1. 创建数据库

create database 数据库名;

1. 删除数据库

drop database 数据库名;

1. 使用数据库

use 数据库名;

1. 显示数据库下多少张表

show tables;

1. 查看表结构

desc 表名

1. 创建表

create table test

(

#整数通常用int

text\_id int,

#小数点数

test\_price decimal,

#普通长度文本，使用default指定默认值

test\_name varchar(255) default ‘xxx’,

#大文本类型

test\_desc text,

#图片

test\_img blob,

test\_date datetime

);

1. 使用子查询建表

#创建hehe数据表，该数据表和user\_inf完全相同，数据也完全相同

create table hehe

as

select \* from user\_inf;

1. 修改表结构，增加字段

#为hehe数据表增加一个hehe\_id字段，该字段类型为int

alter table hehe

add hehe\_id int;

#为hehe数据表增加aaa、bbb字段，两个字段的类型都为varchar(255)

alter table hehe

add

(

aaa varchar(255) default ‘xxx’,

bbb varchar(255)

);

1. 修改表结构，修改字段

#将hehe表的hehe\_id列修改为varchar(255)类型

alter table hehe

modify hehe\_id varchar(255);

#将hehe表的bbb列修改成int类型

alter table hehe

modify bbb int;

1. 修改表结构，删除字段

alter table 表名

drop column\_name;

1. 重命名表

alter table 表名

rename to 新表名

1. 重命名字段

#将wawa数据表的bbb字段重命名为ddd

alter table wawa

change bbb ddd int;

1. 删除表

#删除表wawa

drop table wawa;

1. 删除表但保留表结构

truncate 表名

1. 数据库约束

NOT NULL: 非空约束，字段不能为空

UNIQUE: 唯一约束，指定某字段或几个字段不能重复

PRIMARY KEY:主键，指定该字段的值可以唯一地标识该条记录

FOREIGN KEY：外键 指定该行记录从属于主表中的一条记录，主要用于保证参照完整性

CHECK: 检查，指定一个布尔表达式，用于指定对应列的值必须满足该表达式

(Mysql 不支持CHECK约束)

1. 建表时，字段约束语法

create table heh

(

#建立了非空约束，这意味着hehe\_id不可以为null

hehe\_id int not null,

#MySQL的非空约束不能指定名字

hehe\_name varchar(255) default ‘xyz’ not null,

#下面列可以为空,默认就是可以为空

hehe\_gender varchar(2) null，

#unique唯一约束，使用列级语法建立唯一约束

hehe\_fingerprint varchar(255) unique

);

1. 修改表时，字段约束语法

#增加非空约束

alter table hehe

modify hehe\_gender varchar(2) not null;

#取消非空约束

alter table hehe

modify hehe\_name varchar(2) null;

#取消非空约束，并指定默认值

alter table hehe

modify hehe\_name varchar(255) default ‘abc’ null;

1. 表级别约束

#建表时创建唯一约束，使用表级约束语法建立约束

create table unique\_test2

(

test\_id int nor null,

test\_name varchar(255),

test\_pass varchar(255),

unique (test\_name),

constraint test2\_uk unique(test\_pass)

);

test2\_uk 就是表级约束名，可以删除

create table unique\_test2

(

test\_id int nor null,

test\_name varchar(255),

test\_pass varchar(255),

#指定两字段组合不允许重复

constraint test3\_uk unique(test\_name,test\_pass)

);

#增加唯一约束

alter table unique\_test3

add unique(test\_name,test\_pass);

1. 删除表级约束

#删除unique\_test3 表上的test3\_uk唯一约束

alter table unique\_test3

drop index text3\_uk;

1. 主键

每个表只有一个主键，但主键可以由多个字段组成。

单个字段的主键，不允许重复和为空。

多个字段的主键，任何一个字段都不允许为空，且这些字段的组合不允许重复。

create table primary\_test

(

test\_id int primary key,

test\_name varchar(255)

);

create table primary\_test2

(

test\_id int not null,

test\_name varchar(255),

test\_pass varchar(255),

constraint test2\_pk primary key(test\_id)

);

create table primary\_test3

(

test\_name varchar(255),

test\_pass varchar(255),

primary key(test\_name,test\_pass)

);

1. 删除主键

alter table primary\_test3

drop primary key;

1. 增加主键

alter table primary\_test3

add primary key(test\_name,test\_pass);

1. 自增长主键

create table primary\_test4

(

test\_id int auto\_increment primary key,

test\_name varchar(255),

test\_pass varchar(255)

);

1. 外键

外键用于确保两个字段的参照关系。两个字段来自一个表或两个不同的表。

而子表外键字段的值必须在主表参照字段的值范围内，或者为空。

Mysql中无法用字段约束语法创建外键。只有表约束语法

create table teacher\_table1

(

teacher\_id int auto\_increment,

teacher\_name varchar(255),

primary key(teacher\_id)

);

create table sudent\_table1

(

student\_id int auto\_increment primary key,

student\_name varchar(255),

java\_teacher int,

foreign key(java\_teacher) references teacher\_table1(teacher\_id)

);

Mysql会自动给外键约束创建约束名：table\_name\_ibfk\_n,table\_name是从表的表名，而n是从1开始的整数。

1. 外键表级约束

constraint student\_teacher\_fk foreign key(java\_teacher)

references teacher\_table2(teacher\_id);

1. 多字段组合的外键约束

多字段组合的外键约束，则必须使用表级约束语法。

create table teacher\_table3

(

teacher\_name varchar(255),

teacher\_pass varchar(255),

#以两字段建立组合主键

primary key(teacher\_name,teacher\_pass)

);

create table student\_table3

(

student\_id int auto\_increment primary key,

student\_name varchar(255),

java\_teacher\_name varchar(255),

java\_teacher\_pass varchar(255),

#使用表级约束语法建立外键约束

foreign key(java\_teacher\_name,java\_teacher\_pass)

references teacher\_table3(teacher\_name,teacher\_pass)

);

1. 删除外键约束

alter table student\_table3

drop foreign key student\_table3\_ibfk\_1;

1. 增加外键约束

alter table student\_table3

add foreign key(java\_teacher\_name,java\_teacher\_pass)

references teacher\_table3(teacher\_name,teacher\_pass);

1. 自关联外键约束

表的外键是表自身的字段,

同一张表里面的员工和经理，可能就要用到自关联

create table foreign\_test

(

foreign\_id int auto\_increment primary key,

foreign\_name varchar(255),

#使用该表的refer\_id参照到本表的foreign\_id字段

refer\_id int,

foreign key(refer\_id) references foreign\_test(foreign\_id)

);

1. 删除主表记录时，同时删除从表。

需要在建立外键约束后添加on delete cascade，删除主表记录时，把参照该主表记录的从表记录全部级联删除。

或添加on delete set null，删除主表记录时，把参照该主表记录的从表记录的外键设为null。

create table teacher\_table4

(

teacher\_id int auto\_increment,

teacher\_name varchar(255),

primary key(teacher\_id)

);

create table student\_table4

(

student\_id int auto\_increment primary key,

student\_name varchar(255),

java\_teacher int,

foreign key(java\_teacher) references teacher\_table4(teacher\_id)

on delete cascade #也可用 on delete set null

);

1. CHECK约束

MySQL中这个约束不会有任何作用

create table check\_test

(

emp\_id int auto\_increment,

emp\_name varchar(255),

emp\_salary decimal,

primary key(emp\_id),

#建立CHECK约束

check(emp\_salary>0)

);

1. 索引

索引是存放在模式(schema)中的一个数据库对象，索引从属于表，但它也和数据表一样属于数据库对象。创建索引的唯一作用就是加速对表的查询，索引通过快速路径访问方法来快速定位数据，减少磁盘I/O。

MySQL使用information\_schema数据库里的STATISTICS表来保存该数据库实例中的所有索引信息，用户可通过查询该表来获取该数据库的索引信息。

索引创建方式，

自动：当在表上定义主键约束、唯一约束和外键约束时，系统会为该字段自动创建对应的索引。 当表被删除时，表上索引自动被删除。

手动：用户可以通过create index...语句来创建索引。 可通过drop index... 来删除指定数据表上的索引。

1. 单列索引

#创建名为emp\_last\_name\_idx的索引，提供employees表last\_name自动的查询速度

create index emp\_last\_name\_idx

on employee(last\_name);

1. 多列索引

#下面语句

create index emp\_last\_name\_idx2

on employee(first\_name,last\_name);

1. 删除索引

drop index emp\_last\_name\_idx2

on employees;

有些数据库删除索引无需指定表名，因为它们要求建立索引时每个索引有唯一名字。

当数据表中记录被添加、删除、修改时，数据库系统需要维护索引，有一定开销。

同时存储索引信息需要一定的磁盘空间。

1. 视图

视图时一个或多个数据表中数据的逻辑显示。视图好处：

* 1. 可以限制对数据的访问
  2. 可以使复杂的查询变得简单
  3. 提供了数据的独立性
  4. 提供了对相同数据的不同显示

#创建视图,若视图名已存在，则

create or replace view 视图名

as subquery;

从语意上可见，视图的本质，就是一条SQL查询语句。

视图建立后，使用视图与使用数据表就没有区别了，但通常只是查询数据。

相当于建立临时表，优化查询

MySQL允许在创建视图时使用with check option子句，使其不得修改

create or replace view view\_test

as

select teacher\_name , teacher\_pass from teacher\_table

#指定不允许修改该视图的数据

with check option;

#Oracle采用with read only

1. 删除视图

drop view 视图名;

1. 插入语句

insert into teacher\_table2(teacher\_name)

values(‘xyz’);

#不在表名后列出字段，则默认插入所有字段。

#需要为所有字段指定值，如果值不确定，则为该字段分配null

insert into teacher\_table2

#使用null代替主键字段的值

values(null,’abc’);

#若主键值为auto\_increment,则即使插入null，也会分配值

1. 子查询插入

insert into sutdent\_table2(student\_name)

#子查询的值会被作为插入值

select teacher\_name from teacher\_table2;

1. 插入多个值

insert into teacher\_table2

#同时插入多个值

values(null,”Yeeku”),

(null,”Sharfky”);

1. 更新

update teacher\_table2

set teacher\_name=’孙悟空’;

update teacher\_table2

set teacher\_name=’猪八戒’,teacher\_id=123

where teacher\_id>1;

1. 删除

delete from student\_table2;

delete from teacher\_table2

where teacher\_id>2;

1. 单表查询

select \* from table;

select student\_name from student\_table where java\_teacher>3;

select语句中还可以使用算数运算符(+、-、\*、/),形成算数表达式

规则如下：

* 1. 对数值型数据字段、变量、常量可以使用算术运算符(+、-、\*、/)创建表达式
  2. 对日期型数据字段、变量、常量可以使用部分算术表达式(+、-、\*、/)创建表达式，两个日期之间可以使用减法运算，日期和数值之间可以使用加、减运算。
  3. 运算符可以在两个字段间进行运算

1. 单表查询运算符举例

select teacher\_id+5 from teacher\_table;

select \* from teacher\_table where teacher\_id\*3>4;

select 3\*5,20 from teacher\_table;

1. 单表查询，使用函数

#concat函数实现字符串连接

select concat(teacher\_name,’是sb’) from teacher\_table;

#MySQL中若使用null，会使整个算术表达式返回值为null

select concat(teacher\_name,null) from teacher\_table;

1. 单表查询，别名

select teacher\_id + 5 as MY\_ID from teacher\_table;

#若别名中有特殊符号(单引号等)，或大小写敏感，则别名需加双引号

select teacher\_id + 5 “MY’ID” from teacher\_table;

#多字段起别名，别名与字段名空格隔开，字段与字段间逗号隔开

select teacher\_id + 5 MY\_ID , teacher\_name 老师名

from teacher\_name;

#为表起别名，与字段语法一致。

select teacher\_id + 5 MY\_ID , teacher\_name 老师名

from teacher\_table t;

1. 单表查询，字段使用运算符

select teacher\_id + 5 MY\_ID , concat(teacher\_name , teacher\_id) teacher\_name

from teacher\_table

where teacher\_id \* 2 > 3;

#甚至可以这样

select 5+4

from teacher\_table

where 2<9;

1. 单表查询，去重

select distinct student\_name,java\_teacher

from student\_table;

distinct关键字紧跟select之后，去除后面字段组合的重复值。

1. 特殊比较运算符

#expr1>=expr2并且expr1<=expr3

expr1 between expr2 and expr3

#expr1等于后面括号里任意一个值

expr1 in(expr2,expr3...)

#字符串匹配，like后的字符串支持通配符,下划线\_表示任意一个字符，%表示

like

#要求指定值等于null

is null

例：

select \* from student\_table

where student\_id between 2 and 4;

select \* from student\_table

where 2 between java\_teacher and student\_id;

selct \* from student\_table

where student\_id in(2,4);

select \* from student\_table

where 2 in(student\_id, java\_teacher);

select \* from student\_table

where student\_name like ‘\_\_’;#两个下划线

#反斜线\作为转义字符，选出学生姓名以下划线开头的

select \* from student\_table

where student\_name like ‘\\_%’;

#标准SQL需要用escape关键字显示标记转义

select \* from student\_table

where student\_name like ‘\\_%’ escape ‘\’;

#无法用=null来判断是否为空，因为null=null返回null

select \* from studfent\_table

where student\_name is null;

1. where语句多个条件组合（and、or、not）

select \* from student\_table

#and来组合多个条件

where student\_name like ‘\_\_’ and student\_id>3;

select \* from student\_table

#使用not对where条件取否

where not student\_name like ‘\\_%’;

select \* from student\_table

#使用括号强制先计算or运算

where (student\_id>3 or student\_name>’张’)

and java\_teacher >1;

1. 查询结果排序

默认按插入顺序排序，使用order by子句可以按某字段大小以升序排序

select \* from student\_table

order by java\_teacher;

#使用降序排列需用desc关键字，升序排列关键字asc(默认为此)

select \* from student\_table

order by java\_teacher desc , student\_name;

1. 数据库函数

函数通过对一组数据进行计算，得到最终的一个返回值。

函数可以出现在SQL语句中各个位置，比较常用的位置是select之后和where子句中。

#计算student\_table表中的记录数

select count(\*) from student\_table;

#计算java\_teacher列总共多少个值 distinct去重

select count(distinct java\_teacher) from student\_table;

#统计所有student\_id的和

select sum(student\_id) from student\_table;

#计算的结构是20\*记录的行数

selectg sum(20) from student\_table;

#选出student\_table表中student\_id最大的值

select max(student\_id) from student\_table;

#student\_id最小值  
select min(student\_id) from student\_table;

#count统计行数时，null不会被记录在内

select count(student\_name) from student\_table;

#可能出现null的的字段，可以用ifnull来处理

#计算java\_teacher字段所有记录的平均值

select avg(ifnull(java\_teacher,0)) from student\_table;

1. 分组

group by 后跟一个或多个字段名，将字段值或字段组合值完全相同时，分为一组。

#count(\*)将会对每组得到一个结果

#将java\_teacher列值相同的记录当成一组

select count(\*) from student\_table group by java\_teacher;

1. having过滤

where和having都有过滤功能，

where不能过滤组，仅过滤行。过滤组必须使用having；

where中无法使用组函数，having子句才可使用组函数；

#对组进行过滤

select \* from student\_table group by java\_teacher having count(\*)>2;

1. 多表查询，SQL92规范

多表查询中，多个表可能字段名相同，为免混淆，可使用表名前缀和表别名前缀。

#指定多个数据表，并指定表别名

#使用where指定连接条件

select s.\*,teacher\_name

from student\_table s,teacher\_table t

where s.java\_teacher=t.teacher\_id;

#去掉where条件得到广义笛卡尔积,得到n\*m条结果

select s.\*,teacher\_name from student\_table s,teacher\_table t;

#非等值连接

select s.\*,teacher\_name

from student\_table s,teacher\_table t

where s.java\_teacher > t.teacher\_id;

#执行过滤，过滤条件和连接条件用and连接

select s.\*,teacher\_name

from student\_table s,teacher\_table t

where s.java\_teacher=t.teacher\_id and student\_name is not null;

MySQL不支持SQL92中的左外连接、右外连接，但可使用自连接

自连接的本质是一个表当两个表用，需要为一个表取两个别名

select emp.emp\_id , emp.emp\_name 员工名 , mgr.emp\_name 经理名

from emp\_table emp,emp\_table mgr

where emp.manager\_id=mgr.emp\_id;

1. 多表查询，SQL99规范

#SQL 99多表连接查询的from后只有一个表名

#cross join交叉连接，相当于广义笛卡儿积

select s.\* , teacher\_name

from student\_table s

cross join teacher\_table t;

#natural join 自然连接，使用两表中同名字段作为连接条件

select s.\*,teacher\_name

from student\_table s

natural join teacher\_table t;

#using join 显示指定哪些同名字段作为连接条件

#natural join会在多个字段同名情况下把每个字段作为连接条件

#using join 解决这个问题。

select s.\* , teacher\_name

from student\_table s

join teacher\_table t

using(teacher\_id);

#on子句连接，on子句只指定一个连接条件，需要进行N表连接，则需要N-1个join..on

select s.\* , teacher\_name

from student\_table s

join teacher\_table t

on s.java\_teacher=t.teacher\_id;

#on 子句连接非等值条件

select s.\* , teacher\_name

from student\_table s

join teacher\_table t

on s.java\_teacher>t.teacher\_id

#右连接 右表全部值出现，左表没有匹配值，则为null

select a.\*,b.\* from a right join b on a.id=b.parent\_id

结果是

1 张3 1 23 1

2 李四 2 34 2

null 3 34 4

#左连接 左表全部值出现，右表没有匹配值，则为null

select a.\*,b.\* from a left join b on a.id=b.parent\_id

结果是

1 张3 1 23 1

2 李四 2 34 2

3 王武 null

#全连接 左右表值全部出现

select a.\*,b.\* from a full join b on a.id=b.parent\_id

结果是

1 张3 1 23 1

2 李四 2 34 2

null 　　 3 34 4

3 王武 null

1. 子查询

子查询即查询语句中嵌套另一个查询，支持多重嵌套

* 1. 出现在from语句后，被称为行内视图，实质是一个临时表
  2. 出现在where条件后作为过滤条件的值

使用子查询时需要注意

* 1. 子查询要用括号括起来
  2. 子查询在from后作为临时表时，可取表别名，尤其是作为前缀限定数据字段时，必须取别名
  3. 子查询在where后作为过滤条件时，将子查询放在比较运算符的右边，优化阅读
  4. 子查询在where后作为过滤条件时，单行子查询使用单行运算符，多行子查询使用多行运算符

#子查询作为临时表

select \* from (select \* from student\_table) t

where t.java\_teacher>1;

#返回单行、单字段的子查询可以当标量值使用

select \* from student\_table

where java\_teacher>

(select teacher\_id from teacher\_table where teacher\_name=’Yeeku’);

#子查询返回多个值，使用in、any、all关键字

select \* from student\_table where student\_id

in(select teacher\_id from teacher\_table);

#any关键字必须与一个比较操作符一起使用，表示任意值为true，则返回true

select s1 from t1 where s1>any(select s1 from t2);

#all关键字必须与一个标胶操作符一起使用，表示全部值为true，才为true

select s1 from t1 where s1>any(select s1 from t2);

#some是any的别名

1. 集合运算

select语句的查询结果是一个包含多条数据的结果集，类似于集合，可以进行

交(intersect)、并(union)、和差(minus)运算。

对两个结果集进行运算必须满足条件：

* 1. 两个结果集所包含的数据字段数量必须相等
  2. 两个结果集所包含的数据字段的类型也必须相等

#查询结果集包含两个字段，第一个int类型，第二个varchar类型

select \* from teacher\_table

union

#这个结果集的数据字段必须与前一个结果及的数据字段一一对应

selecrt student\_id , student\_name from student\_table;

#MySQL不支持minus运算，但可曲线救国

select student\_id , student\_name from student\_table

where(student\_id , student\_name)

not in

(select teacher\_id, teacher\_name from teacher\_table);

#MySQL不支持intersect运算，但可曲线救国

select student\_id , student\_name from student\_table

join

teacher\_table

on(student\_id=teacher\_id and student\_name=teacher\_name)

where student\_id<4 and teacher\_name like ‘李%’;