于第1个不为拒绝null的。

如果WHERE条件对于一个查询中的一个外联接操作为拒绝null的，外联接操作被一个内联接操作代替。

例如，前面的查询被下面的查询代替：

SELECT \* FROM T1 LEFT JOIN T2 ON T2.A=T1.A

                 INNER JOIN T3 ON T3.B=T1.B

  WHERE T3.C > 0

对于原来的查询，优化器将评估只与一个访问顺序T1、T2、T3兼容的方案。在替换的查询中，还考虑了访问顺序T3、T1、T2。

一个外联接操作的转化可以触发另一个的转化。这样，查询：

SELECT \* FROM T1 LEFT JOIN T2 ON T2.A=T1.A

                 LEFT JOIN T3 ON T3.B=T2.B

  WHERE T3.C > 0

将首先转换为查询：

SELECT \* FROM T1 LEFT JOIN T2 ON T2.A=T1.A

                 INNER JOIN T3 ON T3.B=T2.B

  WHERE T3.C > 0

该查询等效于查询：

SELECT \* FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3

  WHERE T3.C > 0 AND T3.B=T2.B

现在剩余的外联接操作也可以被一个内联接替换，因为条件T3.B=T2.B为拒绝null的，我们可以得到一个根本没有外联接的查询：

SELECT \* FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3

  WHERE T3.C > 0 AND T3.B=T2.B

有时我们可以成功替换嵌入的外联接操作，但不能转换嵌入的外联接。下面的查询：

SELECT \* FROM T1 LEFT JOIN

              (T2 LEFT JOIN T3 ON T3.B=T2.B)

              ON T2.A=T1.A

  WHERE T3.C > 0

被转换为：

SELECT \* FROM T1 LEFT JOIN

              (T2 INNER JOIN T3 ON T3.B=T2.B)

              ON T2.A=T1.A

  WHERE T3.C > 0，

只能重新写为仍然包含嵌入式外联接操作的形式：

SELECT \* FROM T1 LEFT JOIN

              (T2,T3)

              ON (T2.A=T1.A AND T3.B=T2.B)

  WHERE T3.C > 0。

如果试图转换一个查询中的嵌入式外联接操作，我们必须考虑嵌入式外联接的联接条件和WHERE条件。在下面的查询中：

SELECT \* FROM T1 LEFT JOIN

              (T2 LEFT JOIN T3 ON T3.B=T2.B)

              ON T2.A=T1.A AND T3.C=T1.C

  WHERE T3.D > 0 OR T1.D > 0

WHERE条件对于嵌入式外联接不为拒绝null的，但嵌入式外联接T2.A=T1.A AND T3.C=T1.C的联接条件为拒绝null的。因此该查询可以转换为：

SELECT \* FROM T1 LEFT JOIN

              (T2, T3)

              ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B

  WHERE T3.D > 0 OR T1.D > 0

### 7.2.12. MySQL如何优化ORDER BY

在某些情况中，MySQL可以使用一个索引来满足ORDER BY子句，而不需要额外的排序。

即使ORDER BY不确切匹配索引，只要WHERE子句中的所有未使用的索引部分和所有额外的ORDER BY 列为常数，就可以使用索引。下面的查询使用索引来解决ORDER BY部分：

SELECT \* FROM t1

    ORDER BY *key\_part1*,*key\_part2*,... ;

SELECT \* FROM t1

    WHERE *key\_part1*=*constant*

    ORDER BY *key\_part2*;

SELECT \* FROM t1

    ORDER BY *key\_part1* DESC, *key\_part2* DESC;

SELECT \* FROM t1

    WHERE *key\_part1*=1

    ORDER BY *key\_part1* DESC, *key\_part2* DESC;

在某些情况下，MySQL不能使用索引来解决ORDER BY，尽管它仍然使用索引来找到匹配WHERE子句的行。这些情况包括：

·         对不同的关键字使用ORDER BY：

·                SELECT \* FROM t1 ORDER BY *key1*, *key2；*

·         对关键字的非连续元素使用ORDER BY：

·                SELECT \* FROM t1 WHERE *key2*=*constant* ORDER BY *key\_part2*；

·         混合ASC和DESC：

·                SELECT \* FROM t1 ORDER BY *key\_part1* DESC, *key\_part2* ASC；

·         用于查询行的关键字与ORDER BY中所使用的不相同：

·                SELECT \* FROM t1 WHERE *key2*=*constant* ORDER BY *key1*；

·         你正联接许多表，并且ORDER BY中的列并不是全部来自第1个用于搜索行的非常量表。(这是EXPLAIN输出中的没有const联接类型的第1个表）。

·         有不同的ORDER BY和GROUP BY表达式。

·         使用的表索引的类型不能按顺序保存行。例如，对于HEAP表的HASH索引情况即如此。

通过EXPLAIN SELECT ...ORDER BY，可以检查MySQL是否可以使用索引来解决查询。如果Extra列内有Using filesort，则不能解决查询。参见[7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "explain" \o "7.2.1. EXPLAIN Syntax (Get Information About a SELECT))。

文件排序优化不仅用于记录排序关键字和行的位置，并且还记录查询需要的列。这样可以避免两次读取行。文件排序算法的工作象这样：

1.    读行匹配WHERE子句的行，如前面所示。

2.    对于每个行，记录构成排序关键字和行位置的一系列值，并且记录查询需要的列。

3.    根据排序关键字排序元组

4.    按排序的顺序检索行，但直接从排序的元组读取需要的列，而不是再一次访问表。

该算法比以前版本的Mysql有很大的改进。

为了避免速度变慢，该优化只用于排序元组中的extra列的总大小不超过max\_length\_for\_sort\_data系统变量值的时候。(将该变量设置得太高的的迹象是将看到硬盘活动太频繁而CPU活动较低）。

如果想要增加ORDER BY的速度，首先看是否可以让MySQL使用索引而不是额外的排序阶段。如果不能，可以尝试下面的策略：

·         增加sort\_buffer\_size变量的大小。

·         增加read\_rnd\_buffer\_size变量的大小。

·         更改tmpdir指向具有大量空闲空间的专用文件系统。该选项接受几个使用round-robin(循环)模式的路径。在Unix中路径应用冒号(‘:’)区间开，在Windows、NetWare和OS/2中用分号(‘；’)。可以使用该特性将负载均分到几个目录中。注释：路径应为位于不同物理硬盘上的文件系统的目录，而不是同一硬盘的不同的分区。

默认情况下，MySQL排序所有GROUP BY *col1*，*col2*，...查询的方法如同在查询中指定ORDER BY *col1*，*col2*，...。如果显式包括一个包含相同的列的ORDER BY子句，MySQL可以毫不减速地对它进行优化，尽管仍然进行排序。如果查询包括GROUP BY但你想要避免排序结果的消耗，你可以指定ORDER BY NULL禁止排序。例如：

INSERT INTO foo

SELECT a, COUNT(\*) FROM bar GROUP BY a ORDER BY NULL;

### 7.2.13. MySQL如何优化GROUP BY

[7.2.13.1. 松散索引扫描](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "loose-index-scan)

[7.2.13.2. 紧凑索引扫描](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "tight-index-scan)

满足GROUP BY子句的最一般的方法是扫描整个表并创建一个新的临时表，表中每个组的所有行应为连续的，然后使用该临时表来找到组并应用累积函数(如果有)。在某些情况中，MySQL能够做得更好，通过索引访问而不用创建临时表。

为GROUP BY使用索引的最重要的前提条件是 所有GROUP BY列引用同一索引的属性，并且索引按顺序保存其关键字(例如，这是B-树索引，而不是HASH索引)。是否用索引访问来代替临时表的使用还取决于在查询中使用了哪部分索引、为该部分指定的条件，以及选择的累积函数。

有两种方法通过索引访问执行GROUP BY查询，如下面的章节所描述。在第1个方法中，组合操作结合所有范围判断式使用(如果有)。第2个方法首先执行范围扫描，然后组合结果元组。

#### 7.2.13.1. 松散索引扫描

使用索引时最有效的途径是直接搜索组域。通过该访问方法，MySQL使用某些关键字排序的索引类型(例如，B-树)的属性。该属性允许使用 索引中的查找组而不需要考虑满足所有WHERE条件的索引中的所有关键字。既然该访问方法只考虑索引中的关键字的一小部分，它被称为松散索引扫描。如果没有WHERE子句， 松散索引扫描读取的关键字数量与组数量一样多，可以比所有关键字数小得多。如果WHERE子句包含范围判断式(关于range联接类型的讨论参见[7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "explain" \o "7.2.1. EXPLAIN Syntax (Get Information About a SELECT)))， 松散索引扫描查找满足范围条件的每个组的第1个关键字，并且再次读取尽可能最少数量的关键字。在下面的条件下是可以的：

·         查询针对一个单表。

·         GROUP BY包括索引的第1个连续部分(如果对于GROUP BY，查询有一个DISTINCT子句，则所有显式属性指向索引开头)。

·         只使用累积函数(如果有)MIN()和MAX()，并且它们均指向相同的列。

·         索引的任何其它部分（除了那些来自查询中引用的GROUP BY）必须为常数(也就是说，必须按常量数量来引用它们)，但MIN()或MAX() 函数的参数例外。

此类查询的EXPLAIN输出显示Extra列的Using indexforgroup-by。

下面的查询提供该类的几个例子，假定表t1(c1,c2,c3,c4)有一个索引idx(c1，c2，c3)：

SELECT c1, c2 FROM t1 GROUP BY c1, c2;

SELECT DISTINCT c1, c2 FROM t1;

SELECT c1, MIN(c2) FROM t1 GROUP BY c1;

SELECT c1, c2 FROM t1 WHERE c1 < *const* GROUP BY c1, c2;

SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > *const* GROUP BY c1, c2;

SELECT c2 FROM t1 WHERE c1 < *const* GROUP BY c1, c2;

SELECT c1, c2 FROM t1 WHERE c3 = *const* GROUP BY c1, c2；

由于上述原因，不能用该快速选择方法执行下面的查询：

1.      除了MIN()或MAX()，还有其它累积函数，例如：

     SELECT c1, SUM(c2) FROM t1 GROUP BY c1;

2.      GROUP BY子句中的域不引用索引开头，如下所示：

     SELECT c1,c2 FROM t1 GROUP BY c2, c3;

3.      查询引用了GROUP BY部分后面的关键字的一部分，并且没有等于常量的等式，例如：

     SELECT c1,c3 FROM t1 GROUP BY c1, c2；

#### 7.2.13.2. 紧凑索引扫描

紧凑式索引扫描可以为索引扫描或一个范围索引扫描，取决于查询条件。

如果不满足松散索引扫描条件，GROUP BY查询仍然可以不用创建临时表。如果WHERE子句中有范围条件，该方法只读取满足这些条件的关键字。否则，进行索引扫描。该方法读取由WHERE子句定义的每个范围的所有关键字，或没有范围条件式扫描整个索引，我们将它定义为紧凑式索引扫描。请注意对于紧凑式索引扫描，只有找到了满足范围条件的所有关键字后才进行组合操作。

要想让该方法工作，对于引用GROUP BY关键字元素的前面、中间关键字元素的查询中的所有列，有一个常量等式条件即足够了。等式条件中的常量填充了搜索关键字中的“差距”，可以形成完整的索引前缀。这些索引前缀可以用于索引查找。如果需要排序GROUP BY结果，并且能够形成索引前缀的搜索关键字，MySQL还可以避免额外的排序操作，因为使用有顺序的索引的前缀进行搜索已经按顺序检索到了所有关键字。

上述的第一种方法不适合下面的查询，但第2种索引访问方法可以工作(假定我们已经提及了表t1的索引idx)：

·         GROUP BY中有一个差距，但已经由条件c2 = 'a'覆盖。

     SELECT c1，c2，c3 FROM t1 WHERE c2 = 'a' GROUP BY c1，c3;

·         GROUP BY不以关键字的第1个元素开始，但是有一个条件提供该元素的常量：

     SELECT c1，c2，c3 FROM t1 WHERE c1 = 'a' GROUP BY c2，c3;

### 7.2.14. MySQL如何优化LIMIT

在一些情况中，当你使用LIMIT *row\_count*而不使用HAVING时，**MySQL**将以不同方式处理查询。

·         如果你用LIMIT只选择一些行，当**MySQL选择**做完整的表扫描时，它将在一些情况下使用索引。

·         如果你使用LIMIT *row\_count*与ORDER BY，**MySQL**一旦找到了排序结果的第一个*row\_count*行，将结束排序而不是排序整个表。如果使用索引，将很快。如果必须进行文件排序（filesort），必须选择所有匹配查询没有LIMIT子句的行，并且在确定已经找到第1个*row\_count*行前，必须对它们的大部分进行排序。在任何一种情况下，一旦找到了行，则不需要再排序结果的其它部分，并且MySQL不再进行排序。

·         当结合LIMIT *row\_count*和DISTINCT时，**MySQL**一旦找到*row\_count*个唯一的行，它将停止。

·         在一些情况下，GROUP BY能通过顺序读取键(或在键上做排序)来解决，然后计算摘要直到关键字的值改变。在这种情况下，LIMIT *row\_count*将不计算任何不必要的GROUP BY值。

·         只要MySQL已经发送了需要的行数到客户，它将放弃查询，除非你正使用SQL\_CALC\_FOUND\_ROWS。

·         LIMIT 0将总是快速返回一个空集合。这对检查查询的有效性是有用的。当使用MySQL API时，它也可以用来得到结果列的列类型。(该技巧在MySQL Monitor中不工作，只显示Empty set；应使用SHOW COLUMNS或DESCRIBE）。

·         当服务器使用临时表来进行查询时，使用LIMIT *row\_count*子句来计算需要多少空间。

### 7.2.15. 如何避免表扫描

EXPLAIN的输出显示了当MySQL使用表扫描来解决查询时使用的所有类型列。这通常在如下条件下发生：

·         表很小，扫描表比查找关键字速度快。这对于少于10行并且行较短的表比较普遍。

·         在ON或WHERE子句中没有适用的索引列的约束。

·         正用常量值比较索引列，并且MySQL已经计算到(基于索引树)常数覆盖了表的很大部分并且表扫描将会比较快。参见[7.2.4节，“MySQL怎样优化WHERE子句”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "where-optimizations" \o "7.2.4. How MySQL Optimizes WHERE Clauses)。

·         你正通过另一个列使用一个低的集的势的关键字(许多行匹配关键字)。在这种情况下，MySQL假设通过使用关键字它可能会进行许多关键字查找，表扫描将会更快。

对于小表，表扫描通常合适。对于大表，尝试下面的技巧以避免优化器错选了表扫描：

·         使用ANALYZE TABLE *tbl\_name*为扫描的表更新关键字分布。参见[13.5.2.1节，“ANALYZE TABLE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "analyze-table" \o "13.5.2.1. ANALYZE TABLE Syntax)。

·         对扫描的表使用FORCE INDEX告知MySQL，相对于使用给定的索引表扫描将非常耗时。参见[13.2.7节，“SELECT语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "select" \o "13.2.7. SELECT Syntax)。

·                SELECT \* FROM t1, t2 FORCE INDEX (*index\_for\_column*)

·                    WHERE t1.*col\_name*=t2.*col\_name*；

·         用--max-seeks-for-key=1000选项启动**mysqld**或使用SET max\_seeks\_for\_key=1000告知优化器假设关键字扫描不会超过1,000次关键字搜索。参见[5.3.3节，“服务器系统变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-system-variables" \o "5.3.3. Server System Variables)。

### 7.2.16. INSERT语句的速度

插入一个记录需要的时间由下列因素组成，其中的数字表示大约比例：

* 连接：(3)
* 发送查询给服务器：(2)
* 分析查询：(2)
* 插入记录：（1x记录大小）
* 插入索引：（1x索引）
* 关闭：(1)

这不考虑打开表的初始开销，每个并发运行的查询打开。

表的大小以log*N* (B树)的速度减慢索引的插入。

加快插入的一些方法：

·         如果同时从同一个客户端插入很多行，使用含多个VALUE的INSERT语句同时插入几行。这比使用单行INSERT语句快(在某些情况下快几倍)。如果你正向一个非空表添加数据，可以调节bulk\_insert\_buffer\_size变量，使数据插入更快。参见[5.3.3节，“服务器系统变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-system-variables" \o "5.3.3. Server System Variables)。

·         如果你从不同的客户端插入很多行，能通过INSERT DELAYED语句加快速度。参见[13.2.4节，“INSERT语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "insert" \o "13.2.4. INSERT Syntax)。

·         用MyISAM，如果在表中没有删除的行，能在SELECT语句正在运行的同时插入行。

·         当从一个文本文件装载一个表时，使用LOAD DATA INFILE。这通常比使用很多INSERT语句快20倍。参见[13.2.5节，“LOAD DATA INFILE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "load-data" \o "13.2.5. LOAD DATA INFILE Syntax)。

·         当表有很多索引时，有可能要多做些工作使得LOAD DATA INFILE更快些。使用下列过程：

* 1. 有选择地用CREATE TABLE创建表。
  2. 执行FLUSH TABLES语句或命令**mysqladmin flush-tables**。
  3. 使用**myisamchk --keys-used=0 -rq */path/to/db/tbl\_name***。这将从表中取消所有索引的使用。
  4. 用LOAD DATA INFILE把数据插入到表中，因为不更新任何索引，因此很快。
  5. 如果只想在以后读取表，使用**myisampack**压缩它。参见[15.1.3.3节，“压缩表特性”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "compressed-format" \o "15.1.3.3. Compressed Table Characteristics)。
  6. 用**myisamchk -r -q */path/to/db/tbl\_name***重新创建索引。这将在写入磁盘前在内存中创建索引树，并且它更快，因为避免了大量磁盘搜索。结果索引树也被完美地平衡。
  7. 执行FLUSH TABLES语句或**mysqladmin flush-tables**命令。

请注意如果插入一个空MyISAM表，LOAD DATA INFILE也可以执行前面的优化；主要不同处是可以让**myisamchk**为创建索引分配更多的临时内存，比执行LOAD DATA INFILE语句时为服务器重新创建索引分配得要多。

也可以使用ALTER TABLE *tbl\_name* DISABLE KEYS代替**myisamchk --keys-used=0 -rq */path/to/db/tbl\_name***，使用ALTER TABLE *tbl\_name* ENABLE KEYS代替**myisamchk -r -q */path/to/db/tbl\_name***。使用这种方式，还可以跳过FLUSH TABLES。

·         锁定表可以加速用多个语句执行的INSERT操作：

* LOCK TABLES a WRITE;
* INSERT INTO a VALUES (1,23),(2,34),(4,33);
* INSERT INTO a VALUES (8,26),(6,29);
* UNLOCK TABLES;

这样性能会提高，因为索引缓存区仅在所有INSERT语句完成后刷新到磁盘上一次。一般有多少INSERT语句即有多少索引缓存区刷新。如果能用一个语句插入所有的行，就不需要锁定。

对于事务表，应使用BEGIN和COMMIT代替LOCK TABLES来加快插入。

锁定也将降低多连接测试的整体时间，尽管因为它们等候锁定最大等待时间将上升。例如：

Connection 1 does 1000 inserts

Connections 2, 3, and 4 do 1 insert

Connection 5 does 1000 inserts

如果不使用锁定，2、3和4将在1和5前完成。如果使用锁定，2、3和4将可能不在1或5前完成，但是整体时间应该快大约40%。

INSERT、UPDATE和DELETE操作在**MySQL**中是很快的，通过为在一行中多于大约5次连续不断地插入或更新的操作加锁，可以获得更好的整体性能。如果在一行中进行多次插入，可以执行LOCK TABLES，随后立即执行UNLOCK TABLES(大约每1000行)以允许其它的线程访问表。这也会获得好的性能。

INSERT装载数据比LOAD DATA INFILE要慢得多，即使是使用上述的策略。

·         为了对LOAD DATA INFILE和INSERT在MyISAM表得到更快的速度，通过增加key\_buffer\_size系统变量来扩大 键高速缓冲区。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

### 7.2.17. UPDATE语句的速度

更新查询的优化同SELECT查询一样，需要额外的写开销。写速度依赖于更新的数据大小和更新的索引的数量。没有更改的索引不被更新。

使更改更快的另一个方法是推迟更改然后在一行内进行多次更新。如果锁定表，同时做多个更新比一次做一个快得多。

请注意对使用动态记录格式的MyISAM表，更新一个较长总长的记录可能会切分记录。如果经常这样该，偶尔使用OPTIMIZE TABLE很重要。参见[13.5.2.5节，“OPTIMIZE TABLE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "optimize-table" \o "13.5.2.5. OPTIMIZE TABLE Syntax)。

### 7.2.18. DELETE语句的速度

删除一个记录的时间与索引数量确切成正比。为了更快速地删除记录，可以增加键高速缓冲的大小。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

如果想要删除一个表的所有行，使用TRUNCATE TABLE *tbl\_name* 而不要用DELETE FROM *tbl\_name*。参见[13.2.9节，“TRUNCATE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "truncate" \o "13.2.9. TRUNCATE Syntax)。

### 7.2.19. 其它优化技巧

该节列出了提高查询速度的各种技巧：

·         使用持久的连接数据库以避免连接开销。如果不能使用持久的连接并且你正启动许多新的与数据库的连接，可能要更改thread\_cache\_size变量的值。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

·         总是检查所有查询确实使用已经在表中创建了的索引。在**MySQL**中，可以用EXPLAIN命令做到。参见[7.2.1节，“EXPLAIN语法（获取关于SELECT的信息）”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "explain" \o "7.2.1. EXPLAIN Syntax (Get Information About a SELECT))。

·         尝试避免在频繁更新的表上执行复杂的SELECT查询，以避免与锁定表有关的由于读、写冲突发生的问题。

·         对于没有删除的行的MyISAM表，可以在另一个查询正从表中读取的同时在末尾插入行。如果这很重要，应考虑按照避免删除行的方式使用表。另一个可能性是在删除大量行后运行OPTIMIZE TABLE。参见[15.1节，“MyISAM存储引擎”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "myisam-storage-engine" \o "15.1. The MyISAM Storage Engine)。

·         要修复任何ARCHIVE表可以发生的压缩问题，可以执行OPTIMIZE TABLE。参见[15.8节，“ARCHIVE存储引擎”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "archive-storage-engine" \o "15.8. The ARCHIVE Storage Engine)。

·         如果你主要按*expr1*，*expr2*，...顺序检索行，使用ALTER TABLE ... ORDER BY *expr1*, *expr2*, ...。对表大量更改后使用该选项，可以获得更好的性能。

·         在一些情况下，使得基于来自其它表的列的信息引入一个“ 哈希”的列有意义。如果该列较短并且有合理的唯一值，它可以比在许多列上的一个大索引快些。在**MySQL**中，很容易使用这个额外列：

·                SELECT \* FROM *tbl\_name*

·                     WHERE *hash\_col*=MD5(CONCAT(*col1*,*col2*))

·                     AND *col1*='*constant*' AND *col2*='*constant*';

·         对于频繁更改的MyISAM表，应试图避免所有变长列(VARCHAR、BLOB和TEXT)。如果表包括单一的变长列则使用动态记录格式。参见[第15章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html)*[存储引擎和表类型](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \o "Chapter 15. Storage Engines and Table Types)*。

·         只是因为行太大，将一张表分割为不同的表一般没有什么用处。为了访问行，最大的性能冲击是磁盘搜索以找到行的第一个字节。在找到数据后，大多数新型磁盘对大多数应用程序来说足够快，能读入整个行。确实有必要分割的唯一情形是如果它是使用动态记录格式使之变为固定的记录大小的MyISAM表(见上述)，或如果你需要很频繁地扫描表而不需要大多数列。参见[第15章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html)*[存储引擎和表类型](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \o "Chapter 15. Storage Engines and Table Types)*。

·         如果你需要很经常地计算结果，例如基于来自很多行的信息的计数，引入一个新表并实时更新计数器可能更好一些。下面形式的更新会更快一些：

UPDATE *tbl\_name* SET *count\_col*=*count\_col*+1 WHERE *key\_col*=*constant*;

当你使用象MyISAM那样的只有表级锁定的**MySQL存储引擎**(多重读/单个写)时，这确实很重要。这也给大多数数据库较好的性能，因为行锁定管理器在这种情况下有较少的事情要做。

·         如果你需要从大的记录文件表中收集统计信息，使用总结性的表而不是扫描整个表。维护总结应该比尝试做“实时”统计要快些。当有变化时从日志重新生成新的总结表比改变运行的应用(取决于业务决策)要快得多。

* 如果可能，应该将报告分类为“实时”或“统计”，这里统计报告所需的数据仅仅基于从实际数据定期产生的总结表中产生。
* 充分利用列有默认值的事实。只有当插入的值不同于默认值时，才明确地插入值。这减少MySQL需要做的语法分析从而提高插入速度。
* 在一些情况下，包装并存储数据到一个BLOB列中是很方便的。在这种情况下，必须在你的应用中增加额外的代码来打包/解包信息，但是这种方法可以在某些阶段节省很多访问。当有不符合行和列表结构的数据时，这很实用。
* 在一般情况下，应该尝试以非冗余方式(查看数据库理论中的*第三正则形式*)保存数据，但是为了获得更快的速度，可以复制信息或创建总结表。
* 存储过程或UDF(用户定义函数)可能是获得更好性能的一个好方法，详细信息参见[第20章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/stored-procedures.html)*[存储程序和函数](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/stored-procedures.html" \o "Chapter 20. Stored Procedures and Functions)*和[27.2节，“为MySQL添加新函数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/extending-mysql.html" \l "adding-functions" \o "27.2. Adding New Functions to MySQL)。
* 总是能通过在应用程序中缓存查询/答案并尝试同时执行很多插入/更新来获得一些好处。如果数据库支持锁定表(象**MySQL**和Oracle)，这应该有助于确保索引缓存只在所有更新后刷新一次。还可以利用MySQL的查询缓存来获得类似的结果；参见[5.13节，“MySQL查询高速缓冲”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "query-cache" \o "5.13. The MySQL Query Cache)。
* 当不需要知道何时写入数据时，使用INSERT DELAYED。这样可以加快处理，因为很多记录可以通过一次磁盘写入被写入。
* 当你想要让选择显得更重要时，使用INSERT /\*! LOW\_PRIORITY \*/。
* 使用INSERT LOW\_PRIORITY来取得插入队列的检索，也就是即使有另一个客户等待写入也要执行SELECT。
* 使用多行INSERT语句通过一个SQL命令来存储很多行(许多SQL服务器支持它，包括MySQL)。
* 使用LOAD DATA INFILE装载较大数量的数据。这比使用INSERT要快得多。
* 使用AUTO\_INCREMENT列构成唯一值。
* 当MyISAM使用动态表格式时，偶尔使用OPTIMIZE TABLE可以避免碎片。参见[15.1.3节，“MyISAM表的存储格式”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "myisam-table-formats" \o "15.1.3. MyISAM Table Storage Formats)。
* 可能时使用MEMORY表以得到更快的速度。[参见](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/../../../澶╂渤/2006.1/1/IT/IT/wwn/manual_Server.html" \l "Table_types)[15.4节，“MEMORY (HEAP)存储引擎”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "memory-storage-engine" \o "15.4. The MEMORY (HEAP) Storage Engine)。
* 在Web服务器中，图象和其它二进制资产应该作为文件存储。也就是仅在数据库中存储的本文件的引用而不是文件本身。大多数Web服务器在缓存文件方面比数据库内容要好得多，因此使用文件一般要快得多。
* 对经常访问的不重要数据(如为没有在Web 浏览器中启用cookie的用户最后显示的标语的相关信息)使用内存表。在许多Web应用程序环境中也可以使用用户会话来处理可变状态数据。
* 在不同表中具有相同信息的列应该被声明为相同的并有相同的名字。尝试使名字简单化。例如，在customer表中使用name而不是customer\_name。为了使名字能移植到其它SQL服务器，应该使名字短于18个字符。
* 如果确实需要很高的速度，应该研究一下不同SQL服务器支持的数据存储的低层接口！例如直接访问**MySQL** MyISAM存储引擎，比起使用SQL接口，速度可以提高2-5倍。为了能实现，数据必须与应用程序在同一台服务器上，并且通常只应该被一个进程访问(因为外部文件锁定确实很慢)。通过在**MySQL**服务器中引进低层MyISAM命令能消除以上问题(如果需要，这可能是获得更好性能的一个简单的方法)。通过精心设计数据库接口，应该能相当容易地支持这类优化。
* 如果正使用数字数据，在许多情况下，从一个数据库访问信息(使用实时连接)比访问一个文本文件快些。这是因为数据库中的信息比文本文件更紧凑，因此这将涉及更少的磁盘访问。还可以在应用程序中节省代码，因为不须分析文本文件来找出行和列的边界。

·         通过复制可以提高某些操作的性能。可以在复制服务器中分布客户的检索以均分负载。为了防止备份时主服务器变慢，可以使用一个从服务器来备份。参见[第6章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/replication.html)*[MySQL中的复制](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/replication.html" \o "Chapter 6. Replication in MySQL)*。

·         用DELAY\_KEY\_WRITE=1选项声明MyISAM表可以使索引更新更快，因为在表关闭之前它们不刷新到硬盘上。不利之处是当表打开时如果杀掉服务器，应确保用--myisam-recover选项运行服务器保证没有问题，或者在重启服务器之前运行**myisamchk**。(然而，即使在这种情况下，应通过使用DELAY\_KEY\_WRITE保证不丢失数据，因为关键字信息总是可以从数据行产生）。

## 7.3. 锁定事宜

[7.3.1. 锁定方法](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "internal-locking)

[7.3.2. 表锁定事宜](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "table-locking)

### 7.3.1. 锁定方法

MySQL 5.1支持对MyISAM和MEMORY表进行表级锁定，对BDB表进行页级锁定，对InnoDB表进行行级锁定。

在许多情况下，可以根据培训猜测应用程序使用哪类锁定类型最好，但一般很难说出某个给出的锁类型就比另一个好。一切取决于应用程序，应用程序的不同部分可能需要不同的锁类型。

为了确定是否想要使用行级锁定的存储引擎，应看看应用程序做什么并且混合使用什么样的选择和更新语句。例如，大多数Web应用程序执行许多选择，而很少进行删除，只对关键字的值进行更新，并且只插入少量具体的表。基本MySQL MyISAM设置已经调节得很好。

在MySQL中对于使用表级锁定的存储引擎，表锁定时不会死锁的。这通过总是在一个查询开始时立即请求所有必要的锁定并且总是以同样的顺序锁定表来管理。

对WRITE，**MySQL**使用的表锁定方法原理如下：

* 如果在表上没有锁，在它上面放一个写锁。
* 否则，把锁定请求放在写锁定队列中。

对READ，**MySQL**使用的锁定方法原理如下：

* 如果在表上没有写锁定，把一个读锁定放在它上面。
* 否则，把锁请求放在读锁定队列中。

当一个锁定被释放时，锁定可被写锁定队列中的线程得到，然后是读锁定队列中的线程。

这意味着，如果你在一个表上有许多更新，SELECT语句将等待直到没有更多的更新。

可以通过检查table\_locks\_waited和table\_locks\_immediate状态变量来分析系统上的表锁定争夺：

mysql> **SHOW STATUS LIKE 'Table%';**

+-----------------------+---------+

| Variable\_name         | Value   |

+-----------------------+---------+

| Table\_locks\_immediate | 1151552 |

| Table\_locks\_waited    | 15324   |

+-----------------------+---------+

如果INSERT语句不冲突，可以自由为MyISAM表混合并行的INSERT和SELECT语句而不需要锁定。也就是说，你可以在其它客户正读取MyISAM表的时候插入行。如果数据文件中间不包含空闲块，不会发生冲突，因为在这种情况下，记录总是插入在数据文件的尾部。(从表的中部删除或更新的行可能导致空洞）。如果有空洞，当所有空洞填入新的数据时，并行的插入能够重新自动启用。

如果不能同时插入，为了在一个表中进行多次INSERT和SELECT操作，可以在临时表中插入行并且立即用临时表中的记录更新真正的表。

这可用下列代码做到：

mysql> **LOCK TABLES real\_table WRITE, insert\_table WRITE;**

mysql> **INSERT INTO real\_table SELECT \* FROM insert\_table;**

mysql> **TRUNCATE TABLE insert\_table;**

mysql> **UNLOCK TABLES;**

InnoDB使用行锁定，BDB使用页锁定。对于这两种存储引擎，都可能存在死锁。这是因为，在SQL语句处理期间，InnoDB自动获得行锁定和BDB获得页锁定，而不是在事务启动时获得。

行级锁定的优点：

·         当在许多线程中访问不同的行时只存在少量锁定冲突。

·         回滚时只有少量的更改。

·         可以长时间锁定单一的行。

行级锁定的缺点：

·         比页级或表级锁定占用更多的内存。

·         当在表的大部分中使用时，比页级或表级锁定速度慢，因为你必须获取更多的锁。

·         如果你在大部分数据上经常进行GROUP BY操作或者必须经常扫描整个表，比其它锁定明显慢很多。

·         用高级别锁定，通过支持不同的类型锁定，你也可以很容易地调节应用程序，因为其锁成本小于行级锁定。

在以下情况下，表锁定优先于页级或行级锁定：

·         表的大部分语句用于读取。

·         对严格的关键字进行读取和更新，你可以更新或删除可以用单一的读取的关键字来提取的一行：

·                UPDATE *tbl\_name* SET *column*=*value* WHERE *unique\_key\_col*=*key\_value*;

·                DELETE FROM *tbl\_name* WHERE *unique\_key\_col*=*key\_value*;

·         SELECT 结合并行的INSERT语句，并且只有很少的UPDATE或DELETE语句。

·         在整个表上有许多扫描或GROUP BY操作，没有任何写操作。

不同于行级或页级锁定的选项：

·         版本(例如，为并行的插入在MySQL中使用的技术)，其中可以一个写操作，同时有许多读取操作。这说明数据库或表支持数据依赖的不同视图，取决于访问何时开始。其它共同的术语是“时间跟踪”、“写复制”或者“按需复制”。

·         按需复制在许多情况下优先于页级或行级锁定。然而，在最坏的情况下，它可能比使用常规锁定使用更多的内存。

·         除了行级锁定外，你可以使用应用程序级锁定，例如在MySQL中使用GET\_LOCK()和RELEASE\_LOCK()。这些是建议性锁定，它们只能在运行良好的应用程序中工作。

### 7.3.2. 表锁定事宜

为达到最高锁定速度，除InnoDB和BDB之外，对所有存储引擎，MySQL使用表锁定(而不是页、行或者列锁定)。

对于InnoDB和BDB表，如果你用LOCK TABLES显式锁定表，MySQL只使用表锁定。对于这些表类型，我们建议你根本不要使用LOCK TABLES，因为InnoDB使用自动行级锁定而BDB使用页级锁定来保证事务隔离。

对于大表，对于大多数应用程序，表锁定比行锁定更好，但存在部分缺陷。

表锁定使许多线程同时从一个表中进行读取操作，但如果一个线程想要对表进行写操作，它必须首先获得独占访问。更新期间，所有其它想要访问该表的线程必须等待直到更新完成。

表更新通常情况认为比表检索更重要，因此给予它们更高的优先级。这应确保更新一个表的活动不能“饿死”，即使该表上有很繁重的SELECT活动。

表锁定在这种情况下会造成问题，例如当线程正等待，因为硬盘已满并且在线程可以处理之前必须有空闲空间。在这种情况下，所有想要访问出现问题的表的线程也被设置成等待状态，直到有更多的硬盘空间可用。

表锁定在下面的情况下也存在问题：

·         一个客户发出长时间运行的查询。

·         然后，另一个客户对同一个表进行更新。该客户必须等待直到SELECT完成。

·         另一个客户对同一个表上发出了另一个SELECT语句。因为UPDATE比SELECT优先级高，该SELECT语句等待UPDATE完成，并且等待第1个SELECT完成。

下面描述了一些方法来避免或减少表锁定造成的竞争：

·         试图使SELECT语句运行得更快。你可能必须创建一些摘要(summary)表做到这点。

·         用--low-priority-updates启动**mysqld**。这将给所有更新(修改)一个表的语句以比SELECT语句低的优先级。在这种情况下，在先前情形的第2个SELECT语句将在UPDATE语句前执行，而不需要等候第1个SELECT完成。

·         可以使用SET LOW\_PRIORITY\_UPDATES=1语句指定具体连接中的所有更新应使用低优先级。参见[13.5.3节，“SET语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "set-option" \o "13.5.3. SET Syntax)

·         可以用LOW\_PRIORITY属性给与一个特定的INSERT、UPDATE或DELETE语句较低优先级。

·         可以用HIGH\_PRIORITY属性给与一个特定的SELECT语句较高优先级。参见[13.2.7节，“SELECT语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "select" \o "13.2.7. SELECT Syntax)。

·         为**max\_write\_lock\_count**系统变量指定一个低值来启动**mysqld**来强制MySQL在具体数量的插入完成后临时提高所有等待一个表的SELECT语句的优先级。这样允许在一定数量的WRITE锁定后给出READ锁定。

·         如果你有关于INSERT结合SELECT的问题，切换到使用新的MyISAM表，因为它们支持并发的SELECT和INSERT。

·         如果你对同一个表混合插入和删除，INSERT DELAYED将会有很大的帮助。参见[13.2.4.2节，“INSERT DELAYED语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "insert-delayed" \o "13.2.4.2. INSERT DELAYED Syntax)。

·         如果你对同一个表混合使用SELECT和DELETE语句出现问题，DELETE的LIMIT选项可以有所帮助。参见[13.2.1节，“DELETE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "delete" \o "13.2.1. DELETE Syntax)。

·         对SELECT语句使用SQL\_BUFFER\_RESULT可以帮助使表锁定时间变短。参见[13.2.7节，“SELECT语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "select" \o "13.2.7. SELECT Syntax)。

·         可以更改mysys/thr\_lock.c中的锁代码以使用单一的队列。在这种情况下，写锁定和读锁定将具有相同的优先级，对一些应用程序会有帮助。

这里是一些MySQL中表锁定相关的技巧：

·         如果不混合更新与需要在同一个表中检查许多行的选择，可以进行并行操作。

·         可以使用LOCK TABLES来提高速度，因为在一个锁定中进行许多更新比没有锁定的更新要快得多。将表中的内容切分为几个表也可以有所帮助。

·         如果在MySQL中表锁定时遇到速度问题，可以将你的表转换为InnoDB或BDB表来提高性能。参见[15.2节，“InnoDB存储引擎”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "innodb" \o "15.2. The InnoDB Storage Engine)和[15.5节，“BDB (BerkeleyDB)存储引擎”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "bdb-storage-engine" \o "15.5. The BDB (BerkeleyDB) Storage Engine)。

## 7.4. 优化数据库结构

[7.4.1. 设计选择](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "design)

[7.4.2. 使你的数据尽可能小](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "data-size)

[7.4.3. 列索引](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "indexes)

[7.4.4. 多列索引](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "multiple-column-indexes)

[7.4.5. MySQL如何使用索引](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "mysql-indexes)

[7.4.6. MyISAM键高速缓冲](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "myisam-key-cache)

[7.4.7. MyISAM索引统计集合](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "myisam-index-statistics)

[7.4.8. MySQL如何计算打开的表](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "open-tables)

[7.4.9. MySQL如何打开和关闭表](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "table-cache)

[7.4.10. 在同一个数据库中创建多个表的缺陷](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "creating-many-tables)

### 7.4.1. 设计选择

MySQL将行数据和索引数据保存在不同的文件中。许多(几乎所有)其它数据库将行数据和索引数据混合保存在用一个文件中。我们认为MySQL 选择对广范围的现代系统更好一些。

保存行数据的另一种方式是将每个列的信息保存在单独的区域(例如SDBM和Focus)。这样会对每个访问多个列的查询造成性能问题。因为当访问多个列时退化得很快，我们认为该模型对一般数据库不合适。

更常见的情形是索引和数据保存在一起(例如Oracle/Sybase)。在这种情况下，你可以在索引的叶级页找到行的信息。该布局比较好的事情是在许多情况下，根据索引缓存得怎样，可以保存一个硬盘读取。该布局的不利之处表现在：

·         表扫描要慢得多，因为你必须通读索引以获得数据。

·         你不能只使用表来检索查询的数据。

·         你需要使用更多的空间，因为你必须从节点复制索引(你不能保存节点上的行)。

·         删除要经过一段时间后才退化表(因为删除时通常不会更新节点上的索引)。

·         只缓存索引数据会更加困难。

### 7.4.2. 使你的数据尽可能小

最基本的优化之一是使表在磁盘上占据的空间尽可能小。这能给出巨大的改进，因为磁盘读入较快，并且在查询执行过程中小表的内容被处理时占用较少的主存储器。如果在更小的列上做索引，索引也占据较少的资源。

MySQL支持许多不同的存储引擎(表类型)和行格式。对于每个表，可以确定使用哪个存储引擎和索引方法。为应用程序选择合适的表格式可以大大提高性能。参见[第15章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html)*[存储引擎和表类型](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \o "Chapter 15. Storage Engines and Table Types)*。

可以使用下面的技术可以使表的性能更好并且使存储空间最小：

* 尽可能地使用最有效(最小)的数据类型。**MySQL**有很多节省磁盘空间和内存的专业化类型。
* 尽可能使用较小的整数类型使表更小。例如，MEDIUMINT经常比INT好一些，因为MEDIUMINT列使用的空间要少25%。
* 如果可能，声明列为NOT NULL。它使任何事情更快而且每列可以节省一位。注意如果在应用程序中确实需要NULL，应该毫无疑问使用它，只是避免 默认地在所有列上有它。
* 对于MyISAM表，如果没有任何变长列(VARCHAR、TEXT或BLOB列)，使用固定尺寸的记录格式。这比较快但是不幸地可能会浪费一些空间。参见[15.1.3节，“MyISAM表的存储格式”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "myisam-table-formats" \o "15.1.3. MyISAM Table Storage Formats)。即使你已经用CREATE选项让VARCHAR列ROW\_FORMAT=fixed，也可以提示想使用固定长度的行。
* 在MySQL/InnoDB中，InnoDB表使用更紧凑的存储格式。在以前版本的MySQL中，InnoDB记录包含一些冗余信息，例如列数目和每个列的长度，即使对于固定大小的列。默认情况，创建的表为紧凑格式(ROW\_FORMAT=COMPACT)。如果想要降级旧版本的MySQL/InnoDB，可以用ROW\_FORMAT=REDUNDANT要求旧的格式。
* 紧凑的InnoDB格式也改变了包含UTF-8数据的CHAR列的保存方式。在ROW\_FORMAT=REDUNDANT格式中，UTF-8 CHAR(*n*)占用3\**n*字节，假定UTF-8编码的字符的最大长度是3字节。许多语言可以主要用单字节UTF-8字符来编写，固定的存储长度通常会浪费空间。通过根据需要剥离尾部的空格，ROW\_FORMAT=COMPACT格式为这些列分配可变数量的n..3\*n字节。最小存储长度按顺序保存为n字节，以在典型情况下帮助更新。
* 每张表的主索引应该尽可能短。这使一行的识别容易而有效。
* 只创建你确实需要的索引。索引对检索有好处，但是当你需要快速存储东西时就变得糟糕。如果主要通过搜索列的组合来存取一个表，对它们做一个索引。第一个索引部分应该是最常用的列。如果从表中选择时总是使用许多列，应该首先以更多的副本使用列以获得更好的索引压缩。
* 如果很可能一个索引在头几个字符上有唯一的前缀，仅仅索引该前缀比较好。**MySQL**支持对一个字符列的最左边部分创建一个索引(参见[13.1.4节，“CREATE INDEX语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "create-index" \o "13.1.4. CREATE INDEX Syntax))。更短的索引会更快，不仅因为它们占较少的磁盘空间，而且因为它们将在索引缓存中提供更多的访问，因此磁盘搜索更少。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

·         在一些情形下，将一个经常被扫描的表分割为2个表是有益的。特别是如果它是一个动态格式的表，并且可能使用一个扫描表时能用来找出相关行的较小静态格式的表。

### 7.4.3. 列索引

所有MySQL列类型可以被索引。对相关列使用索引是提高SELECT操作性能的最佳途径。

根据存储引擎定义每个表的最大索引数和最大索引长度。参见[第15章：](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html)*[存储引擎和表类型](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \o "Chapter 15. Storage Engines and Table Types)*。所有存储引擎支持每个表至少16个索引，总索引长度至少为256字节。大多数存储引擎有更高的限制。

在索引定义中用*col\_name*(*length*)语法，你可以创建一个只使用CHAR或VARCHAR列的第1个*length*字符的索引。按这种方式只索引列值的前缀可以使索引文件小得多。

MyISAM和InnoDB存储引擎还支持对BLOB和TEXT列的索引。当索引一个BLOB或TEXT列时，你必须为索引指定前缀长度。例如：

CREATE TABLE test (blob\_col BLOB, INDEX(blob\_col(10)));

在MySQL 5.1中，对于MyISAM和InnoDB表，前缀可以达到1000字节长。请注意前缀的限制应以字节为单位进行测量，而CREATE TABLE语句中的前缀长度解释为字符数。当为使用多字节字符集的列指定前缀长度时一定要加以考虑。

还可以创建FULLTEXT索引。该索引可以用于全文搜索。只有MyISAM存储引擎支持FULLTEXT索引，并且只为CHAR、VARCHAR和TEXT列。索引总是对整个列进行，不支持局部(前缀)索引。详情参见[12.7节，“全文搜索功能”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/functions.html" \l "fulltext-search" \o "12.7. Full-Text Search Functions)。

也可以为空间列类型创建索引。只有MyISAM存储引擎支持空间类型。空间索引使用R-树。

默认情况MEMORY(HEAP)存储引擎使用hash索引，但也支持B-树索引。

### 7.4.4. 多列索引

MySQL可以为多个列创建索引。一个索引可以包括15个列。对于某些列类型，可以索引列的前缀(参见[7.4.3节，“列索引”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "indexes" \o "7.4.3. Column Indexes))。

多列索引可以视为包含通过连接索引列的值而创建的值的排序的数组。

MySQL按这样的方式使用多列索引：当你在WHERE子句中为索引的第1个列指定已知的数量时，查询很快，即使你没有指定其它列的值。

假定表具有下面的结构：

CREATE TABLE test (

    id INT NOT NULL,

    last\_name CHAR(30) NOT NULL,

    first\_name CHAR(30) NOT NULL,

    PRIMARY KEY (id),

    INDEX name (last\_name,first\_name)

);

name索引是一个对last\_name和first\_name的索引。索引可以用于为last\_name，或者为last\_name和first\_name在已知范围内指定值的查询。因此，name索引用于下面的查询：

SELECT \* FROM test WHERE last\_name='Widenius';

SELECT \* FROM test

    WHERE last\_name='Widenius' AND first\_name='Michael';

SELECT \* FROM test

    WHERE last\_name='Widenius'

    AND (first\_name='Michael' OR first\_name='Monty');

SELECT \* FROM test

    WHERE last\_name='Widenius'

    AND first\_name >='M' AND first\_name < 'N';

然而，name索引不用于下面的查询：

SELECT \* FROM test WHERE first\_name='Michael';

SELECT \* FROM test

    WHERE last\_name='Widenius' OR first\_name='Michael';

MySQL使用索引提高查询性能的方式将在[7.4.5节，“MySQL如何使用索引”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "mysql-indexes" \o "7.4.5. How MySQL Uses Indexes)中讨论。

### 7.4.5. MySQL如何使用索引

索引用于快速找出在某个列中有一特定值的行。不使用索引，MySQL必须从第1条记录开始然后读完整个表直到找出相关的行。表越大，花费的时间越多。如果表中查询的列有一个索引，MySQL能快速到达一个位置去搜寻到数据文件的中间，没有必要看所有数据。如果一个表有1000行，这比顺序读取至少快100倍。注意如果你需要访问大部分行，顺序读取要快得多，因为此时我们避免磁盘搜索。

大多数MySQL索引(PRIMARY KEY、UNIQUE、INDEX和FULLTEXT)在B树中存储。只是空间列类型的索引使用R-树，并且MEMORY表还支持hash索引。

字符串自动地压缩前缀和结尾空格。参见[13.1.4节，“CREATE INDEX语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "create-index" \o "13.1.4. CREATE INDEX Syntax)。

总的来说，按后面的讨论使用索引。本节最后描述hash索引(用于MEMORY表)的特征。

索引用于下面的操作：

·         快速找出匹配一个WHERE子句的行。

·         删除行。如果可以在多个索引中进行选择，MySQL通常使用找到最少行的索引。

·         当执行联接时，从其它表检索行。

·         对具体有索引的列*key\_col*找出MAX()或MIN()值。由预处理器进行优化，检查是否对索引中在key\_*col*之前发生所有关键字元素使用了WHERE *key\_part\_#* = *constant*。在这种情况下，MySQL为每个MIN()或MAX()表达式执行一次关键字查找，并用常数替换它。如果所有表达式替换为常量，查询立即返回。例如：

·                SELECT MIN(*key\_part2*),MAX(*key\_part2*)

·                    FROM *tbl\_name* WHERE *key\_part1*=10;

·         如果对一个可用关键字的最左面的前缀进行了排序或分组(例如，ORDER BY key\_part\_1,key\_part\_2)，排序或分组一个表。如果所有关键字元素后面有DESC，关键字以倒序被读取。参见[7.2.12节，“MySQL如何优化ORDER BY”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "order-by-optimization" \o "7.2.12. How MySQL Optimizes ORDER BY)。

·         在一些情况中，可以对一个查询进行优化以便不用查询数据行即可以检索值。如果查询只使用来自某个表的数字型并且构成某些关键字的最左面前缀的列，为了更快，可以从索引树检索出值。

·                SELECT *key\_part3* FROM *tbl\_name*

·                    WHERE *key\_part1*=1

假定你执行下面的SELECT语句：

mysql> **SELECT \* FROM *tbl\_name* WHERE col1=val1 AND col2=val2;**

如果col1和col2上存在一个多列索引，可以直接取出相应行。如果col1和col2上存在单列索引，优化器试图通过决定哪个索引将找到更少的行来找出更具限制性的索引并且使用该索引取行。

如果表有一个多列索引，优化器可以使用最左面的索引前缀来找出行。例如，如果有一个3列索引(col1,col2,col3)，则已经对(col1)、(col1,col2)和(col1,col2,col3)上的搜索进行了索引。

如果列不构成索引最左面的前缀，**MySQL**不能使用局部索引。假定有下面显示的SELECT语句。

SELECT \* FROM *tbl\_name* WHERE col1=val1;

SELECT \* FROM *tbl\_name* WHERE col1=val1 AND col2=val2;

SELECT \* FROM *tbl\_name* WHERE col2=val2;

SELECT \* FROM *tbl\_name* WHERE col2=val2 AND col3=val3;

如果 (col1，col2，col3)有一个索引，只有前2个查询使用索引。第3个和第4个查询确实包括索引的列，但(col2)和(col2，col3)不是 (col1，col2，col3)的最左边的前缀。

也可以在表达式通过=、>、>=、<、<=或者BETWEEN操作符使用B-树索引进行列比较。如果LIKE的参数是一个不以通配符开头的常量字符串，索引也可以用于LIKE比较。例如，下面的SELECT语句使用索引：

SELECT \* FROM *tbl\_name* WHERE *key\_col* LIKE 'Patrick%';

SELECT \* FROM *tbl\_name* WHERE *key\_col* LIKE 'Pat%\_ck%';

在第1个语句中，只考虑带'Patrick' <=key\_*col* < 'Patricl'的行。在第2个语句中，只考虑带'Pat' <=key\_*col* < 'Pau'的行。

下面的SELECT语句不使用索引：

SELECT \* FROM *tbl\_name* WHERE *key\_col* LIKE '%Patrick%';

SELECT \* FROM *tbl\_name* WHERE *key\_col* LIKE *other\_col*;

在第一条语句中，LIKE值以一个通配符字符开始。在第二条语句中，LIKE值不是一个常数。

如果使用... LIKE '%*string*%'并且*string*超过3个字符，MySQL使用Turbo Boyer-Moore算法初始化字符串的模式然后使用该模式来更快地进行搜索。

如果*col\_name*被索引，使用*col\_name* IS NULL的搜索将使用索引。

任何不跨越WHERE子句中的所有AND级的索引不用于优化查询。换句话说，为了能够使用索引，必须在每个AND组中使用索引前缀。

下面的WHERE子句使用索引：

... WHERE *index\_part1*=1 AND *index\_part2*=2 AND *other\_column*=3

    /\* *index* = 1 OR *index* = 2 \*/

... WHERE *index*=1 OR A=10 AND *index*=2

    /\* optimized like "*index\_part1*='hello'" \*/

... WHERE *index\_part1*='hello' AND *index\_part3*=5

    /\* Can use index on *index1* but not on *index2* or *index3* \*/

... WHERE *index1*=1 AND *index2*=2 OR *index1*=3 AND *index3*=3;

下面的WHERE子句不使用索引：

    /\* *index\_part1* is not used \*/

... WHERE *index\_part2*=1 AND *index\_part3*=2

    /\*  Index is not used in both parts of the WHERE clause  \*/

... WHERE *index*=1 OR A=10

    /\* No index spans all rows  \*/

... WHERE *index\_part1*=1 OR *index\_part2*=10

有时MySQL不使用索引，即使有可用的索引。一种情形是当优化器估计到使用索引将需要MySQL访问表中的大部分行时。(在这种情况下，表扫描可能会更快些，因为需要的搜索要少）。然而，如果此类查询使用LIMIT只搜索部分行，MySQL则使用索引，因为它可以更快地找到几行并在结果中返回。

Hash索引还有一些其它特征：

·         它们只用于使用=或<=>操作符的等式比较(但很快)。它们用于比较 操作符，例如发现范围值的<。

·         优化器不能使用hash索引来加速ORDER BY操作。(该类索引不能用来按顺序搜索下一个条目）。

·         MySQL不能确定在两个值之间大约有多少行(这被范围优化器用来确定使用哪个索引)。如果你将一个MyISAM表改为hash-索引的MEMORY表，会影响一些查询。

·         只能使用整个关键字来搜索一行。(用B-树索引，任何关键字的最左面的前缀可用来找到行）。

### 7.4.6. MyISAM键高速缓冲

[7.4.6.1. 共享键高速缓冲访问](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "shared-key-cache)

[7.4.6.2. 多键高速缓冲](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "multiple-key-caches)

[7.4.6.3. 中点插入策略](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "midpoint-insertion)

[7.4.6.4. 索引预加载](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "index-preloading)

[7.4.6.5. 键高速缓冲块大小](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "key-cache-block-size)

[7.4.6.6. 重构键高速缓冲](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "key-cache-restructuring)

为了使硬盘I/O最小化，MyISAM存储引擎使用一个被许多数据库管理系统使用的策略。它使用一个缓存机制将经常访问的表锁在内存中：

·         对于索引块，维护一个称之为*键高速缓冲*(或键高速缓冲区)的特殊结构。该结构包含大量块缓存区，其中放置了最常用的索引块。

·         对于数据块，MySQL不使用特殊缓存。而使用原生的操作系统文件系统的缓存。

本节首先描述了MyISAM键高速缓冲的基本操作。然后讨论了提高 键高速缓冲性能并使你更好地控制缓存操作的最新的更改：

·         多个线程可以并行访问缓存。

·         可以设置多个键高速缓冲，并将表索引指定给具体缓存。

可以使用key\_buffer\_size系统变量控制 键高速缓冲的大小。如果该变量设置为零，不使用键高速缓冲。如果key\_buffer\_size值太小不能分配最小数量的块缓存区(8)，也不使用 键高速缓冲。

如果键高速缓冲不工作，只使用操作系统提供的原生文件系统缓存区访问索引文件。(换句话说，使用与表数据块相同的策略表来访问索引块）。

索引块是一个连续的访问MyISAM索引文件的单位。通常一个索引块的大小等于索引B-树节点的大小。(在硬盘上使用B-树数据结构表示索引。树底部的节点为叶子节点。叶子节点上面的节点为非叶子节点）。

键高速缓冲结构中的所有块缓存区大小相同。该大小可以等于、大于或小于表索引块的大小。通常这两个值中的一个是另一个的几倍。

当必须访问表索引块中的数据时，服务器首先检查是否它可以用于键高速缓冲中的某些块缓存区。如果适用，服务器访问键高速缓冲中的数据而不是硬盘上的数据。也就是说，从缓存读取或写入缓存，而不是从硬盘读写。否则，服务器选择一个包含一个不同的表索引块的缓存块缓存区，并用需要的表索引块的拷贝替换那里的数据。一旦新的索引块位于缓存中，可以访问索引数据。

如果用于替换的块已经被修改了，块被视为“脏了”。在这种情况下，在替换前，其内容被刷新到它来自的表索引。

通常服务器遵从LRU(最近最少使用)策略：当选择一个块用于替换时，它选择最近最少使用的索引块。为了使该选择更容易， 键高速缓冲模块维护所有使用的块的专门队列(LRU链)。当访问块时，它被放到队列最后。当块需要替换时，队列开头的块是最近最少使用的块，并成为第1个候选者。

#### 7.4.6.1. 共享键高速缓冲访问

在以下条件下，线程可以同时访问键高速缓冲缓存区：

·         没有被更新的缓存区可以被多个线程访问。

·         正被更新的缓存区让需要使用它的线程等待直到更新完成。

·         多个线程可以发起请求替换缓存块，只要它们不彼此干扰(也就是说，只要它们需要不同的索引块，并且使不同的缓存块被替换)。

对键高速缓冲的共享访问允许服务器大大提高吞吐量。

#### 7.4.6.2. 多键高速缓冲

对键高速缓冲的共享访问可以提高性能但不能完全消除线程之间的竟争。它们仍然竞争对键高速缓冲缓存区的访问进行管理的控制结构。为了进一步降低 键高速缓冲访问竟争，MySQL 5.1还提供了多个键高速缓冲，允许你为不同的键高速缓冲分配不同的表索引。

有多个键高速缓冲时，当为给定的MyISAM表处理查询时，服务器必须知道使用哪个缓存。默认情况，所有MyISAM表索引被缓存到默认 键高速缓冲中。要想为具体键高速缓冲分配表索引，应使用CACHE INDEX语句(参见[13.5.5.1节，“CACHE INDEX语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "cache-index" \o "13.5.5.1. CACHE INDEX Syntax))。

例如，下面的语句将表t1、t2和t3的索引分配给名为hot\_cache的 键高速缓冲：

mysql> **CACHE INDEX t1, t2, t3 IN hot\_cache;**

+---------+--------------------+----------+----------+

| Table   | Op                 | Msg\_type | Msg\_text |

+---------+--------------------+----------+----------+

| test.t1 | assign\_to\_keycache | status   | OK       |

| test.t2 | assign\_to\_keycache | status   | OK       |

| test.t3 | assign\_to\_keycache | status   | OK       |

+---------+--------------------+----------+----------+

可以用SET GLOBAL参数设置语句或使用服务器启动选项设置在CACHE INDEX语句中引用的键高速缓冲的大小来创建键高速缓冲。例如：

mysql> **SET GLOBAL keycache1.key\_buffer\_size=128\*1024;**

要想删除键高速缓冲，将其大小设置为零：

mysql> **SET GLOBAL keycache1.key\_buffer\_size=0;**

请注意不能删除默认键高速缓冲。删除默认键高速缓冲的尝试将被忽略：

mysql> **set global key\_buffer\_size = 0;**

mysql> **show variables like 'key\_buffer\_size';**

+-----------------+---------+

| Variable\_name   | Value   |

+-----------------+---------+

| key\_buffer\_size | 8384512 |

+-----------------+---------+

键高速缓冲变量是结构式系统变量，有一个名和组件。对于keycache1.key\_buffer\_size，keycache1是缓存变量名，key\_buffer\_size是缓存组件。关于引用结构式 键高速缓冲系统变量所使用的语法的描述，参见[9.4.1节，“结构式系统变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/language-structure.html" \l "structured-system-variables" \o "9.4.1. Structured System Variables)

默认情况下，表索引被分配给服务器启动时创建的主要(默认)键高速缓冲。当 键高速缓冲被删除后，所有分配给它的索引被重新分配给默认键高速缓冲。

对于一个忙的服务器，我们建议采用使用三个键高速缓冲的策略：

·         占用为所有键高速缓冲分配的空间的20%的“热”键高速缓冲。该缓存用于频繁用于搜索但没有更新的表。

·         占用为所有键高速缓冲分配的空间的20%的“冷”键高速缓冲。该缓存用于中等大小、大量修改的表，例如临时表。

·         占用键高速缓冲空间的20%的“温”键高速缓冲。使用它作为默认 键高速缓冲，默认情况被所有其它表使用。

使用3个键高速缓冲有好处的一个原因是对一个键高速缓冲结构的访问不会阻挡对其它的访问。访问分配给一个缓存的表的查询不会与访问分配给其它缓存的表的查询竞争。由于其它原因也会提高性能：

·         热缓存只用于检索查询，因此其内容决不会被修改。结果是，无论何时需要从硬盘上拉入索引块，选择用于替换的缓存块的内容不需要先刷新。

·         对于分配给热缓存的索引，如果没有查询需要索引扫描，很有可能对应索引B-树的非叶子节点的索引块仍然在缓存中。

·         当更新的节点位于缓存中并且不需要先从硬盘读入时，为临时表频繁执行的更新操作会执行得更快。如果临时表的索引的大小可以与冷键高速缓冲相比较，很可能更新的节点位于缓存中。

CACHE INDEX在一个表和 键高速缓冲之间建立一种联系，但每次服务器重启时该联系被丢失。如果你想要每次服务器重启时该联系生效，一个发办法是使用选项文件：包括配置 键高速缓冲的变量设定值，和一个init-file选项用来命名包含待执行的CACHE INDEX语句的一个文件。例如：

key\_buffer\_size = 4G

hot\_cache.key\_buffer\_size = 2G

cold\_cache.key\_buffer\_size = 2G

init\_file=/path/to/data-directory/mysqld\_init.sql

每次服务器启动时执行mysqld\_init.sql中的语句。该文件每行应包含一个SQL语句。下面的例子分配几个表，分别对应hot\_cache和cold\_cache：

CACHE INDEX a.t1, a.t2, b.t3 IN hot\_cache

CACHE INDEX a.t4, b.t5, b.t6 IN cold\_cache

#### 7.4.6.3. 中点插入策略

默认情况，键高速缓冲管理系统采用LRU策略选择要收回的键高速缓冲块，但它也支持更复杂的方法，称之为“中点插入策略”。

当使用中点插入策略时，LRU链被分为两个部分：一条热子链和一条温子链。两部分之间的划分点不固定，但 键高速缓冲管理系统关注温部分不“太短”，总是包含至少key\_cache\_division\_limit比例的 键高速缓冲块。key\_cache\_division\_limit是结构式 键高速缓冲变量的一个组件，因此其值是一个可以根据每个缓存进行设置的参数。

当一个索引块从表中读入键高速缓冲，它被放入温子链的末端。经过一定量的访问后(访问块)，它被提升给热子链。目前，需要用来提升一个块(3)的访问次数与所有索引块的相同。

提升到热子链的块被放到子链的末端。块然后在该子链中循环。如果块在子链的开头停留足够长的时间，它被降到温链。该时间由键高速缓冲key\_cache\_age\_threshold组件的值确定。

对于包含*N*个块的 键高速缓冲，阈值表示，热子链开头的没有在最后*N* \*key\_cache\_age\_threshold/100次访问中被访问的块将被移动到温子链开头。该块然后变为第1个挤出的候选者，因为替换的块总是来自温子链的开头。

中点插入策略允许你将更有价值的块总是在缓存中。如果你想使用简单的LRU策略，使key\_cache\_division\_limit值保持其默认值100。

若执行的查询要求索引扫描有效推出所有索引块对应有数值的高级B-树节点的缓存，中点插入策略可以帮助提高性能。要想避免，必须使用中点插入策略，而key\_cache\_division\_limit设置为远小于100。然后在索引扫描操作过程中，有数值的经常访问的节点被保留在热子链中。

#### 7.4.6.4. 索引预加载

如果键高速缓冲内有足够的块以容纳整个索引的块，或者至少容纳对应其非叶节点的块，则在使用前，预装含索引块的键高速缓冲很有意义。预装可以以更有效的方式将表索引块放入 键高速缓冲缓存区中：通过顺序地从硬盘读取索引块。

不进行预装，块仍然根据查询需要放入键高速缓冲中。尽管块将仍然在缓存中（因为有足够的缓存区保存它们），它们以随机方式从硬盘上索取，而不是以顺序方式。

要想将索引预装到缓存中，使用LOAD INDEX INTO CACHE语句。例如，下面的语句可以预装表t1和t2索引的节点(索引块)：

mysql> **LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;**

+---------+--------------+----------+----------+

| Table   | Op           | Msg\_type | Msg\_text |

+---------+--------------+----------+----------+

| test.t1 | preload\_keys | status   | OK       |

| test.t2 | preload\_keys | status   | OK       |

+---------+--------------+----------+----------+

IGNORE LEAVES修改器只允许预装索引非叶节点所用的块。这样，上述的语句预装t1中的所有索引块，但只预装t2中的非叶节点对应的块。

如果已经使用CACHE INDEX语句为一个索引分配了一个键高速缓冲，预装可以将索引块放入该缓存。否则，索引被装入默认键高速缓冲。

#### 7.4.6.5. 键高速缓冲块大小

可以使用key\_cache\_block\_size变量为具体的 键高速缓冲指定块缓存区的大小。这样允许为索引文件调节I/O操作的性能。

当读缓存区的大小等于原生操作系统I/O缓存区的大小时，可以获得I/O操作的最佳性能。但是将关键字节点的大小设置为等于I/O缓存区的大小并不总是能保证最佳整体性能。当读取大的叶节点时，服务器读入大量的不需要的数据，结果防止读入其它叶子的节点。

目前，你不能控制表内索引块的大小。该大小由服务器在创建.MYI索引文件时设置，取决于表定义中索引的关键字大小。在大多数情况下，它被设置为与I/O缓存区大小相等。

#### 7.4.6.6. 重构键高速缓冲

键高速缓冲可以通过更新其参数值随时重新构建。例如：

mysql> **SET GLOBAL cold\_cache。key\_buffer\_size=4\*1024\*1024；**

如果你为key\_buffer\_size或key\_cache\_block\_size键高速缓冲组件分配的值与组件当前的值不同，服务器将毁掉缓存的旧结构并根据新值创建一个新的。如果缓存包含任何脏的块，服务器在销毁前将它们保存到硬盘上并重新创建缓存。如果你设置其它 键高速缓冲参数，则不会发生重新构建。

当重新构建键高速缓冲时，服务器首先将任何脏缓存区的内容刷新到硬盘上。之后，缓存内容不再需要。然而，重新构建并不阻塞需要使用分配给缓存的索引的查询。相反，服务器使用原生文件系统缓存直接访问表索引。文件系统缓存不如使用 键高速缓冲有效，因此尽管查询可以执行，但速度会减慢。缓存被重新构建后，它又可以缓存分配给它的索引了，并且索引不再使用文件系统缓存。

### 7.4.7. MyISAM索引统计集合

存储引擎搜集优化器使用的表的统计信息。表统计基于数数值组，其中数数值组是一系列有相同的关键字前缀值的记录。对于优化器，重要的统计即为数数值组的平均大小。

MySQL用下述方式使用平均数数值组：

·         估计必须为每个ref访问读取多少行

·         估计部分联接将产生多少行；也就是说，下述形式的操作将产生的行数：

·                  (...) JOIN *tbl\_name* ON *tbl\_name*.*key* = *expr*

随着索引的平均数数值组大小的增加，索引将更没有用，因为每个查找的平均行数增加：为了让索引有利于优化目的，最好是每个索引值对应表内的少量行数。当某个给定的索引值产生较多行时，索引更加没有用，MySQL更不可能使用它。

平均数数值组大小与表的集的势相关，即数数值组的数目。SHOW INDEX语句显示集的势值（基于*N*/*S*），其中*N*是表内的记录数，*S*是平均数数值组大小。该比例产生表内数数值组的大约数。

对于基于<=>比较 操作符的联接，NULL并不视为与任何其它值不同：NULL <=> NULL，正如对于其它*N* ，*N* <=> *N*。

然而，对于基于=操作符的联接，NULL与非NULL值不同：当*expr1*或*expr2*(或两者)为NULL时，*expr1* = *expr2*不为真。这样影响比较形式*tbl\_name.key* = *expr*的ref访问：如果*expr*当前的值为NULL，MySQL不会访问表，因为比较不能为真。

对于=比较，表内有多少NULL值并不重要。为了优化目的，相关值为非NULL数值组的平均大小。然而，MySQL目前不允许搜集或使用该平均大小。

对于MyISAM表，你可以使用myisam\_stats\_method系统变量部分控制表统计信息的搜集。该变量有两个可能的不同值，如下所示：

·         当myisam\_stats\_method为nulls\_equal时，所有NULL值被视为相等的(也就是说，它们都形成一个数值组)。

如果NULL数值组大小远大于平均非NULL数值组大小，该方法向上倾斜平均数数值组大小。这样使索引对于优化器来说比它实际为查找非NULL值的联接更加没有用。结果是，nulls\_equal方法会使优化器进行ref访问时本应使用索引而没有使用。

·         当myisam\_stats\_method为nulls\_unequal时，NULL值不视为相同。相反，每个NULL值形成一个单独的数值组，大小为1。

如果你有许多NULL值，该方法向下倾斜平均数数值组大小。如果平均非NULL数值组较大，统计大小为1的每个组的NULL值会使优化器过高估计查找非NULL值的联接的索引值。结果是，当其它方法会更好时，nulls\_unequal方法会使优化器为ref查找使用该索引。

如果你要使用许多使用<=>而不是=的联接，在比较过程中NULL值并不特殊，一个NULL等于另一个NULL。在这种情况下，nulls\_equal是合适的统计方法。

myisam\_stats\_method系统变量有全局和会话值。设置全局值会影响MyISAM 为所有MyISAM表的统计的搜集。设置会话值只影响当前客户连接的统计的搜集。这说明你可以强制用给定的方法重新生成表的统计的搜集，而不需要因为设置myisam\_stats\_method的会话值而影响其它客户。

可以使用下面任一方法来重新生成表的统计信息：

·         设置myisam\_stats\_method，然后执行CHECK TABLE语句

·         执行**myisamchk --stats\_method=method\_name --analyze**

·         更改表，使其统计信息不为最新(例如，插入一行然后删除它)，然后设置myisam\_stats\_method并执行ANALYZE TABLE语句

使用myisam\_stats\_method的一些警告：

你可以强制显式搜集表的统计信息，如上所述。然而，MySQL也可以自动搜集统计信息。例如，如果在为表执行语句的过程中，一些语句修改了表，MySQL可以搜集统计信息。(例如，大批插入或删除，或者执行ALTER TABLE语句时可能发生）。如果发生，使用myisam\_stats\_method此时所有的值搜集统计信息。这样，如果你使用一个方法搜集统计信息，但当后面自动搜集一个表的统计信息时myisam\_stats\_method被设置为另一个方法，将使用其它方法。

对于给定的MyISAM表，还不能说出使用哪个方法来产生统计信息。

myisam\_stats\_method只适合MyISAM表。其它存储引擎只有一个方法来搜集表的统计信息。通常它接近于nulls\_equal方法。

### 7.4.8. MySQL如何计算打开的表

当运行**mysqladmin status**时，将看见象这样的一些东西：

Uptime: 426 Running threads: 1 Questions: 11082

Reloads: 1 Open tables: 12

如果你仅有6个表，Open tables值为12可能有点令人困惑。

**MySQL**是多线程的，因此许多客户可以同时在同一个表上进行查询。为了使多个客户线程在同一个表上有不同状态的问题减到最小，表被每个并发进程独立地打开。这样需要额外的内存但一般会提高性能。对于MyISAM表，数据文件需要为每个打开表的客户提供一个额外的文件描述符。(索引文件描述符在所有线程之间共享)。

下一节中提供了该主题的更多的信息。参见[7.4.9节，“MySQL如何打开和关闭表”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "table-cache" \o "7.4.9. How MySQL Opens and Closes Tables)。

### 7.4.9. MySQL如何打开和关闭表

table\_cache、max\_connections和max\_tmp\_tables系统变量影响服务器保持打开的文件的最大数量。如果你增加这些值其中的一个或两个，会遇到操作系统为每个进程打开文件描述符的数量强加的限制。许多操作系统允许你增加打开的文件的限制，尽管该方法随系统的不同而不同。查阅操作系统文档以确定是否可以增加限制以及如何操作。

table\_cache与max\_connections有关。例如，对于200个并行运行的连接，应该让表的缓存至少有200 \* *N*，这里*N*是可以执行的查询的一个联接中表的最大数量。还需要为临时表和文件保留一些额外的文件描述符。

确保操作系统可以处理table\_cache设置所指的打开的文件描述符的数目。如果table\_cacheis设得太高，MySQL可能为文件描述符耗尽资源并拒绝连接，不能执行查询，并且很不可靠。还必须考虑到MyISAM存储引擎需要为每个打开的表提供两个文件描述符。可以在**mysqld\_safe**中使用--open-files-limit启动选项来增加MySQL适用的文件描述符的数量。参见[A.2.17节，“文件未找到”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/problems.html" \l "not-enough-file-handles" \o "A.2.17. File Not Found)。

打开表的缓存可以保持在table\_cache条。 默认为64；可以用**mysqld**的--table\_cache选项来更改。请注意 MySQL可以临时打开更多的 表以执行查询。

在下面的条件下，未使用的表将被关闭并从表缓存中移出：

·         当缓存满了并且一个线程试图打开一个不在缓存中的表时。

·         当缓存包含超过table\_cache个条目，并且缓存中的表不再被任何线程使用。

·         当表刷新操作发生。当执行FLUSH TABLES语句或执行**mysqladmin flush-tables**或**mysqladmin refresh**命令时会发生。

当表缓存满时，服务器使用下列过程找到一个缓存入口来使用：

·         当前未使用的表被释放，以最近最少使用顺序。

·         如果缓存满了并且没有表可以释放，但是一个新表需要打开，缓存必须临时被扩大。

如果缓存处于一个临时扩大状态并且一个表从在用变为不在用状态，它被关闭并从缓存中释放。

对每个并发访问打开一个表。这意味着，如果2个线程访问同一个表或在同一个查询中访问表两次(例如，将表连接为自身时)，表需要被打开两次。每个并行的打开要求在表缓存中有一个条目。任何表的第一次打开占2个文件描述符：一个用于数据文件另一个用于索引文件。表的每一次额外使用仅占一个数据文件的文件描述符。索引文件描述符在所有线程之间共享。

如果你正用HANDLER *tbl\_name* OPEN语句打开一个表，将为该线程专门分配一个表。该表不被其它线程共享，只有线程调用HANDLER *tbl\_name* CLOSE或线程终止后才被关闭。表关闭后，被拉回表缓存中(如果缓存不满)。参见[13.2.3节，“HANDLER语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "handler" \o "13.2.3. HANDLER Syntax)。

可以通过检查**mysqld**的状态变量Opened\_tables确定表缓存是否太小：

mysql> **SHOW STATUS LIKE 'Opened\_tables';**

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| Opened\_tables | 2741  |

+---------------+-------+

如果值很大，即使你没有发出许多FLUSH TABLES语句，也应增加表缓存的大小。参见[5.3.3节，“服务器系统变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-system-variables" \o "5.3.3. Server System Variables)和[5.3.4节，“服务器状态变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-status-variables" \o "5.3.4. Server Status Variables)。

### 7.4.10. 在同一个数据库中创建多个表的缺陷

如果在同一个数据库目录中有许多MyISAM表，打开、关闭和创建操作将会很慢。如果对许多不同的表执行SELECT语句，当表缓存满时，将有一点开销，因为对每个必须打开的表，另外一个必须被关闭。可以通过使表缓存更大些来减少这个开销。

## 7.5. 优化MySQL服务器

[7.5.1. 系统因素和启动参数的调节](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "system)

[7.5.2. 调节服务器参数](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters)

[7.5.3. 控制查询优化器的性能](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "controlling-optimizer)

[7.5.4. 编译和链接怎样影响MySQL的速度](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "compile-and-link-options)

[7.5.5. MySQL如何使用内存](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "memory-use)

[7.5.6. MySQL如何使用DNS](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "dns)

### 7.5.1. 系统因素和启动参数的调节

我们从系统级因素开始，因为必须尽早地进行部分决策以获得较大性能。在其它情况下，快速浏览该节就足够了。但是，了解一下更改该层次的参数能够获得多少性能提高是很有意义的。

使用的操作系统很重要。为了更好地使用多CPU机器，应使用Solaris(因为其线程工作得很好)或Linux(因为2.4和以后的内核有很好的SMP支持)。请注意默认情况旧的Linux内核有一个2GB的文件大小限制。如果有这样的一个内核并且需要文件大于2GB，应得到ext2文件系统的大文件支持(LFS)补丁。其它文件系统例如ReiserFS和XFS没有此2GB限制。

将MySQL用于生产前，我们建议你在想用的平台上对它进行测试。

其它技巧：

·         如果有足够的RAM，可以移除所有的交换设备。有些操作系统即使有自由内存也使用交换设备。

·         使用--skip-external-locking MySQL选项以避免外部锁定。该选项默认开启。

请注意只要你只运行一个服务器，--skip-external-locking选项不影响MySQL的功能。只要记住运行**myisamchk**之前关闭服务器(或锁定并刷新相关表)。在一些系统上该选项是强制的，因为在任何情况下外部锁定均不工作。

不能使用--skip-external-locking的唯一情况是对相同的数据运行多个MySQL服务器(非客户)的情况，或者如果没有事先告诉服务器刷新并锁定一个表即运行**myisamchk**来检查(非修复)该表。请注意一般不建议使用多个MySQL服务器来并行访问相同的数据，除了使用MySQL Cluster时。

即使使用--skip-external-locking，仍然可以使用LOCK TABLES和UNLOCK TABLES。

### 7.5.2. 调节服务器参数

可以用这个命令得到**mysqld**服务器 默认缓存区的大小：

shell> **mysqld --verbose --help**

这个命令生成所有**mysqld**选项和可配置变量的列表。输出包括 默认值并且看上去象这样：

help                              TRUE

abort-slave-event-count           0

allow-suspicious-udfs             FALSE

auto-increment-increment          1

auto-increment-offset             1

automatic-sp-privileges           TRUE

basedir                           /home/jon/bin/mysql/

bdb                               FALSE

bind-address                      (No default value)

character-set-client-handshake    TRUE

character-set-server              latin1

character-sets-dir                /home/jon/bin/mysql/share/mysql/charsets/

chroot                            (No default value)

collation-server                  latin1\_swedish\_ci

completion-type                   0

concurrent-insert                 1

console                           FALSE

datadir                           /home/jon/bin/mysql/var/

default-character-set             latin1

default-collation                 latin1\_swedish\_ci

default-time-zone                 (No default value)

disconnect-slave-event-count      0

enable-locking                    FALSE

enable-pstack                     FALSE

engine-condition-pushdown         FALSE

external-locking                  FALSE

gdb                               FALSE

large-pages                       FALSE

init-connect                      (No default value)

init-file                         (No default value)

init-slave                        (No default value)

innodb                            TRUE

innodb\_checksums                  TRUE

innodb\_data\_home\_dir              (No default value)

innodb\_doublewrite                TRUE

innodb\_fast\_shutdown              1

innodb\_file\_per\_table             FALSE

innodb\_flush\_log\_at\_trx\_commit    1

innodb\_flush\_method               (No default value)

innodb\_locks\_unsafe\_for\_binlog    FALSE

innodb\_log\_arch\_dir               (No default value)

innodb\_log\_group\_home\_dir         (No default value)

innodb\_max\_dirty\_pages\_pct        90

innodb\_max\_purge\_lag              0

innodb\_status\_file                FALSE

innodb\_table\_locks                TRUE

innodb\_support\_xa                 TRUE

isam                              FALSE

language                          /home/jon/bin/mysql/share/mysql/english

local-infile                      TRUE

log                               /home/jon/bin/mysql/var/master1.log

log-bin                           /home/jon/bin/mysql/var/master1

log-bin-index                     (No default value)

log-bin-trust-routine-creators    FALSE

log-error                         /home/jon/bin/mysql/var/master1.err

log-isam                          myisam.log

log-queries-not-using-indexes     FALSE

log-short-format                  FALSE

log-slave-updates                 FALSE

log-slow-admin-statements         FALSE

log-slow-queries                  (No default value)

log-tc                            tc.log

log-tc-size                       24576

log-update                        (No default value)

log-warnings                      1

low-priority-updates              FALSE

master-connect-retry              60

master-host                       (No default value)

master-info-file                  master.info

master-password                   (No default value)

master-port                       3306

master-retry-count                86400

master-ssl                        FALSE

master-ssl-ca                     (No default value)

master-ssl-capath                 (No default value)

master-ssl-cert                   (No default value)

master-ssl-cipher                 (No default value)

master-ssl-key                    (No default value)

master-user                       test

max-binlog-dump-events            0

memlock                           FALSE

myisam-recover                    OFF

ndbcluster                        FALSE

ndb-connectstring                 (No default value)

ndb-mgmd-host                     (No default value)

ndb-nodeid                        0

ndb-autoincrement-prefetch-sz     32

ndb-distibution                   KEYHASH

ndb-force-send                    TRUE

ndb\_force\_send                    TRUE

ndb-use-exact-count               TRUE

ndb\_use\_exact\_count               TRUE

ndb-shm                           FALSE

ndb-optimized-node-selection      TRUE

ndb-cache-check-time              0

ndb-index-stat-enable             TRUE

ndb-index-stat-cache-entries      32

ndb-index-stat-update-freq        20

new                               FALSE

old-alter-table                   FALSE

old-passwords                     FALSE

old-style-user-limits             FALSE

pid-file                          /home/jon/bin/mysql/var/hostname.pid1

port                              3306

relay-log                         (No default value)

relay-log-index                   (No default value)

relay-log-info-file               relay-log.info

replicate-same-server-id          FALSE

report-host                       (No default value)

report-password                   (No default value)

report-port                       3306

report-user                       (No default value)

rpl-recovery-rank                 0

safe-user-create                  FALSE

secure-auth                       FALSE

server-id                         1

show-slave-auth-info              FALSE

skip-grant-tables                 FALSE

skip-slave-start                  FALSE

slave-load-tmpdir                 /tmp/

socket                            /tmp/mysql.sock

sporadic-binlog-dump-fail         FALSE

sql-mode                          OFF

symbolic-links                    TRUE

tc-heuristic-recover              (No default value)

temp-pool                         TRUE

timed\_mutexes                     FALSE

tmpdir                            (No default value)

use-symbolic-links                TRUE

verbose                           TRUE

warnings                          1

back\_log                          50

binlog\_cache\_size                 32768

bulk\_insert\_buffer\_size           8388608

connect\_timeout                   5

date\_format                       (No default value)

datetime\_format                   (No default value)

default\_week\_format               0

delayed\_insert\_limit              100

delayed\_insert\_timeout            300

delayed\_queue\_size                1000

expire\_logs\_days                  0

flush\_time                        0

ft\_max\_word\_len                   84

ft\_min\_word\_len                   4

ft\_query\_expansion\_limit          20

ft\_stopword\_file                  (No default value)

group\_concat\_max\_len              1024

innodb\_additional\_mem\_pool\_size   1048576

innodb\_autoextend\_increment       8

innodb\_buffer\_pool\_awe\_mem\_mb     0

innodb\_buffer\_pool\_size           8388608

innodb\_concurrency\_tickets        500

innodb\_file\_io\_threads            4

innodb\_force\_recovery             0

innodb\_lock\_wait\_timeout          50

innodb\_log\_buffer\_size            1048576

innodb\_log\_file\_size              5242880

innodb\_log\_files\_in\_group         2

innodb\_mirrored\_log\_groups        1

innodb\_open\_files                 300

innodb\_sync\_spin\_loops            20

innodb\_thread\_concurrency         20

innodb\_commit\_concurrency         0

innodb\_thread\_sleep\_delay         10000

interactive\_timeout               28800

join\_buffer\_size                  131072

key\_buffer\_size                   8388600

key\_cache\_age\_threshold           300

key\_cache\_block\_size              1024

key\_cache\_division\_limit          100

long\_query\_time                   10

lower\_case\_table\_names            0

max\_allowed\_packet                1048576

max\_binlog\_cache\_size             4294967295

max\_binlog\_size                   1073741824

max\_connect\_errors                10

max\_connections                   100

max\_delayed\_threads               20

max\_error\_count                   64

max\_heap\_table\_size               16777216

max\_join\_size                     4294967295

max\_length\_for\_sort\_data          1024

max\_relay\_log\_size                0

max\_seeks\_for\_key                 4294967295

max\_sort\_length                   1024

max\_tmp\_tables                    32

max\_user\_connections              0

max\_write\_lock\_count              4294967295

multi\_range\_count                 256

myisam\_block\_size                 1024

myisam\_data\_pointer\_size          6

myisam\_max\_extra\_sort\_file\_size   2147483648

myisam\_max\_sort\_file\_size         2147483647

myisam\_repair\_threads             1

myisam\_sort\_buffer\_size           8388608

myisam\_stats\_method               nulls\_unequal

net\_buffer\_length                 16384

net\_read\_timeout                  30

net\_retry\_count                   10

net\_write\_timeout                 60

open\_files\_limit                  0

optimizer\_prune\_level             1

optimizer\_search\_depth            62

preload\_buffer\_size               32768

query\_alloc\_block\_size            8192

query\_cache\_limit                 1048576

query\_cache\_min\_res\_unit          4096

query\_cache\_size                  0

query\_cache\_type                  1

query\_cache\_wlock\_invalidate      FALSE

query\_prealloc\_size               8192

range\_alloc\_block\_size            2048

read\_buffer\_size                  131072

read\_only                         FALSE

read\_rnd\_buffer\_size              262144

div\_precision\_increment           4

record\_buffer                     131072

relay\_log\_purge                   TRUE

relay\_log\_space\_limit             0

slave\_compressed\_protocol         FALSE

slave\_net\_timeout                 3600

slave\_transaction\_retries         10

slow\_launch\_time                  2

sort\_buffer\_size                  2097144

sync-binlog                       0

sync-frm                          TRUE

sync-replication                  0

sync-replication-slave-id         0

sync-replication-timeout          10

table\_cache                       64

table\_lock\_wait\_timeout           50

thread\_cache\_size                 0

thread\_concurrency                10

thread\_stack                      196608

time\_format                       (No default value)

tmp\_table\_size                    33554432

transaction\_alloc\_block\_size      8192

transaction\_prealloc\_size         4096

updatable\_views\_with\_limit        1

wait\_timeout                      28800

如果有一个**mysqld**服务器正在运行，通过连接它并执行这个命令，可以看到实际上使用的变量的值：

mysql> **SHOW VARIABLES;**

还可以通过下面的语句看到运行服务器的统计和状态指标：

mysql>SHOW STATUS；

使用**mysqladmin**还可以获得系统变量和状态信息：

shell> **mysqladmin variables**

shell> **mysqladmin extended-status**

关于所有系统和状态变量的完全描述参见[5.3.3节，“服务器系统变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-system-variables" \o "5.3.3. Server System Variables)和[5.3.4节，“服务器状态变量”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/database-administration.html" \l "server-status-variables" \o "5.3.4. Server Status Variables)。

MySQL使用完全可以升级的算法，因此通常运行时可以用很少的内存。然而，通常情况若给MySQL更多的内存性能会更好。

当调节MySQL服务器时，要配置的两个最重要的变量是key\_buffer\_size和table\_cache。在试图更改其它变量前你应先确信已经适当地配置了这些变量。

下面的例子显示了部分典型的不同的运行时配置的变量值。

·         如果至少有256MB内存和许多表，想要在中等数量的客户时获得最大性能，应使用：

·                shell> **mysqld\_safe --key\_buffer\_size=64M --table\_cache=256 \**

·                           **--sort\_buffer\_size=4M --read\_buffer\_size=1M &**

·         如果只有128MB内存和少量表，但仍然要进行大量的排序，可以使用：

·                shell> **mysqld\_safe --key\_buffer\_size=16M --sort\_buffer\_size=1M**

如果有许多并行连接，交换问题会发生，除非**mysqld**已经配置成为每个连接分配很少的内存。如果有足够的内存用于所有连接，**mysqld**会执行得更好。

·         对于少量内存和大量连接，应使用：

·                shell> **mysqld\_safe --key\_buffer\_size=512K --sort\_buffer\_size=100K \**

·                           **--read\_buffer\_size=100K &**

或甚至为：

shell> **mysqld\_safe --key\_buffer\_size=512K --sort\_buffer\_size=16K \**

**--table\_cache=32 --read\_buffer\_size=8K \**

**--net\_buffer\_length=1K &**

如果正对远远大于可用内存的表执行GROUP BY或ORDER BY操作，应增加read\_rnd\_buffer\_size的值以加速排序操作后面的行读取。

如果已经安装了MySQL，support-files目录包含一些不同的my.cnf示例文件：my-huge.cnf、my-大.cnf、my-medium.cnf和my-small.cnf。可以使用这些文件来优化系统。

请注意如果在命令行中为**mysqld**或**mysqld\_safe**指定一个选项，它只在该次服务器调用中保持有效。要想每次服务器运行时使用该选项，将它放在选项文件中。

要想看参数更改的效果，应执行：

shell> **mysqld --key\_buffer\_size=32M --verbose ---help**

变量值列于输出的最后。确保--verbose和---help选项在最后。否则，在命令行中列于它们后面的选项的效果不会反映到输出中。

关于调节InnoDB存储引擎的信息，参见[15.2.11节，“InnoDB性能调节提示”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/storage-engines.html" \l "innodb-tuning" \o "15.2.11. InnoDB Performance Tuning Tips)。

### 7.5.3. 控制查询优化器的性能

查询优化器的任务是发现执行SQL查询的最佳方案。因为“好”方案和“坏”方案之间的性能差别会巨大(也就是说，秒相对于小时或甚至天)，大多数查询优化器，包括MySQL的查询优化器，总或多或少地在所有可能的查询评估方案中搜索最佳方案。对于联接查询，MySQL优化器所调查的可能的方案数随查询中所引用的表的数目呈指数增长。对于小数量的表(典型小于7-10)，这不是一个问题。然而，当提交的查询更大时，查询优化所花的时间会很容易地成为服务器性能的主要瓶颈。

查询优化的一个更加灵活的方法是允许用户控制优化器详尽地搜索最佳查询评估方案。一般思想是优化器调查的方案越少，它编译一个查询所花费的时间越少。另一方面，因为优化器跳过了一些方案，它可能错过一个最佳方案。

优化器关于方案数量评估的行为可以通过两个系统变量来控制：

·         optimizer\_prune\_level变量告诉优化器根据对每个表访问的行数的估计跳过某些方案。我们的试验显示该类“有根据的猜测”很少错过最佳方案，并且可以大大降低查询编辑次数。这就是为什么默认情况该选项为on(optimizer\_prune\_level=1)。然而，如果你认为优化器错过了一个更好的查询方案，则该选项可以关闭(optimizer\_prune\_level=0)，风险是查询编辑花费的时间更长。请注意即使使用该启发，优化器仍然可以探测呈指数数目的方案。

·         optimizer\_search\_depth变量告诉优化器对于每个未完成的“未来的”方案，应查看多深，以评估是否应对它进一步扩大。optimizer\_search\_depth值较小会使查询编辑次数大大减小。例如，如果optimizer\_search\_depth接近于查询中表的数量，对12、13或更多表的查询很可能需要几小时甚至几天的时间来编译。同时，如果用optimizer\_search\_depth等于3或4编辑，对于同一个查询，编译器编译时间可以少于1分钟。如果不能确定合理的optimizer\_search\_depth值，该变量可以设置为0，告诉优化器自动确定该值。

### 7.5.4. 编译和链接怎样影响MySQL的速度

下列大多数测试是在Linux上并用**MySQL**基准进行的，但是它们能对其它操作系统和工作负载给出一些指示。

当你用-static链接时，可以得到最快的可执行文件。

在Linux上，最好用**pgcc**和-O3编译服务器。为了用这些选项编译“sql\_yacc.cc”，需要大约200M内存，因为**gcc**或**pgcc**需要大量的内存使所有函数嵌入（inline）。在配置**MySQL**时，也应该设定CXX=gcc以避免包括libstdc++库(它不需要)。请注意对于某些版本的**pgcc**，生成的二进制只能运行在真Pentium处理器上，即使你使用编译器选项说明你想让最终的代码在所有x586-类处理器上工作(例如AMD)。

只通过使用一个较好的编译器或较好的编译器选项，在应用中能得到10-30%的加速。如果你自己编译SQL服务器，这特别重要！

当我们测试Cygnus CodeFusion或Fujitsu编译器时，二者均还没足够不出错来让MySQL启用优化进行编译。

标准MySQL二进制分发编译为支持所有字符集。当你自己编译MySQL时，应只包括将使用的字符集的支持。通过**configure**的--with-charset选项来控制。

这里是我们做过的一些测量表：

·         如果你使用**pgcc**并用-O6编译，**mysqld**服务器比用**gcc** 2.95.2快11%。

·         如果你动态地链接(没有-static)，在Linux中结果慢了13%。注意你仍能在客户应用程序中使用动态链接MySQL库。只有服务器对性能是关键的。

·         如果你用strip mysqld剥离**mysqld**二进制，生成的二进制可以快4%。

·         对于在同一主机上运行的客户与服务器之间的连接，如果你使用TCP/IP而非Unix套接字文件进行连接，结果慢7.5%。(在Unix中，如果你连接localhost主机，MySQL默认使用一个套接字文件）。

·         对于从客户到服务器的TCP/IP连接，从另一台主机连接一台远程服务器要比连接同一主机上的服务器慢8-11%，即使通过100Mb/s以太网进行连接。

·         当使用安全连接运行我们的基准测试时(所有数据用内部SSL支持进行加密)，性能比未加密连接慢55%。

·         如果你用--with-debug=full编译，大多数查询慢20%。部分查询时间会很长；例如，MySQL基准的运行要慢35%。如果你使用--with-debug(没有=full)，速度只下降15%。对于用--with-debug=full编译的**mysqld**版本，可以用--skip-safemalloc选项启动以便在运行时禁用内存检查。执行速度则接近用--with-debug配置的时候。

·         在Sun UltraSPARC-Iie上，用Forte 5.0编译的服务器比用**gcc** 3.2编译的要快4%。

·         在Sun UltraSPARC-Iie上，用Forte 5.0编译的32位模式服务器比64位模式服务器要快4%。

·         用**gcc** 2.95.2编译带-mcpu=v8 -Wa的UltraSPARC，使用-xarch=v8plusa选项性能会提高4%。

·         在Solaris 2.5.1上，在单个处理器上MIT-pthreads比带原生线程的Solaris慢8-12%。如果有更大的负载/cpus，差别应该更大。

·         在Linux-x86上使用**gcc**编译而不用帧指针(-fomit-frame-pointer or -fomit-frame-pointer -ffixed-ebp)，可以使**mysqld**快1-4%。

MySQL AB提供的Linux上的二进制MySQL分发一般用**pgcc**编译。我们必须返回到常规**gcc**，因为**pgcc**中有一个bug，使生成的二进制不能在AMD上运行。我们将继续使用**gcc**直到该bug被解决。同时，如果你有一个非AMD机，你可以用**pgcc**编译构建一个更快的二进制。标准MySQL Linux二进制是通过静态链接，以使它更快并且更加易于移植。

### 7.5.5. MySQL如何使用内存

下面的列表中列出了**mysqld**服务器使用内存的一些方法。在适用的地方，给出了内存相关的系统变量名：

·         键缓存(变量key\_buffer\_size)被所有线程共享；服务器使用的其它缓存则根据需要分配。参见[7.5.2节，“调节服务器参数”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "server-parameters" \o "7.5.2. Tuning Server Parameters)。

·         每个连接使用具体线程的空间：

o        堆栈(默认64KB，变量thread\_stack)

o        连接缓存区(变量net\_buffer\_length)

o        结果缓存区(变量net\_buffer\_length)

连接缓存区和结果缓存区可以根据需要动态扩充到max\_allowed\_packet。当某个查询运行时，也为当前查询字符串分配内存。

·         所有线程共享相同的基本内存。

·         只有压缩MyISAM表映射到内存。这是因为4GB的32位内存空间不足以容纳大多数大表。当64位地址空间的系统变得越来越普遍后，我们可以增加常规的内存映射支持。

·         对表进行顺序扫描的请求将分配一个缓存区(变量read\_buffer\_size)。

·         当按任意顺序读取行时(例如，按照排序顺序)，将分配一个随机读 缓存区(变量read\_rnd\_buffer\_size)以避免硬盘搜索。

·         所有联合在一个令牌内完成，并且大多数联合甚至可以不用临时表即可以完成。大多数临时表是基于内存的(HEAP)表。具有大的记录长度的临时表 (所有列的长度的和)或包含BLOB列的表存储在硬盘上。

如果某个内部heap（堆积）表大小超过tmp\_table\_size，MySQL可以根据需要自动将内存中的heap表改为基于硬盘的MyISAM表。还可以通过设置**mysqld**的tmp\_table\_size选项来增加临时表的大小，或设置客户程序中的SQL选项SQL\_BIG\_TABLE。参见[13.5.3节，“SET语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "set-option" \o "13.5.3. SET Syntax)。

·         进行排序的大多数请求将分配一个排序缓存区，并根据结果集的大小为两个临时文件分配零。参见[A.4.4节，“MySQL将临时文件储存在哪里”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/problems.html" \l "temporary-files" \o "A.4.4. Where MySQL Stores Temporary Files)。

·         几乎所有解析和计算在局部内存中完成。小项目不需要内存，因此避免了普通的慢内存分配和释放。只为不期望的大字符串分配内存；使用函数malloc()和free()来完成。

·         对于每个打开的MyISAM表，索引文件打开一次；数据文件为每个并行运行的线程打开一次。对于每个并行线程，将分配一个表结构、一个每个列的列结构和大小为3 \* *N*的缓存区(其中*N*是最大行的长度，而不是计算BLOB列)。一个BLOB列需要5至8个字节加上BLOB数据的长度。MyISAM 存储引擎维护一个额外的行缓存区供内部应用。

·         对于每个具有BLOB列的表，将对缓存区进行动态扩大以读入大的BLOB 值。如果你扫描一个表，则分配一个与最大的BLOB值一样大的缓存区。

·         所有使用的表的句柄结构保存在高速缓存中并以FIFO管理。默认情况，高速缓存有64个入口。如果某个表同时被两个运行的线程使用，高速缓存则为该提供两个入口。参见[7.4.9节，“MySQL如何打开和关闭表”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "table-cache" \o "7.4.9. How MySQL Opens and Closes Tables)。

·         当并行执行的线程结束时，FLUSH TABLE语句或**mysqladmin flush-table**命令可以立即关闭所有不使用的表并将所有使用中的表标记为已经关闭。这样可以有效释放大多数使用中的内存。FLUSH TABLE在关闭所有表之前不返回结果。

ps和其它系统状态程序可以报导mysqld使用很多内存。这可以是在不同的内存地址上的线程栈造成的。例如，Solaris版本的ps将栈间未用的内存算作已用的内存。你可以通过用swap -s检查可用交换区来验证它。我们用商业内存漏洞探查器测试了mysqld，因此应该有没有内存漏洞。

### 7.5.6. MySQL如何使用DNS

当新的客户连接**mysqld**时，**mysqld**创建一个新的线程来处理请求。该线程先检查是否主机名在主机名缓存中。如果不在，线程试图解析主机名：

·         如果操作系统支持线程安全gethostbyaddr\_r ()和gethostbyname\_r()调用，线程使用它们来执行主机名解析。

·         如果操作系统不支持线程安全调用，线程锁定一个互斥体并调用gethostbyaddr()和gethostbyname()。在这种情况下，在第1个线程解锁互斥体前，没有其它线程可以解析不在主机名缓存中的主机名。

你可以用--skip-name-resolve选项启动**mysqld**来禁用DNS主机名查找。然而，在这种情况下，你只可以使用MySQL中的授权表中的IP号。

如果你有一个很慢的DNS和许多主机，你可以通过用--skip-name-resolve禁用DNS查找或增加HOST\_CACHE\_SIZE定义(默认值：128)并重新编译**mysqld**来提高性能。

你可以用--skip-host-cache选项启动服务器来禁用主机名缓存。要想清除主机名缓存，执行FLUSH HOSTS语句或执行**mysqladmin flush-hosts**命令。

如果你想要完全禁止TCP/IP连接，用--skip-networking选项启动**mysqld**。

## 7.6. 磁盘事宜

[7.6.1. 使用符号链接](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "symbolic-links)

·         磁盘搜索是巨大的性能瓶颈。当数据量变得非常大以致于缓存性能变得不可能有效时，该问题变得更加明显。对于大数据库，其中你或多或少地随机访问数据，你可以确信对读取操作需要至少一次硬盘搜索，写操作需要多次硬盘搜索。要想使该问题最小化，应使用搜索次数较少的磁盘。

·         通过链接文件到不同的磁盘或对硬盘分段来增加可用磁盘锭数量（因此降低搜索成本）：

o        使用符号链接

这说明，对于MyISAM表，你符号链接索引文件和/或数据文件，从它们数据目录内的通常位置到另一个硬盘（也可以被条纹化）。这将使搜索和读次数达到最好，假定硬盘不再为其它目的使用。参见[7.6.1节，“使用符号链接”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "symbolic-links" \o "7.6.1. Using Symbolic Links)。

o        分条

分条意味着你有许多磁盘，将第1个块放到第1个硬盘，第2个块放到第2个磁盘，并且第*N*块在（*N* mod number\_of\_disks）磁盘上等等。这意味着如果正常数据大小小于分条大小（或完全匹配），能够得到最佳性能。分条完全取决于操作系统和分条大小，因此用不同的条纹大小对应用程序进行基准测试。参见[7.1.5节，“使用自己的基准”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "custom-benchmarks" \o "7.1.5. Using Your Own Benchmarks)。

分条的不同速度完全依赖于参数。依赖于怎样设置条纹参数和硬盘数量，可以根据不同数量级别得到不同的标准。你必须进行选择以便优化随机或顺序存取。

·         为了高可靠性你可能想使用RAID 0+1（条纹加镜像），但在这种情况下，需要2\*N块磁盘来保持N个磁盘的数据。如果你肯为它花钱，这可能是最好的选项。然而，你可能还必须投资一部分资金到卷管理软件中以便有效地管理它。

·         一个较好的选择是根据数据类型的重要性程度改变RAID级别。例如，保存可以在RAID 0硬盘上重新生成的不太重要的数据，但保存 真正重要数据（例如主机信息和日志）到RAID 0+1或RAID N硬盘。如果你有许多写操作，RAID N可能会存在问题，因为需要时间来更新校验位。

·         在Linux上，通过配置磁盘接口时使用hdparm，你可以获得更多的性能。（在一般负载下达到100%并不困难。）下面的hdparm选项 应该非常适用于MySQL，并且可能适用于许多其它应用程序：

     hdparm -m 16 -d 1

请注意，当使用该命令时，性能和可靠性依赖于硬件，因此我们强烈建议，使用hdparm后，完全测试你的系统。为获取更多详细信息，请查阅hdparm手册帮助页。如果hdparm使用的不好，会导致文件系统破坏，因此试验之前请做好每个备份!

·         也可以设置数据库使用的文件系统参数：

如果不需要知道文件最后一次访问的时间（这对数据库服务器并没有实际的用途），可以用-o noatime选项安装文件系统。这将跳过对文件系统中的节点的最后一次访问时间的更新，从而能够避免一些硬盘搜索。

在许多操作系统中，用-o async选项安装，可以将文件系统设置为异步更新。如果电脑相当稳定，这应该给予你更高的性能而不需要牺牲太多的可靠性。（该标志是Linux中默认开启。）

### 7.6.1. 使用符号链接

[7.6.1.1. 在Unix上使用针对数据库的符号链接](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "symbolic-links-to-databases)

[7.6.1.2. 在Unix平台上使用表的符号链接](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "symbolic-links-to-tables)[7.6.1.3. 在Windows平台上使用关于数据库的符号链接](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/optimization.html" \l "windows-symbolic-links)

你可以将表和数据库从数据库目录移动到其它的位置并且用指向新位置的符号链接进行替换。你可能需要这样作，例如，移动数据库到一个有更多空闲空间的文件系统中或通过将表分散到不同的磁盘上以增加系统的访问速度。

推荐的方法值需要将数据库通过符号链接指到不同的磁盘。符号链接表仅作为是最后的办法。

#### 7.6.1.1. 在Unix上使用针对数据库的符号链接

在Unix中，符号链接一个数据库的方法是，首先在一些有空闲空间的硬盘上创建一个目录，然后从MySQL数据目录中创建它的一个符号链接。

shell> **mkdir /dr1/databases/test**

shell> **ln -s /dr1/databases/test */path/to/datadir***

MySQL不支持链接一个目录到多个数据库。用一个符号链接替换一个数据库目录，只要你不在数据库之间制作符号链接。假定你有一个数据库db1在MySQL数据目录下，然后生成一个符号链接db2指向 db1：

shell> **cd */path/to/datadir***

shell> **ln -s db1 db2**

对于db1中的任何表tbl\_a，在db2也出现并且还是表tbl\_a。如果一个客户更新db1.tbl\_a并且另一个客户更新db2.tbl\_a，可能会出现问题。

然而，如果你真的需要这样做，可以通过改变源文件mysys/my\_symlink.c来实现，在该文件中你应查找下面的语句：

if (!(MyFlags & MY\_RESOLVE\_LINK) ||

    (!lstat(filename,&stat\_buff) && S\_ISLNK(stat\_buff.st\_mode)))

将该语句修改为：

if (1)

请注意，对于所有的Windows服务器，默认启用符号链接支持。

#### 7.6.1.2. 在Unix平台上使用表的符号链接

你不应在没有完全可操作的realpath()调用的系统中对表进行符号链接。（Linux和Solaris支持realpath())。可以通过发出一个SHOW VARIABLES LIKE 'have\_symlink'语句，检查系统是否支持符号链接。

只有MyISAM表完全支持符号链接。对于其它表类型，如果试图在操作系统中的文件上用前面的任何语句使用符号链接，可能会出现奇怪的问题。

对于MyISAM表的符号链接的处理如下：

·         在数据目录指，一定会有表定义文件、数据文件和索引文件。数据文件和索引文件可以移到别处和在数据目录中符号链接替代。表定义文件不能进行符号链接替换。

·         可以分别通过符号链接将数据文件和索引文件指到不同的目录。

·         如果**mysqld**没有运行，符号链接可以从服务器命令行使用ln -s手动完成。同样，通过使用DATA DIRECTORY和INDEX DIRECTORY选项创建表，你可以指示运行的MySQL服务器执行符号链接。参见[13.1.5节，“CREATE TABLE语法”](mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/sql-syntax.html" \l "create-table" \o "13.1.5. CREATE TABLE Syntax)。

·         **myisamchk**不用数据文件或索引文件替换符号链接。它直接工作在符号链接指向的文件。任何临时文件创建在数据文件或索引文件所处的目录中。

·         **注释：**当你删掉一个表时，如果该表使用了符号链接，符号链接和该符号链接指向的文件都被删除掉。这就是你不应以系统root用户运行**mysqld**或允许系统用户对MySQL数据库目录有写访问权限的原因。

·         如果你用ALTER TABLE ... RENAME重命名一个表并且不将表移到另一个数据库，数据库目录中的符号链接被重新命名为一个新名字并且数据文件和索引文件也相应地重新命名。

·         如果你用ALTER TABLE ... RENAME移动一个表到另一个数据库，表移动到另一个数据库目录。旧的符号链接和其所指向的文件被删除。换句话说，新表不再被链接。

·         如果不使用符号链接，你应对**mysqld**使用--skip-symbolic-links选项以确保没有人能够使用**mysqld**来删除或重新命名数据目录之外的文件。

表符号链接还不支持以下操作：

·         ALTER TABLE忽略DATA DIRECTORY 和INDEX DIRECTORY表选项。

·         BACKUP TABLE 和RESTORE TABLE不考虑符号链接。

·         .frm文件必须绝不能是一个符号链接（如前面所述，只有数据和索引文件可以是符号链接）。如果试图这样做（例如，生成符号链接）会产生不正确的结果。假定你在MySQL数据目录下有一个数据库db1，该数据库有一个表tbl1，并且在db1目录中你制作了一个符号链接tbl2指向tbl1：

·                shell> **cd */path/to/datadir*/db1**

·                shell> **ln -s tbl1.frm tbl2.frm**

·                shell> **ln -s tbl1.MYD tbl2.MYD**

·                shell> **ln -s tbl1.MYI tbl2.MYI**

如果一个线程读取db1.tbl1同时另一个线程更新 db1.tbl2会发生问题：

o        查询缓存将变为“傻瓜” (它没有办法知道tbl1是否被更新，因此它返回过时的结果)。

o        tbl2上的ALTER语句也会失败。

#### 7.6.1.3. 在Windows平台上使用关于数据库的符号链接

Windows的**mysqld-max**和mysql-max-nt服务器使用-DUSE\_SYMDIR选项编译成。允许你放置数据库目录到一个不同的硬盘，通过设置一个符号链接指向它。这类似于Unix中的符号链接，尽管设置链接的过程不同。

符号链接默认为启用。如果你不需要，使用skip-symbolic-links选项来禁用它：

[mysqld]

skip-symbolic-links

在Windows中，通过在数据目录中创建一个文件，该文件包含目标目录的路径，你可以为MySQL数据库创建一个符号链接。该文件应该被命名为db\_name.sym，其中*db\_name*是数据库名。

假定MySQ数据目录是C:\mysql\data并且你想要数据库foo放置在D:\data\foo。设置一个符号链接如下所示：

1.    确保D:\data\foo目录存在，如果必要创建它。如果你在数据目录内有一个命名为foo的数据库目录，你应移动它到D:\data目录。否则，符号链接无效。为避免出现问题，当你移动数据库目录时服务器不应该运行。

2.    创建一个文本文件C:\mysql\data\foo.sym，该本文文件包含路径名D:\data\foo\。

此后，数据库foo创建的所有表将创建在D:\data\foo文件中。请注意，如果在MySQL数据目录中存在同名的数据库目录，不能使用符号链接。

这是MySQL参考手册的翻译版本，关于MySQL参考手册，请访问[dev.mysql.com](http://dev.mysql.com/doc/mysql/en" \t "mk:@MSITStore:D:\\IT\\Tool_book\\Tool参考书\\mysql_orcal\\mysql5.1参考手册.chm::/_top)。 原始参考手册为英文版，与英文版参考手册相比，本翻译版可能不是最新的。