# 第1章：数据库基础

## 1.1 数据库系统概述

### 1.1.1 数据库技术的发展

### 1.1.2 数据库系统的组成

## 1.2 数据模型

### 1.2.1 数据模型的概念

### 1.2.2 常见的数据模型

### 1.2.3 关系数据库的规划化

### 1.2.4 关系数据库的设计原则

### 1.2.5 实体与关系

## 1.3数据库的实体结构

### 1.3.1 数据库三级模式结构

### 1.3.2 三级模式之间的映射

## 1.4 总结

# 第2章：初识MySQL

## 2.1了解MySQL

### 2.1.1 MySQL数据库的概念

### 2.1.2 MySQL的优势

### 2.1.3 MySQL的发展史

## 2.2 MySQL的特性

## 2.3 MySQL的应用环境

## 2.4 MySQL服务器的安装和配置

### 2.4.1 MySQL服务器下载

### 2.4.2 MySQL服务器安装

### 2.4.3 启动、连接、断开和停止MySQL服务器

(1) 启动和停止MySQL服务器

net start mysql 🡪启动

net stop mysql 🡪停止

(2) 连接和断开MySQ服务器

mysql –uroot –p password（密码） 🡪连接

quit 🡪断开

### 2.4.4 打开MySQL5.6 Command Line Client

## 2.5 如何学好MySQL？

## 2.6 总结

# 第3章：使用MySQL图形界面化工具

# 第4章：数据库操作

## 4.1认识数据库

### 4.1.1数据库基本概念

数据库？

数据库系统？==DBS

数据库管理系统？==DBMS

关系数据库？

DBA? DDL? DQL?

### 4.1.2数据库常用对象

表，字段，索引，视图，存储过程

（1）表：表是数据库中所有数据的数据库对象，由行和列组成，用于组织和存储数据。

（2）字段：表中的每列称为一个字段，字段具有自己的属性，如字段类型，字段大小等。

其中字段类型是字段最重要的属性，它决定子字段能够存储哪种数据。

SQL规定5种基本字段类型：字符型；文本型；数值型；逻辑型；日期时间型。

（3）索引：索引是一个单独的、物理的数据结构。它是依赖于表建立的，在数据库中索引使数据库程序无须对整个表进行扫描，就可以在其中找到所需要的数据。

（4）视图：视图是从一张或多张表中导出的表（也称虚拟表），是用户查看数据表中数据的一种方式。表中包括几个被定义的数据列与数据行，其结构和数据建立在对表的查询基础之上。

（5）存储过程：存储过程是一组为了完成特定功能的SQL语句集合（包括查询、插入、删除和更新等操作），经编译后以名称的形式存储在SQL server服务器端的数据库中，由用户通过指定存储过程的名字来执行。当这个存储过程被调用执行时，这些操作也会同步执行。

### 4.1.3系统数据库

## 4.2 创建数据库

使用CREATE DATABASE或者CREATE SCHEMA创建数据库。语法格式如下：

CREATE {DATABASE/SCHEMA} [IF NOT EXISTS] 数据库名[

[DEFAULT] CHARACTER SET [=] 字符集 |

[DEFAULT] COLLATE [=] 校对规则名称

];

注意：在语法中，花括号“{}”表示必选项；中括号“[]”表示可选项；竖线“|”表示分隔符两边的内容为“或”关系

参数说明：

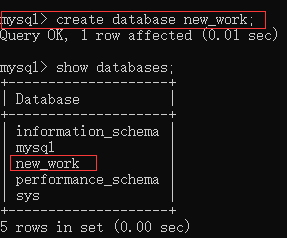
IF NOT EXISTS表示创建数据库前进行判断，只有该数据库不存在的时候才会执行创建语句；

数据库名必须指定，在文件系统中，MySQL的数据存储区将以目录方式表示MySQL数据库。因此这里的数据库名必须符合操作系统文件夹命名规则。

在Windows下数据库名、表名的大小写不敏感，而在Linux系统下大小写敏感。

1. 常见创建方式：

create database new\_work;



1. 创建指定字符集的数据库；

create database db\_admin character set = GBK;

1. 创建前检查是否存在同名数据库；

create database if not exists db\_admin;

## 4.3 查看数据库

使用SHOW关键字来查看数据库。语法如下：

SHOW {DATABASE|SCHEMA} [LIKE ‘模式’ WHERE 条件];

1、常用：

show databases;

2、筛选：

show databases like ‘db\_%’;

## 4.4 选择数据库

使用USE关键词选择数据库。语法如下：

USE 数据库名；

## 4.5 修改数据库

修改数据库的字符集和校对规则名称等使用ALTER关键词。语法如下：

ALTER {DATABASE|SCHEMA} [数据库名]

[DEFAULT] CHARACTER SET [=] 字符集 |

[DEFAULT] COLLATER [=] 校对规则名称；

注意：这里的修改是指修改被创建数据库的一些相关参数，并不能修改数据库的名。

示例：

alter database db\_admin

default character set = gbk

default collate gbk\_chinese\_ci;

## 4.6 删除数据库

使用DROP关键字删除数据库。语法如下：

DROP {DATABASE|SCHEMA} [IF EXISTS] 数据库名；

## 4.7 总结

(1) 数据库的创建、选择、查看、修改、删除:

1.创建：create database 数据库名;

2.选择：use 数据库名；

3.查看：show databases;

4.修改：alter database character =字符集;

5.删除：drop database 数据库名；

(2) 数据库的常用对象：

1.表：数据库中有很多表

2.字段：表中的每一列都是一个字段

3.索引：

4.视图

5.存储过程

# 第5章：存储引擎及数据类型

## 5.1 MySQL存储引擎

### 5.1.1 存储引擎的概念

使用存储引擎可以加快查询的速度，每一种引擎都存在不同的含义。

存储引擎其实就是如何存储数据，如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。

### 5.1.2 查询支持的存储引擎

SHOW ENGINES;

SHOW ENGINES \g

SHOW ENGINES \G

### 5.1.3 InnoDB存储引擎

为MySQL提供了事务、回滚、奔溃修复能力和多版本并发控制的事务安全；

是MySQL第一个提供外键约束的表引擎；

优势在于对事务的处理能力；

支持自动增长列AUTO INCREMENT，值不能为空，且值必须唯一；

MySQL规定自增列必须为主键；

在插入值时，自动增长列：不输入，0或者null，则自动为自增长后的值，如果输入某个确定的值并且这个值在前面没有出现过，则可以直接插入使用；

支持外键FOREIGN KEY，外键所在的表为子表，外键所依赖的表为父表，父表中被子表外键关联的字段必须为主键，当更新、删除父表中某条信息时，子表也必须有相应的改变；

总之：InnoDB存储引擎的优势在于提供了良好的事务管理，奔溃修复能力和并发控制。缺点是其读写效率稍差，占用的数据空间相对比较大。

InnoDB是下列情况的理想型：

1.更新密集的表：适合处理多重并发的更新请求；

2.事务：唯一支持事务的MySQL引擎，是管理敏感信息（金融，注册信息）的必须软件；

3.自动灾难恢复：恢复时间较短；

提供了ACID独立性，一致性，隔离性，持久性兼容事务处理能力。

### 5.1.4 MyISAM存储引擎

曾是MySQL的默认存储引擎，是基于ISAM引擎发展起来的，解决了许多ISAM的不足。

总之：MyISAM存储优势在于占用空间小，处理速度快；缺点是不支持事务的完整性和并发性；

### 5.1.5 MEMORY存储引擎

一种特殊的存储引擎，使用存储在内存中的内容来创建表，并且所有的数据也是放在内存中的。

## 5.2 MySQL数据类型

在数据库中，每一条数据都有其数据类型。MySQL支持的数据类型主要分为3类：数字类型，字符串（字符）类型，日期和时间类型；

### 5.2.1 数字类型

TINYINT

INT

FLOAT

### 5.2.2 字符串类型

char(M)

varchar

TEXT

BLOB

Enum

Set

创建表时，使用字符串类型应遵循以下原则：

1.从速度方面，要选择固定的列，使用CHAR类型

2.要节省空间，使用动态的列，使用VARCHAR类型

3.要将列中的内容限制在一种选择，使用ENUM类型

4.允许在一个列中有多于一个的条目，使用SET类型

5.如果搜素的内容不区分大小写，使用TEXT类型

6.如果要搜索的内容区分大小写，使用BLOB类型

### 5.2.3 日期和时间类型

DATE

TIME

DATETIME

TIMESTAMP

YEAR

## 5.3 总结

1.存储引擎

InnoDB、MyISAM、MEMORY

MySQL的默认存储引擎是InnoDB，其优势在于具有强大的事务处理能力？并发控制能力？

2.MySQL的数据类型

数字类型：int/float

字符类型：char/varchar/

时期类型：date/time/datetime

# 第6章：操作数据表

## 6.1 创建数据表

使用CREATE关键词创建数据表

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] 数据表名

[(create\_define, …)][table\_options][select\_statement]

参数说明：

TEMPORARY表示创建一个临时表

IF NOT EXISTS起到检查作用，避免创建表存在的情况下出现的错误

create\_define表示列属性部分，且至少包含一列

table\_options表示数据表的一些特性参数，例如表数据如何存储和存储在何处，定义表的存储引擎

select\_statement表示select语句描述部分，可以快速创建表

重点介绍create\_define部分：

col\_name type[NOT NULL|NULL] [DEFAULT default\_value] [AUTO\_INCREMENT]

[PRIMARY KEY] [reference\_definition]

col\_name字段名

type字段类型

NOT NULL|NULL

DEFAULT default\_value

AUTO\_INCREMENT

PRIMARY KEY

reference\_definition

示例：

create table if not exists students(

id auto\_increment,

student\_name varchar(20) not null,

student\_age int not null,

student\_city varchar(20) not null

)

create table tb\_admin(

id int auto\_increment primary key,

user varchar(30) not null,

password varchar(30) not null,

createtime datetime

);

## 6.2 查看表结构

### 6.2.1 使用SHOW COLUMNS语句查看

SHOW [FULL] COLUMNS FROM 数据表名 [FROM 数据库名];

SHOW [FULL] COLUMNS 数据库名.数据表名;

### 6.2.2 使用DESCRIBE语句查看

DESCRIBE表名;

DESCRIBE表名 列名;

注意：其中DESCRIBE 可以简写成DESC;

因此通常使用desc 表名；来查看表的结构

## 6.3 修改表结构

修改表的结构使用ALTER TABLE语句。

修改表结构指的是增加或者删除字段、修改字段名或者字段类型、设置取消主键外键、设置取消索引以及修改表的注释，语法如下：

ALTER [IGNORE] TABLE 数据表名 alter\_spec[, alter\_apec]… | table\_options

参数说明：

IGNORE：可选项，表示如果出现重复关键的行，则只执行一行，其他重复的行被删除；

alter\_spec子句：用于定义要修改的内容，其语法格式如下:

ADD [COLUMN] create\_definition [FIRST |AFTER column\_name] //添加新字段

ADD INDEX [index\_name] (index\_col\_name,…) //添加索引名称

ADD PRIMARY KEY (index\_col\_name,…) //添加主键名称

ADD UNIQUE [index\_name] (index\_col\_name,…) //添加唯一索引

ALTER [COLUMN] col\_name {SET DEFAULT literal |DROP DEFAULT} //修改字段默认值

CHANGE [COLUMN] old\_col\_name create\_definition //修改字段名/类型

MODIFY [COLUMN] create\_definition //修改子句定义字段

DROP [COLUMN] col\_name //删除字段名称

DROP PRIMARY KEY //删除主键名称

DROP INDEX index\_name //删除索引名称

RENAME [AS] new\_tbl\_name //更改表名

table\_options:用于指定表的一些特性参数，其中大部分选项涉及的是表数据如何存储及存储在何处，如ENGINE选项用于定义表的存储引擎。

### 6.3.1 添加新字段及修改字段定义

在MySQL的ALTER TABLE语句中，我们通过使用ADD [COLUMN] create\_definition [FRIST | AFTER column\_name]来添加新字段；使用MODIFY [COLUMN] create\_definition可以修改已经定义的字段；

示例：添加一个新的字段email，类型为varchar(50)，not null，将字段user的类型由varchar(30)改为varchar(40)。

USE db\_admin;

ALTER TABLE tb\_admin ADD email varchar(50) not null,

modify user varchar(40);

注意：通过ALTER语句修改表列，其前提是必须将表中数据全部删除，然后才可以修改表列。

### 6.3.2 修改字段名

在MySQL的ALTER TABLE语句中，使用CHANGE [COLUMN] old\_col\_name create\_definition子句可以修改字段名和字段类型。

示例：将tb\_userNew1的字段名user修改称username：

ALTER TABLE db\_admin.tab\_usernew1

CHANGE COLUMN user username VARCHAR(30) NULL DEFAULT NULL;

### 6.3.3 删除字段

在MySQL的ALTER TABLE语句中，使用DROP [COLUMN] col\_name子句可以删除指定字段。

示例：将数据库db\_admin中的数据表tb\_userNew1更名为tb\_userOld

USE db\_admin;

ALTER TABLE tb\_admin DROP email;

### 6.3.4 修改表名

在MySQL的ALTER TABLE语句中，使用RENAME [AS] new\_tbl\_name子句可以修改表名。

示例：将数据表tb\_admin中的字段email删除

USE db\_admin;

ALTER TABLE tb\_usernew1 RENAME AS tb\_userOld;

## 6.4 重命名表

在MySQL中，重命名数据表可以使用RENAME TABLE语句来实现。语法格式如下：

RENAME TABLE 数据表名1 To 数据表名2;

注意：该语句可以同时对多个数据表进行重命名，多个表之间以逗号“,”分隔开。

示例：对数据表tb\_admin进行重命名后，更名后的数据表为tb\_user

RENAME TABLE tb\_admin TO tb\_user;

## 6.5 复制表

创建表的CREATE TABLE命令还有另外一种语法结构，在一张已经存在的数据表的基础上创建一份该表的备份，也就是复制表。语法格式如下：

CREATE TABLE [IF NOT EXISTS] 数据表名

LIKE 源数据表名;

示例：创建一个数据表tb\_user的备份tb\_userNew

CREATE TABLE tb\_userNew

LIKE tb\_user;

注意：该方式复制表的时候并没有复制表的数据，如果想要同时复制表中的内容，需要使用如下代码实现：

CREATE TABLE tb\_userNew

AS SELECT \* FROM tb\_user;

## 6.6 删除表

使用DROP TABLE 语句实现删除数据表。语法格式如下：

DROP TABLE [IF EXISTS] 数据表名;

注意：可以同时删除多个数据表，各个数据表之间使用逗号隔开。

示例：删除数据表名tb\_user

USE db\_admin;

DROP TABLE;

注意：删除数据表要谨慎，一旦删除，数据表的全部内容都会被清空。

## 6.7 总结

如何创建数据表，查看表的结构，修改表结构，重命名表，复制表和删除表的操作。各语句总结如下：

1.创建一个数据表；

create table 数据表名 (

col\_name type(length) [NOT NULL|NULL] [AUTO\_INCREMENT] [PRIMARY KEY]

)

如：创建一个学生表

create table tb\_users(

id int not null auto\_increment primary key,

name varchar(20) not null,

age int not null,

city varchar(20) not null

);

2.查看表的结构

desc table\_name;

3.修改表的结构

# 第7章：MySQL基础

## 7.1 运算符

### 7.1.1 算术运算符

+ 加

- 减

\* 乘

/ 除

% 求余

DIV 相当于 /

MOD 相当于 %

示例：

select row, row+row, row-row, row\*row, row/row from tb\_admin;

### 7.1.2 比较运算符

=

>

<

>=

<=

!=或<>

IS NULL

IS NOT NULL

BETWEEN AND

IN

NOT IN

LIKE

NOT LIKE

REGEXP

详解：

“=”用来判断数字，字符串和表达式等是否相等。如果相等，就返回1，否则返回0。

注意：在判断两个字符是否相同时，数据库系统是根据字符的ASCII码进行判断的。如果ASCII相等，则表示这两个字符相同，否则就不相同。切记，空值NULL不能使用“=”来判断。

“<>”和“!=”用来判断数字，字符串，表达式是否不相等。如果不相等，则返回1，否则返回0。切记不能判断空值NULL。

“>”用来判断左边的操作数是否大于右边的操作数。如果大于，返回1，否则返回0。同样不能判断空值NULL。

“<”和“<=”和“>=”和“>”使用方式一致。

IS NULL用来判断操作数是否为空值NULL。操作数为NULL值时，返回1，否则返回0。

IS NOT NULL刚好与IS NULL 相反。

BETWEEN AND用来判断数据是否在某个取值范围内，其表达式如下：[M, N]

x1 BETWEEN m AND n（如果x1大于等于m，且小于等于n，结果将返回1，否则将返回0）

在范围内返回1，否则返回0，空值返回NULL。

IN 用来判断数据是否存在于某个集合之中，其表达式如下所示：

x1 in (值1，值2，值3，。。。，值n)

如果x1等于值1到值n中的任何一个值，结果将返回1，否则返回0。

LIKE用来匹配字符串，其表达式如下：

x1 LIKE s1

如果x1与字符串s1匹配，结果返回1，否则返回0。

REGEXP匹配字符串，但其使用的是正则表达式进行匹配，其表达式如下：

x1 REGEXP ‘匹配模式’

如果x1满足匹配方式，结果将返回1，否则返回0。

说明：= <> != > >= < <=等运算符不能用来判断空值NULL。一旦使用，结果将返回NULL。如果要判断一个值是否为空值，则可以使用<=>，IS NULL和IS NOT NULL。

REGEXP运算符经常与“^”,“$”，“.”一起使用。其中“^”匹配字符串的开始部分，‘$’匹配的是字符串的结尾部分，‘.’代表的字符串中的一个字符。

### 7.1.3 逻辑运算符

判断真假。如果表达式为真则返回1，如果表达式为假，则返回0。支持以下表达：

&& 或 AND

如果所有数据不为0且不为空值（NULL）时，返回1；如果存在任何一个数据为0时，返回0；如果存在一个数据为NULL且没有数据为0时，返回NULL。

|| 或 OR

如果所有数据中存在一个数据为非0的数字时，返回1；如果数据中不包含非0的数字，但包含NULL时，返回NULL；如果操作数中有0时，返回0。

! 或 NOT

如果操作数据是非0的数字，结果返回0；如果操作数据是0，返回1,；如果操作数据是NULL，结果返回NULL。

XOR 异或

只要其中任何一个操作数据为NULL时，结果返回NULL；如果两个操作数都是非0值，或者都是0，则返回结果为0；如果一个为0，另一个为非0值，返回结果1。

### 7.1.4 位运算符

位运算符会将操作数变成二进制数再进行位运算符，然后再将计算结果从二进制变成十进制数。支持6种位运算符，分别为按位与，按位或，按位取反，按位异或，按位左移，按位右移。如下所示：

& 按位与

| 按位或

~ 按位取反

^ 按位异或

<< 按位左移

>> 按位右移

### 7.1.5 运算符的优先级

以下运算符的优先由高到低，同级别从左到右

!

~

^

\*，DIV，%，MOD

+，-

>>，<<

&

|

= ，<=> ，< ，<= ，> ，>= ，!= ，<> ，IN ，IS ，NULL ，LIKE ，REGEXP

BETWEEN AND，CASE ，WHEN ，THEN ，ELSE

NOT

&&， AND

|| ，OR， XOR

:=

## 7.2 流程控制语句

### 7.2.1 IF语句

IF语句用来进行条件判断，根据不同的条件执行不同的操作。该语句在执行时会首先判断IF后的条件是否为真，则执行THEN后的语句，如果为假则继续判断IF语句直到真为止，当以上条件都不满足的时候，则执行ELSE语句后的内容。IF语句形式如下：

IF condition THEN

…

[ELSE condition THEN]

…

[ELSE]

…

ENDIF

示例：这里有问题

delimiter //

create procedure example\_if (in x int)

begin

if x=1 then select 1;

elseif x=2 then select 2;

else select 3;

end if;

end

//

调用该存储过程：

call example\_if(2)//

### 7.2.2 CASE语句

CASE语句为多分支语句结构，该语句首先从WHEN后的VALUE中查找与CASE后的VALUE相等的值，如果查找到就执行该分支的内容，否则执行ELSE后的内容。表现形式如下：

CASE value

WHEN value THEN …

[WHEN value THEN…]

[ELSE…]

END CASE

其中，value参数表示条件判断的变量；WHEN..THEN中的value参数表示变量的取值。CASE语句的另一种语法表达形式如下：

CASE

WHEN value THEN …

[WHEN value THEN …]

[ELSE …]

END CASE

示例：

delimiter //

create procedure example\_case (in x int)

begin

case x

when 1 then select 1;

when 2 then select 2;

else select 3;

end case;

end

//

调用该存储过程：

call example\_case(3)

### 7.2.3 WHILE语句

WHILE循环语句执行时首先判断condition条件是否为真，如果是则执行循环体，否则退出循环。该语句表示形式如下：

WHILE condition DO

…

END WHILE;

示例：求前100项的和

delimiter //

create procedure example\_while (out sum int)

begin

declare i int default 1;

declare s int default 0;

while i<=100 do

set s=s+i;

set i=i+1;

end while;

set sum=s;

end

//

调用该存储过程：

call example\_while(@s)

mysql>select @s

### 7.2.4 LOOP循环语句

该循环没有内置的循环条件，但是可以使用LEAVE语句退出循环。LOOP语句允许某特定语句或语句群的重复执行，实现一个简单的循环构造，中间省略的部分是需要重复执行的语句。在循环内的语句一直重复直至循环被退出，退出循环应用LEAVE语句。

表示形式如下：

LOOP

...

END LOOP

其中LEAVE语句经常和BEGIN...END或循环一起使用，其表示形式如下：

LEAVE label

其中label是语句中标注的名字，这个名字是自定义的，加上LEAVE关键字就可以用来退出被标注的循环语句。

示例：求前100项的和

delimiter //

create procedure example\_loop (out sum int)

begin

declare i int default 1;

declare s int default 0;

loop\_label:loop

set s=s+i;

set i=i+1;

if i>100 then

leave loop\_label;

end if;

end loop;

set sum=s;

end

//

调用存储过程：

call example\_loop(@s)

select @s

### 7.2.5 REPEAT循环语句

该语句先执行一次循环体，之后判断condition条件是否为真，为真则退出循环，否则继续执行循环体。表示形式如下：

REAPEAT

...

UNTIL condition

END REPEAT

示例：求前100项的和

delimiter //

create procedure\_repeat (out sum int)

begin

declare i int default 1;

declare s int default 0;

repeat

set s=s+i;

set i=i+1;

until I >100

end repeat;

set sum=s;

end

//

调用存储过程：

call example\_repeat(@s)

select @s

循环语句中还有一个ITEAATE语句，它可以出现在LOOP、REPEAT和WHILE语句内，其意为“再次循环”。格式如下：

ITERATE label

该语句的格式与LEATE大同小异，区别在于：LEAVE语句是离开一个循环，而ITERAE语句是重新开始一个循环。

## 7.3 总结

# 第8章：数据表的增删改

## 8.1 插入数据

建立一个空的数据库和数据表之后，可以使用INSERT语句向数据表中插入新行，也就是插入一行新记录。INSERT语句中共有3种语法形式，分别是INSERT ... VALUES语句、INSERT ...SET语句和INSERT...SELECT语句。

### 8.1.1 使用INSERT ... VALUES语句插入数据

语法形式如下：

INSERT [LOW\_PRIORITY | DELAYED |HIGH\_PRIORITY] [IGNORE]

[INTO] 数据表名 [(字段名, ...)]

values ({值| DEFAULT}, ...), (...), ...

[ON DUPLICATE KEY UPDATE 字段名=表达式, ...]

1.插入完整数据

insert into tb\_admin values (1, ’mr’, ‘mrsoft’, ‘2014-09-05 10:25:20’);

2.插入数据记录的一部分：

也就是只插入表的一行中某几个字段的值。这里示例只向数据表tb\_admin中插入user和password字段的值：

insert into tb\_admin (user, password) values (‘gkdka’, ‘111’);

3.插入多条记录：

insert into tb\_admin (user, password, createtime)

values (‘mrbccd’, ‘111’, ‘2019-09-05 10:35:26’),

(‘mingri’, ‘111’, ‘2019-09-05 10:45:27’),

(‘mingrisoft’, ‘111’, ‘2019-09-05 10:55:28’);

### 8.1.2 使用INSERT ... SET 语句插入数据

INSERT [LOW\_PRIORITY | DELAYED |HIGH\_PRIORITY] [IGNORE]

[INTO] 数据表名

SET 字段名 = {值 | DEFAULT}, ...

[ON DUPLICATE KEY UPDATE 字段名 = 表达式, ...]

示例：

insert into tb\_admin set user =‘mrdj’, password= ‘111’, createtime=’2021-08-01’;

### 8.1.3 插入查询结果

INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]

[INTO] 数据表名 [(字段名, ...)]

SELECT ...

[ON DUPLICATE KEY UPDATE 字段名=表达式, ...]

示例：从数据表tb\_mrbook中查询出user和pass字段的值，插入到数据表tb\_admin中

insert into tb\_admin (user, password) select user, pass from tb\_mrbook;

## 8.2 修改数据

修改数据表使用UPDATE语句，形式如下：

UPDATE [LOW\_PRIORITY] [IGNORE] 数据表名

SET字段1=值1 [, 字段2=值2, ...]

[WHERE 条件表达式]

[ORDER BY ...]

[LIMIT 行数];

示例：修改用户名为mrbccd的管理员密码为123

update tb\_admin set password=’123’ where user=’mrbccd’;

## 8.3 删除数据

使用DELETE语句或者TRUNCATE TABLE语句删除表中的一行或多行数据

### 8.3.1 通过DELETE语句删除数据

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM 数据表名

[WHERE 条件表达式]

[ORDER BY ...]

[LIMIT 行数];

示例：删除数据表tb\_admin中用户名为mr的记录信息

delete from tb\_admin where user=’mr’;

注意：

（1）如果没有指定WHERE筛选条件，将删除所有的记录；

（2）实际应用中，执行删除操作时，执行删除的条件一般应该为数据的id，而不是具体某个字段的值，因为id是唯一的；

### 8.3.2 通过TRUNCATE TABLE语句删除数据

形式如下：

TRUNCATE [TABLE] [数据库名.]数据表名

示例：清空管理员数据表tb\_admin;

truncate table tb\_admin;

注意：该语句会删除数据表的中所有数据，且无法恢复。

### 8.3.3 DELETE语句和TRUNCATE TABLE语句中的区别如下：

（1）使用TRUNCATE TABLE语句后，表中的AUTO\_INCREMENT计数器将重新设置为该列的初始值；

（2）对那些参与索引和视图的表，不能使用TRUNCATE TABLE语句来删除数据，而应该使用DELETE语句;

（3）TRUNCATE TABLE操作与DELETE操作使用的系统和事务日志资源少。DELETE语句每删除一行都会在事务日志中添加一行记录，而TRUNCATE TABLE语句是通过释放存储表数据所用的数据页来删除数据的，因此只在事务日志中记录页的释放；

### 8.3.4 总结

# 第9章：数据查询

## 9.1 基本查询语句

SELECT语句的基本语法如下：

select selection\_list //要查询的内容，选择哪些列（字段）

from 数据表名 //指定数据表

where primary\_constraint //查询时满足的条件，行必须满足的条件

group by grouping\_columns //如何对结果进行分组

order by sorting\_cloumns //如何对结果进行排序

having secondary\_constraint //查询时满足的第二条件

limit count; //限定输出的查询结果数量

1. 查询一个数据表

select \* from tb\_students;

2. 查询数据表中的一个或多个列（字段）

select name, age from tb\_students;

3. 从一个或多个表中查询数据

select tb\_admin.id, tb\_admin.tb\_user, tb\_students.id, tb\_students.name from tb\_admin, tb\_students;

4.使用where连接多个数据表

select salary from user, jtsr where user.user=jtsr.user and user.id=1;

注意：

在查询数据表中的数据时，如果数据中涉及到中文字符串，有可能在输出时会出现乱码，那么在最后执行查询操作之前，通过set names语句设置前编码格式，然后再输出中文字符时就不会出现乱码了。设置编码格式为gb2312

## 9.2 单表查询

指的是从一个表中查询所需要的数据。

### 9.2.1 查询所有字段

select \* from 数据表名; //星号“\*”代表所有的列（字段）

### 9.2.2 查询指定字段

select 字段1, 字段2, ... from 数据表名; //查询多个字段使用逗号“，”隔开

### 9.2.3 查询指定数据

如果要从很多记录中查询出指定的记录，那么就需要一个查询的条件：使用WEHRE子句

适用于WHERE子句的运算符：

= < > >= <= !=(或<>) is null is not null between and in not in like not like regexp

示例：

select \* from tb\_login where user=’mr’;

(1)带关键字in的查询

关键字in可以判断某个字段的值是否在指定的集合之中。如果字段的值在集合中就会被查询出来，否则就不满足查询条件：

select \* from 数据表名 where 条件 [not] in (元素1，元素2，元素3，...，元素n);

示例：

select \* from tb\_login where user in (‘mr’, ‘lx’);

select \* from tb\_login where user not in (‘mr’, ‘lx’);

(2)带关键字between and的查询

关键字between and可以判断某个字段的值是否在指定的范围内。如果字段的值在指定范围之内，则记录会被查询出来，否则不满足查询条件。

示例：

select \* from tb\_login where id between 5 and 7;

select \* from tb\_login where id not between 5 and 7;

(3)带like的字符匹配查询

like是常用的比较运算符，能实现模糊查询。它有两种通配符：%和\_

其中“%”可以匹配一个或多个字符，可以代表任意长度的字符串，长度可以为0。而“\_”只匹配一个字符。

示例：

select \* from tb\_login where user like ‘%mr%’;

(4)用is null查询空值

关键字is null可以用来判断字段的值是否为空值（null）。如果字段为空值，则满足查询条件，则该记录会被查询出来，否则不满足查询条件。

示例：

select books, row from tb\_book where row is null;

select books, row from tb\_book where row is not null;

(5)带关键字and的多条件查询

关键字and可以用来联合多个条件进行查询。使用关键字and时，只要同时满足所有查询条件，则记录会被查询出来。如不满足这些查询条件中的其中一个，则记录会被排除掉。

select \* from 数据表名 where 条件1 and 条件2 [... and 条件n]；

示例：

select \* from tb\_login where user=’mr’ and section=’php’;

(5)带关键字or的多条件查询

关键字or也可以联合多个条件进行查询，但是与关键and不同是关键字or只要满足查询条件中的一个，那么此记录就会被记录下来；如果不满足其中的任何一个，记录就会被排除掉。

select \* from 数据表名 where 条件1 or 条件2 [... or 条件n];

示例：

select\* from tb\_login where section=’php’ or section=’程序开发’;

(6)用关键字distinct去除结果中的重复行

使用关键字distinct可以去除查询结果中的重复记录，语法如下：

select distinct 字段名 from 数据表名;

示例：

select distinct name from tb\_login;

(7)用关键字order by 对查询结果排序

使用关键字order by 可以对查询的结果进行升序（asc）和降序（desc）排列，在默认的情况下，order by按照升序输出结果。语法格式如下：

order by 字段名 [asc|desc];

示例：

select \* from tb\_login order by id desc;

(8) 用关键字group by分组查询

通过关键字group by 可以将数据划分到不同的组中，实现对记录进行分组查询。在查询时，所查询的列必须包含在分组的列中，目的是使查询的数据没有矛盾。

1)使用关键字group by来分组

单独使用关键字group by查询结果只显示每组中的一条记录

示例：

selec id, books, talk from tb\_book group by talk;

2)关键字group by 与group\_concat()函数一起使用

一起使用可以将每个组中的所有字段值都显示出来

示例：

select id, books, group\_concat(talk) from tb\_book group by talk;

3)按多个字段进行分组

示例：

select id, books, talk, user from tb\_book group by user, talk;

(9) 用关键字limit限制查询结果的数量

查询数据时，可能会查询出很多记录，而用户需要的记录可能只是很少的一部分，这样就需要限制查询结果的数量。

示例：

select \* from tb\_login order by id asc limit 3;

select \* from tb\_login where id order by asc limit 1, 2;

## 9.3 聚合函数

聚合函数的最大特点就是它们根据一组数据求出一个值。聚合函数的结果值只根据选定行中非NULL的值进行计算，而NULL值将会被忽略。

### 9.3.1 COUNT()函数

COUNT()函数除了“\*”以外的任何参数，返回所选择集合中非NULL值的行的数目；对于参数“\*”，返回选择集合中所有行的数目，包含NULL值的行。没有WHERE子句的COUNT(\*)是经过内部优化的，能够快速返回表中所有的记录总数。

示例：

select count(\*) from tb\_login;

### 9.3.2 SUM()函数

SUM()函数可以求出表中某个字段取值的总和。

示例：

select sum(row) from tb\_books;

### 9.3.3 AVG()函数

AVG()函数可以求出表中某个字段取值的平均值。

示例：

select avg(row) from tb\_books;

### 9.3.4 MAX()函数

MAX()函数可以求出表中某个字段取值的最大值。

示例：

select max(row) from tb\_books;

### 9.3.5 MIN()函数

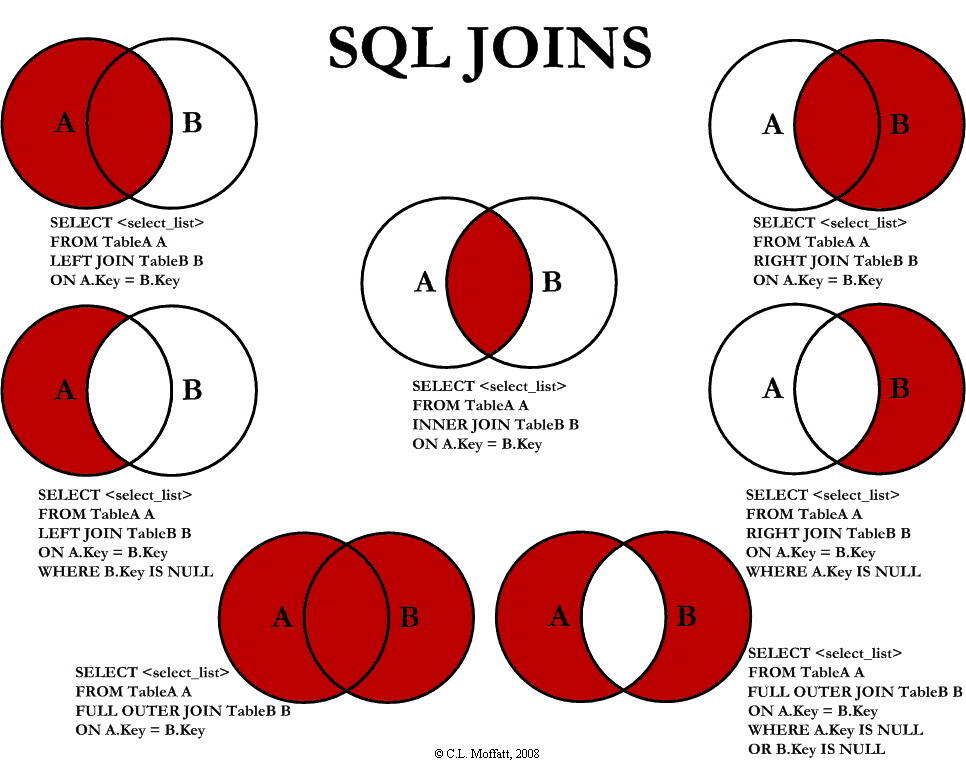
MIN()函数可以求出表中某个字段取值的最大值。

示例：

select min(row) from tb\_books;

## 9.4 连接查询

连接是把不同表的记录连到一起的最普遍的方法。



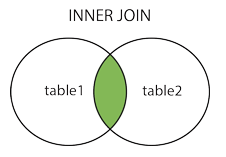
### 9.4.1 内连接查询

内连接是最普遍的连接类型，而且是最匀称的，因为它们要求构成连接的每一部分的每个表的匹配，不匹配的行就被排除。

内连接最常见的例子是相等连接，也就是连接后的表中的某个字段与每个表中的都相同。这种情况下，最后的结果集只包含参加连接的表中与指定字段相符的行。

示例：有tb\_login用户信息表和tb\_bok图书信息表。

select name, books from tb\_login, tb\_book where tb\_login=tb\_book.user;



### 9.4.2 外连接查询

与内连接不同，外连接是指使用OUTER JOIN关键字将两个表连接起来。外连接生成的结果集不仅包含符合连接条件的行数据，而且包括左表（左外连接时的表）、右表（右外连接时的表）或两边连接表（全外连接时的表）中所有的数据行。语法格式如下：

SELECT 字段名称 FROM表1 LEFT|RIGHT JOIN表2 ON表1.字段1 =表2.字段2;

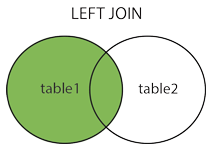
外连接分为左外连接（LEFT JOIN）、右外连接（RIGHT JOIN）和全外连接3种类型：

(1) 左外连接

左外连接是指将左表中的所有数据分别与右表中的每条数据进行连接组合，返回的结果除内连接的数据外，还包括左表中不符合条件的数据，并在右表的相应列中添加NULL值。

示例：

select section, tb\_login, user, books, row from tb\_login left join tb\_book on tb\_login.user = tb\_book.user;

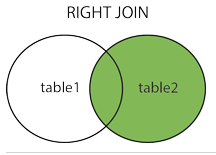


(2) 右外连接

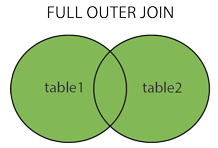
右外连接是指将右表中的所有数据分别与左表中的每条数据进行连接组合，返回的结果除内连接的数据外，还包括右表中不符合条件的数据，并在左表的相应列中添加NULL值。

示例：

select section, tb\_login, user, books, row from tb\_login right join tb\_book on tb\_login.user = tb\_book.user;



(3) 全外连接



### 9.4.3 复合条件连接查询

在连接查询时，也可以增加其他的限制条件。通过多个条件的复合查询，可以使得查询结果更加准确。

select section, tb\_book.user, books, row from tb\_book, tb\_login where tb\_book.user=tb\_login.user and row>5;

## 9.5 子查询

子查询就是SELECT查询是另一个查询的附属。MySQL4.1可以嵌套多个查询，在外面一层的查询中使用里面一层查询产生的结果集。这样就不是执行两个（或者多个）独立的查询，而是执行包含一个（或者多个）子查询的单独查询。

当遇到这样的多层查询时，MySQL从最内层的查询开始，然后从它开始向外向上移动到外层（主）查询，在这个过程中每个查询产生的结果集都被赋给包围它的父查询，接着这个父查询被执行，他的结果也被指定给父查询。

除了结果集经常由包含一个或多个值的一列组成外，子查询和常规SELECT查询的执行方式一样。子查询可以用在任何可以使用表达式的地方，它必须由父查询包围，其他，如同常规的SELECT查询。

### 9.5.1 带关键字IN的子查询

### 9.5.2 带比较运算符的子查询

### 9.5.3 带关键字EXISTS的子查询

### 9.5.4 带关键字ANY的子查询

### 9.5.5 带关键字ALL的子查询

## 9.6 合并查询结果

### 9.6.1 UNION

### 9.6.2 UNION ALL

## 9.7 定义表和字段的别名

### 9.7.1 为表取别名

### 9.7.2 为字段取别名

## 9.8 使用正则表达式查询

### 9.8.1 匹配指定字符的任意一个

### 9.8.2 使用“\*”和“+”来匹配多个字段

### 9.8.3 匹配以指定的字符开头和结束的记录

## 9.9 总结

# 第10章：常用函数

# 第11章：索引

索引是一个特殊的数据库结构，是提高数据库性能的重要方式，可以用来快速查询数据库表中的特定记录，MySQL中所有的数据类型都可以被索引。MySQL的索引包括普通索引、唯一性索引、全文索引、单列索引、多列索引和空间索引等。

## 11.1 索引概述

在MySQL中，索引由数据表中一列或多列组合而成，创建索引的目的就是为了优化数据库的查询速度。其中，用户创建的索引指向数据库中具体数据所在的位置。当用户通过索引查询数据库中的数据时，不需要遍历所有数据库中的所有数据，大幅度提高了查询效率。

### 11.1.1 MySQL索引概述

索引是一种将数据库中单列或者多列的值进行排序的结构。应用索引，可以大幅度的提高查询的速度。

用户通过索引查询数据，不但可以提高查询速度，还可以降低服务器的负载。用户查询数据时，系统可以不必遍历数据表中的所有记录，而是查询索引列。因此，使用索引可以有效的提高数据库系统的整体性能。

应用MySQL数据库时，并非用户在查询数据的时候，总需要应用索引来优化查询。凡事都有双面性，使用索引可以提高检索数据的速度，对于依赖关系的子表和父表之间的联合查询，可以提高查询速度，并且可以提高整体的系统性能。但是，创建索引和维护需要耗费时间，并且该耗费时间与数据量的大小成正比；另外，索引需要占用物理空间，给数据的维护造成很多麻烦。

整体来说，索引可以提高查询速度，但是会影响用户操作数据库的插入操作。因为，向有索引的表中插入记录时，数据库系统会按照索引进行排序。所以，用户可以删除索引后，插入数据，当数据插入操作完成后，用户可以重新创建索引。

### 11.1.2 MySQL索引分类

MySQL的索引包括普通索引、唯一性索引、全文索引、单列索引、多列索引和空间索引等。

(1)普通索引

(2)唯一性索引

(3)全外索引

(4)单列索引

(5)多列索引

(6)空间索引

## 11.2 创建索引

### 11.2.1 在建立数据表时创建索引

### 11.2.2 在已建立数据表中创建索引

### 11.2.3 修改数据表结构添加索引

## 11.3 删除索引

## 11.4 总结

# 第12章：视图

视图是从一个或多个表中导出的表，是一种虚拟存在的表。视图就像一个窗口，通过这个窗口可以看到系统专门提供的数据。这样，用户可以不用看到整个数据库表中的数据，而只关心对自己有用的数据。视图可以使用户的操作更加方便，而且可以保障数据库系统的安全性。

## 12.1 视图概述

### 12.1.1 视图的概念

### 12.1.2 视图的作用

## 12.2 创建视图

### 12.2.1 查看创建视图的权限

### 12.2.2 创建视图的步骤

### 12.2.3 创建视图的注意事项

## 12.3 视图操作

### 12.3.1 查看视图

### 12.3.2 修改视图

### 12.3.3 更新视图

### 12.3.4 删除视图

## 12.4 总结

# 第13章：数据完整性约束

# 第14章：存储过程与存储函数

# 第15章：触发器

# 第16章：事务的应用

在操作MySQL过程中，对于一般简单的业务逻辑或中小型程序而言，无须考虑应用MySQL事务。但是在比较复杂的情况下，往往在执行某些数据操作过程中，需要通过一组SQL语句执行多项并行业务逻辑或程序，这样，就必须保证所有命令执行的同步性，使执行序列中产生依赖关系的动作能够同时操作成功或同时返回初始状态。在此情况下，就需要优先考虑使用MySQL事务处理。

## 16.1 MySQL事务概述

在MySQL中，事务由单独单元的一条或多条SQL语句组成。在这个单元中，每条MySQL语句是相互依赖的。而整个单独单元作为一个不可分割的整体，如果单元中一旦某条SQL语句执行失败或产生错误，整个单元就要回滚，所有受到影响的数据将返回到事务开始以前的状态；如果单元中的所有SQL语句均执行成功，则事务被顺利执行。

在现实生活中，事务处理数据的应用非常广泛，如网上交易，银行事务等。网上购物中，添加商品到购物车，在线付款，商家发货等构成了一个基本的事务。整个交易流程可以被看作一个完整的单元，用于实现整体事务。

通过InnoDB和BDB类型表，MySQL事务能够完全满足事务安全的ACID测试，但是不是所有表类型都支持事务，如MyISAM类型表就不能支持事务，只能通过伪事务对表实现事务处理。

注：ACID提出每个事务型RDBMS必须遵守4个属性，即原子性、一致性、孤立性和持久性。

### 16.1.1 原子性

### 16.1.2 一致性

### 16.1.3 孤立性

### 16.1.4 持久性

## 16.2 MySQL事务的创建与存在周期

创建事务的一般过程是：初始化事务，创建事务，应用SELECT语句查询数据是否被录入和提交事务。如果用户不在操作数据库完成后执行事务提交，则系统会默认执行回滚操作。如果用户在提交事务前选择撤销事务，则用户在撤销前的所有事务将被取消，数据库系统会回到初始状态。

默认情况下，在MySQL中创建的数据表类型都是MyISAM，但是该类型的数据表并不支持事务。所以，如果用户想要数据表支持事务处理能力，必须将当前操作数据表的类型设置为InnoDB或BDB。

在创建事务的过程中，用户需要创建一个InnoDB或BDB类型的数据表，其基本命令结构如下：

CREATE TABLE 数据表名(field\_defintions) TYPE = INNODB/BDB;

其中field\_definitions为表内定义的字段等属性，TYPE指定数据表的类型，既可以是InnoDB类型，也可以是BDB类型。

### 16.2.1 初始化事务

### 16.2.2 创建事务

### 16.2.3应用SELECT语句查询数据是否被正确录入

### 16.2.4 提交事务

### 16.2.5 撤销事务（事务回滚）

### 16.2.6 事务的存在周期

## 16.3 MySQL事务行为

### 16.3.1 自动提交

### 16.3.2事务的孤立级

### 16.3.3修改事务的孤立级

## 16.4 事务的性能

### 16.4.1 应用小事务

### 16.4.2 选择合适的孤立级

### 16.4.3死锁的概念与避免

## 16.5 MySQL伪事务

### 16.5.1 用表锁定代替事务

### 16.5.2应用表锁实现伪事务

# 第17章：事件

# 第18章：备份和恢复

# 第19章：MySQL性能优化

性能优化是通过某些有效的方法提高MySQL数据库的性能。性能优化的目的是为了使MySQL数据运行速度更快，占用的磁盘空间更小。性能优化包括很多方面，例如优化查询速度，优化更新速度和优化MySQL服务器等。

## 19.1 优化概述

## 19.2 优化查询

### 19.2.1 分析查询语句

使用EXPLAIN和DESCRIBE可以用来分析查询语句，形式如下：

EXPLAIN|DESCRIBE SELECT语句;

### 19.2.2 索引对查询速度的影响

在查询过程中使用索引，势必会提高数据库的查询效率，应用索引来查询数据库中的内容，可以减少查询的记录数，从而达到查询优化的目的。

### 19.2.3 使用索引查询

在MySQL中，索引可以提高查询的速度，但并不能充分发挥其作用，所以在应用索引查询时，也可以通过关键字或其他方式来对查询进行优化处理。

(1) 应用关键词LIKE优化索引查询

(2) 查询语句中使用多列索引

(3) 查询语句中使用关键字OR

## 19.3 优化数据库结构

数据库结构是否存在合理，需要考虑是否存在冗余，对表的查询和更新速度，表中字段的数据类型是否合理等多方面的内容。

### 19.3.1 将字段很多的表分解成多个表

对于字段特别多且有些字段的使用频率较低的表，可以将其分解成多个表。

### 19.3.2 增加中间表

有时需要经常查询某两个表中的几个字段。如果经常进行联表查询，会降低MySQL数据库的查询速度。对于这种情况，可以建立中间表来提高查询速度。

先分析经常需要同时查询哪几个表中的哪些字段，然后将这些字段建立一个中间表，并将原来那几个表的数据插入到中间表中，之后就可以使用中间表来进行查询和统计。

### 19.3.3 优化插入记录的速度

插入记录时，索引、唯一性校检都会影响到插入记录的速度。而且，一次插入多条记录和多次插入多条记录锁耗费的时间是不一样的。根据这些情况，分别进行不同的优化。

(1) 禁用索引

(2) 禁用唯一性检查

(3) 优化insert语句

### 19.3.4 分析表，检查表和优化表

分析表主要作用是分析关键字的分布。检查表主要作用是检查表中是否存在错误。优化表主要作用是消除删除或者更新造成的空间浪费。

(1) 分析表

(2) 检查表

(3) 优化表

## 19.4 查询告诉缓存

在MySQL中，用户通过SELECT语句查询数据时，该操作将结果集保存到一个特殊的高级缓存中，从而实现查询操作。首次查询后，当用户再次做相同查询操作时，MySQL即可从高速缓存中检索结果。这样一来，既提高了查询速度，也起到了优化查询的作用。

### 19.4.1 检验高速缓存是否开启

### 19.4.2 使用高速缓存

## 19.5 优化多表查询

在MySQL中，用户可以通过连接来实现多表查询，在查询过程中，用户将表中的一个或多个共同字段进行连接，定义查询条件，返回统一的查询结果。这通常用来建立RDBMS常规表之间的关系。在多表查询中，可以应用子查询来优化多表查询，即在SELECT语句中嵌套其他SELECT语句。采用子查询优化多表查询的好处有很多，其中，可以将分步查询的结果整合成一个查询，这样就不需要再执行多个单独查询，从而提高了多表查询的效率。

## 19.6 优化表设计

在MySQL中，为了优化查询，使查询能够更加精炼、搞笑，在用户设计数据表的同时，也应该考虑一些因素。

## 19.7 总结

# 第20章：权限管理及安全控制

# 第21章：PHP管理MySQL数据库中的数据

# 第22章：Apache+PHP+MySQL实现网上社区

# 第23章：Struts+Spring+Hibernate+MySQL实现网络商城