## 1. 数据库基本增删改查语句

mysql管理员的用户名和密码：root root

创建一个名称为mydb1的数据库

create database mydb1;

show databases;

创建一个使用utf-8字符集的mydb2数据库。

create database student1 character set utf8;

创建一个使用utf-8字符集，并带校对规则的mydb3数据库。

create database mydb3 character set utf8 collate utf8\_general\_ci;

查看前面创建的mydb2数据库的定义信息

show create database mydb2;

删除前面创建的mydb1数据库

drop database mydb1;

查看服务器中的数据库，并把其中某一个库的字符集修改为gb2312;

alter database mydb2 character set gb2312;

show create database mydb2;

演示恢复和备份

create database tt;

use tt;

create table a

(

name varchar(20)

);

insert into a(name) values('aaaa');

select \* from a;

-----看到a表有数据

对tt作备份操作，启动一个window命令行窗口，执行如下命令

mysqldump -uroot -p tt>c:\tt.sql

演示恢复

1.先删除库

drop database tt;

2.恢复tt库(1)

2.1 为恢复库，要先创建库 create database tt;

2.2 再恢复tt库

use tt;

source c:\tt.sql （source:可以执行一个 sql脚本）

3.恢复tt库(2)

2.1 为恢复库，要先创建库 create database tt;

2.2 恢复库 mysql -uroot -proot tt<c:\1.sql; (window命令)

创建一个员工表

use mydb2;

create table employee

(

id int,

name varchar(40),

sex varchar(4),

birthday date,

entry\_date date,

job varchar(40),

salary decimal(8,2),

resume text

);

show tables; 查看库的所有表(查看库里的表要先打开库)

show create table employee; 查看表的创建细节

desc employee; 看表结构

在上面员工表的基本上增加一个image列。

alter table employee add image blob;

修改job列，使其长度为60。

alter table employee modify job varchar(60);

删除sex列

alter table employee drop sex;

表名改为user。

rename table employee to user;

修改表的字符集为utf-8

alter table user character set utf8;

列名name修改为username

alter table user change column name username varchar(40);

删除表

drop table user;

使用insert语句向表中插入三个员工的信息。

rename table user to employee;

insert into employee(id,username,birthday,entry\_date,job,salary,resume) values(1,'aaa','1980-09-09','1980-09-09','bbb',90,'aaaaa');

select \* from employee;

插入数据的细节1

insert into employee values(1,'aaa','1980-09-09','1980-09-09','bbb',90,'aaaaa');

插入数据的细节2

insert into employee values('1','aaa','1980-09-09','1980-09-09','bbb','90','aaaaa');

插入数据的细节3(插入中文)

要告诉mysql客户采用gb2312编码

show variables like 'chara%';

set character\_set\_client=gb2312;

要想查看时不乱码

show variables like 'chara%';

set character\_set\_results=gb2312;

select \* from employee;

ztp关于编码格式，注：

1. student.sql文件的编码格式：gb2312;

2. mysql的编码格式：

* Client: gb2312;
* Result: gb2312;
* Database: utf8
* Server: utf8

将mysql的ini文件改为：mysql 🡪 client 🡪 gb2312：是改除了server和database之外的所有编码格式为gb2312. 嫌麻烦可以改这里。

将所有员工薪水修改为5000元。

update employee set salary=5000;

将姓名为’bbb’的员工薪水修改为3000元。

update employee set salary=3000 where username='bbb';

将姓名为’bbb的员工薪水修改为4000元,job改为ccc。

update employee set salary=4000,job='ccc' where username='bbb';

将bbb的薪水在原有基础上增加1000元。

update employee set salary=salary+1000 where username='bbb';

更新要注意的问题

update employee set username='ccc',salary=9000,birthday='1980-09-09',.....................

update where id=1;

删除表中名称为’zs’的记录。

delete from employee where username='bbb';

删除表中所有记录。

delete from employee;

使用truncate删除表中记录。

truncate table employee;

## 2. Student库的student表，练习使用select from语句（单表查询）

查询表中所有学生的信息。

select \* from student;

查询表中所有学生的姓名和对应的英语成绩。

select name,english from student;

过滤表中重复的英语数据。

select distinct english from student;

在所有学生总分上加10分特长分。

select name,(chinese+english+math)+10 from student;

统计每个学生的总分。

select name,(chinese+english+math) from student;

使用别名表示学生分数。

select name as 姓名,(chinese+english+math)+10 as 总分 from student;

select name 姓名,(chinese+english+math)+10 总分 from student;

查询姓名为wu的学生成绩

select \* from student where name='王五';

查询英语成绩大于90分的同学

select \* from student where english>'90';

查询总分大于200分的所有同学

select name from student where (chinese+english+math)>200;

查询英语分数在 80－90之间的同学。

select name from student where english>80 and english<90;

select name from student where english between 80 and 90; == select name from student where english>=80 and english<=90;

查询数学分数为89,90,91的同学。

select \* from student where math in(89,90,91);

查询所有姓李的学生成绩。

select \* from student where name like '李%';

select \* from student where name like '李\_';

查询数学分>80，语文分>80的同学。

select \* from student where math>80 and chinese>80;

对数学成绩排序后输出。

select name,math from student order by math;

对总分排序后输出，然后再按从高到低的顺序输出

select name 姓名,(chinese+english+math) 总分 from student order by (chinese+english+math) desc;

select name 姓名,(chinese+english+math) 总分 from student order by 总分 desc;

对姓李的学生成绩排序输出

select \* from student where name like '李%' order by (chinese+english+math) desc;

统计一个班级共有多少学生？

select count(name) from student;

select count(\*) from student;

统计数学成绩大于90的学生有多少个？

select count(\*) from student where math>80;

统计总分大于250的人数有多少？

select count(\*) from student where (chinese+english+math)>250;

关于 count的函数的细节 （count只统有值的行）

统计一个班级数学总成绩？

select sum(math) from student;

统计一个班级语文、英语、数学各科的总成绩

select sum(chinese),sum(english),sum(math) from student;

统计一个班级语文、英语、数学的成绩总和

select sum(chinese+english+math) from student;

统计一个班级语文成绩平均分

select sum(chinese)/count(\*) from student;

统计一个班级语文成绩平均分

select avg(chinese) from student;

求一个班级总分平均分

select avg(chinese+math+english) from student;

求班级最高分和最低分

select max(chinese+math+english),min(chinese+math+english) from student;

对订单表中商品归类后，显示每一类商品的总价

select product,sum(price) from orders group by product;

（归组！）

查询购买了哪些（几类）商品，并且每类总价大于100的商品

select product from orders group by product having sum(price)>100;

注意：where后面不能跟合计语句，having后面可以跟合计语句，一般跟在order by 后面。

## 3. 数据库表的约束

定义主键约束（每一个表必须有且只有一个主键列）

create table student

(

id int primary key,

name varchar(40)

);

定义主键自动增长

create table student

(

id int primary key auto\_increment,

name varchar(40)

);

定义唯一约束

drop table student;

create table student

(

id int primary key auto\_increment,

name varchar(40) unique

);

定义非空约束

drop table student;

create table student

(

id int primary key auto\_increment,

name varchar(40) unique not null

);

定义外键约束

create table husband

(

id int primary key,

name varchar(40)

);

create table wife

(

id int primary key,

name varchar(40),

husband\_id int,

constraint husband\_id\_FK foreign key(husband\_id) references husband(id)

);

## 4. 数据库表的设计

### 1. 一对多

表的设计方案: 在多的一方加外键

部门和员工

create table department

(

id int primary key,

name varchar(40)

);

create table employee

(

id int primary key,

name varchar(40),

salary decimal(8,2),

department\_id int,

constraint department\_id\_FK foreign key(department\_id) references department(id)

);

### 2. 多对多

建3张表，student, teacher, student\_teacher(中间表)

多对多对象的表的设计（老师和学生）

create table teacher

(

id int primary key,

name varchar(40),

salary decimal(8,2)

);

create table student

(

id int primary key,

name varchar(40)

);

create table teacher\_student

(

teacher\_id int,

student\_id int,

primary key(teacher\_id,student\_id),

constraint teacher\_id\_FK foreign key(teacher\_id) references teacher(id),

constraint student\_id\_FK foreign key(student\_id) references student(id)

);

注： primary key(teacher\_id,student\_id), 这是联合主键，即这两列的值组成的数组项，不可重复，不可为空。

或者这么设计：给中间表单独配一个主键

create table teacher\_student

(

Id int primary key,

teacher\_id int,

student\_id int,

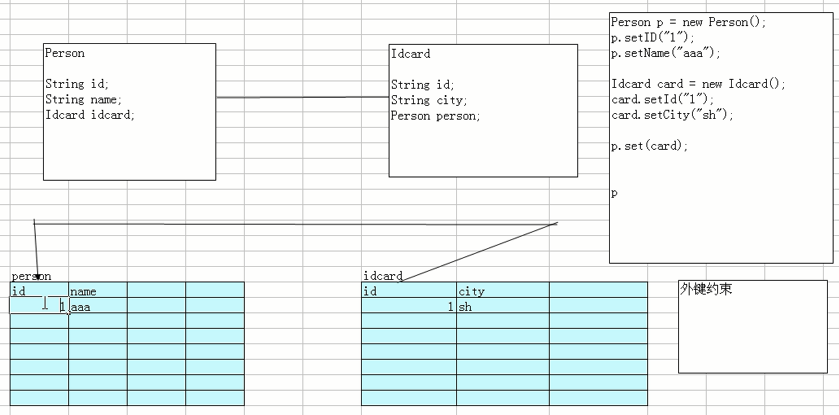
constraint teacher\_id\_FK foreign key(teacher\_id) references teacher(id),

constraint student\_id\_FK foreign key(student\_id) references student(id)

);

### 3. 一对一

一对一的对象的数据库设计，有主从关系，在从表加外键，此外键还要加：非空，唯一的约束。在一般的情况下，直接把从表的主键列设置成为外键列，自带唯一和非空约束。

Eg：

create table person

(

id int primary key,

name varchar(40)

);

create table idcard

(

id int primary key,

city varchar(40),

constraint id\_FK foreign key(id) references person(id)

);

### 4. 自连接的表

create table person

(

id int primary key,

name varchar(40),

parent\_id int,

constraint parent\_id\_FK foreign key(parent\_id) references person(id)

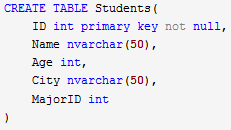
);

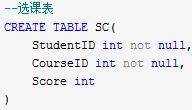
多对多的关系时：（毕业设计的讨论点）

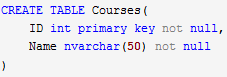
查询性能要求高：则使用单表，缺点是会有冗余项；

没有冗余项： 则使用3表方案；

## 5.数据库的单表查询：







### 1、Where（条件）

作用：按照一定的条件查询数据

语法：

SELECT 列名1,列名2 FROM 表名 WHERE 列名1 运算符 值

运算符：

|  |  |
| --- | --- |
| **运算符** | **描述** |
| = | 等于 |
| <> | 不等于 |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| BETWEEN | 在某个范围内 |
| LIKE | 搜索某种模式 |

比较操作符都比较简单，不再赘述。关于BETWEEN和LIKE，专门拿出来重点说下

### 2、BETWEEN

在两个值之间，比如我从学生中查询年龄在18-20之间的学生信息

SELECT ID,Name,Age FROM Students WHERE Age BETWEEN **18** AND **20**

### 3、LIKE

作用：模糊查询。LIKE关键字与通配符一起使用

主要的通配符：

|  |  |
| --- | --- |
| **通配符** | **描述** |
| **%** | 替代一个或多个字符 |
| **\_** | 仅替代一个字符 |
| **[charlist]** | 字符列中的任何单一字符 |
| **[^charlist]**  **或者**  **[!charlist]** | 不在字符列中的任何单一字符 |

实例：

1）查询姓氏为张的学生信息

SELECT ID,Name FROM Students WHERE Name LIKE '张%'

 2）查询名字最后一个为“生”的同学

SELECT ID,Name FROM Students WHERE Name LIKE '%生'

3）查询名字中含有“生”的学生信息

SELECT ID,Name FROM Students WHERE Name LIKE '%生%'

4）查询姓名为两个字，且姓张学生信息

SELECT ID,Name FROM Students WHERE Name LIKE '张\_'

5)查询姓氏为张、李的学生信息

这个可以使用or关键字，但是使用通配符更简单高效

SELECT ID,Name FROM Students WHERE Name LIKE '[张李]%'

6)查询姓氏非张、李的学生信息

这个也可以使用NOT LIKE 来实现，用下面方法更好。

SELECT ID,Name FROM Students WHERE Name LIKE '[^张李]%'

或者：

SELECT ID,Name FROM Students WHERE Name LIKE '[!张李]%'

### 4、AND

AND 在 WHERE 子语句中把两个或多个条件结合起来。表示和的意思，多个条件都成立。

1）查询年龄大于18且姓张的学生信息

SELECT ID,Name FROM Students WHERE Age>**18** AND Name LIKE '张%'

### 5、OR

 OR可在 WHERE 子语句中把两个或多个条件结合起来。或关系，表示多个条件，只有一个符合即可。

1)查询姓氏为张、李的学生信息

SELECT ID,Name FROM Students WHERE Name LIKE '张%' OR Name LIKE '李%'

### 6、IN

IN 操作符允许我们在 WHERE 子句中规定多个值。表示：在哪些值当中。

1）查询年龄是18、19、20的学生信息

SELECT ID,Name FROM Students WHERE Age IN (**18**,**19**,**20**)

### 7、NOT 否定

NOT对于条件的否定，取非。

1）查询非张姓氏的学习信息

SELECT ID,Name FROM Students WHERE Name NOT LIKE '张%'

### 8、ORDER BY（排序）

功能：对需要查询后的结果集进行排序

|  |  |  |
| --- | --- | --- |
| **标识** | **含义** | **说明** |
| ASC | 升序 | 默认 |
| DESC | 倒序 |  |

实例：

1）查询学生信息表的学号、姓名、年龄，并按Age升序排列

SELECT ID,Name,Age FROM Students ORDER BY Age

或指明ASC

SELECT ID,Name,Age FROM Students ORDER BY Age ASC

2)查询学生信息，并按Age倒序排列

SELECT ID,Name,Age FROM Students ORDER BY Age DESC

除了制定某个列排序外，还能指定多列排序，每个排序字段可以制定排序规则

说明：优先第一列排序，如果第一列相同，则按照第二列排序规则执行，以此类推。

3）查询学生的信息，按照总成绩倒序、学号升序排列

SELECT ID,Name,Score FROM Students ORDER BY Score DESC,ID ASC

这个查询含义：首先按Score倒序排列，如果有多条记录Score相同，再按ID升序排列。

查询结果，例子：

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Score** |
| 2 | 广坤 | 98 |
| 3 | 老七 | 98 |
| 1 | 赵四 | 79 |

### 9、AS(Alias)

可以为列名称和表名称指定别名（Alias）

作用：我们可以将查询的列，或者表指定需要的名字，如表名太长，用其简称，在连表查询中经常用到。

1) 将结果列改为需要的名称

SELECT ID AS StudentID,Name AS StudentName FROM Students

2）用表名的别名，标识列的来源

SELECT S.ID,S.Name,M.Name AS MajorName

FROM Students AS S

LEFT JOIN Majors AS M

ON S.MajorID = M.ID

3)在合计函数中，给合计结果命名

SELECT COUNT(ID) AS StudentCount FROM Students

### 10、Distinct

含义：不同的

作用：查询时忽略重复值。

**语法：**

SELECT DISTINCT 列名称 FROM 表名称

**实例：**

1）查询学生所在城市名，排除重复

SELECT DISTINCT City FROM Student

2)查询成绩分布分布情况

SELECT DISTINCT(Score),Count(ID) FROM Student GROUP BY Score

学生成绩可能重复，以此得到分数、得到这一成绩的学生数。后续会详细介绍GROUP BY 用法。

### 11、MAX/MIN

MAX 函数返回一列中的最大值。NULL 值不包括在计算中。

MIN 函数返回一列中的最小值。NULL 值不包括在计算中。

MIN 和 MAX 也可用于文本列，以获得按字母顺序排列的最高或最低值。

1）查询学生中最高的分数

SELECT MAX(Score) FROM Students

2)查询学生中最小年龄

SELECT MIN(Age) FROM Students

### 12、SUM

查询某列的合计值。

1）查询ID为1001的学生的各科总成绩

SC即为学生的成绩表，字段：StudentID，CourseID，Score.

SELECT SUM(Score) AS TotalScore FROM SC WHERE StudentID='1001'

### 13、AVG

AVG 函数返回数值列的平均值

1)查询学生的平均年龄

SELECT AVG(Age) AS AgeAverage FROM Students

2）求课程ID为C001的平均成绩

SELECT AVG(Score) FROM SC WHERE CourseID='C001'

### 14、COUNT

COUNT() 函数返回匹配指定条件的行数。

1)查询学生总数

SELECT COUNT(ID) FROM Students

2)查询学生年龄分布的总数

SELECT COUNT(DISTINCT Age) FROM Students

3)查询男生总数

SELECT COUNT(ID) FROM Students WHERE Sex='男'

4）查询男女生各有多少人

SELECT Sex,COUNT(ID) FROM Students GROUP BY Sex

### 15、GROUP BY

 GROUP BY 语句用于结合合计函数，根据一个或多个列对结果集进行分组。

1）查询男女生分布，上面已经给了答案。

SELECT Sex,COUNT(ID) FROM Students GROUP BY Sex

2) 查询学生的城市分布情况

SELECT City,COUNT(ID) FROM Students GROUP BY City

3)学生的平均成绩，查询结果包括：学生ID，平均成绩

SELECT StudentID,AVG(Score) FROM SC GROUP BY StudentID

 4）删除学生信息中重复记录

根据列进行分组，如果全部列相同才定义为重复，则就需要GROUP BY所有字段。否则可按指定字段进行处理。

DELETE FROM Students WHERE ID NOT IN (SELECT MAX(ID) FROM Students GROUP BY ID,Name,Age,Sex,City,MajorID)

### 16、HAVING

 在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与合计函数一起使用。

语法：

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value

GROUP BY column\_name

HAVING aggregate\_function(column\_name) operator value

1）查询平均成绩大等于于60的学生ID及平均成绩

SELECT StudentID,AVG(Score) FROM SC GROUP BY StudentID HAVING AVG(Score)>=**60**

2）还是用HAVING的SQL语句中，可以有普通的WHERE条件

查询平均成绩大于等于60，且学生ID等于1的学生的ID及平均成绩。

SELECT StudentID,AVG(Score) FROM SC

WHERE StudentID='1'

GROUP BY StudentID

HAVING AVG(Score)>=**60**

3）查询总成绩在600分以上（包括600）的学生ID

SELECT StudentID FROM SC GROUP BY StudentID HAVING SUM(Score)>=**600**

### 17、TOP

TOP 子句用于规定要返回的记录的数目。对于大数据很有用的，在分页时也会常常用到。

1）查询年龄最大的三名学生信息

SELECT TOP **3** ID,Name FROM Students ORDER BY Age DESC

2）还是上一道题，如果有相同年龄的如何处理呢？

SELECT ID,Name,Age FROM Students WHERE Age IN (SELECT TOP **3** Age FROM Students)

### 18、Case语句

计算条件列表，并返回多个可能的结果表达式之一。  
CASE 表达式有两种格式：

* CASE 简单表达式，它通过将表达式与一组简单的表达式进行比较来确定结果。
* CASE 搜索表达式，它通过计算一组布尔表达式来确定结果。

简单表达式语法：

CASE input\_expression

WHEN when\_expression THEN result\_expression [ ...n ]

[ ELSE else\_result\_expression ]

END

搜索式语法：

CASE

WHEN Boolean\_expression THEN result\_expression [ ...n ]

[ ELSE else\_result\_expression ]

END

1）查询学习信息，如果Sex为0则显示为男，如果为1显示为女，其他显示为其他。

SELECT ID, Name, CASE Sex WHEN '0' THEN '男' WHEN '1' THEN '女' ELSE '其他' END AS Sex

FROM Students

2）查询学生信息，根据年龄统计是否成年，大于等于18为成年，小于18为未成年

SELECT ID, Name, CASE WHEN Age>=**18** THEN '成年' ELSE '未成年'END AS 是否成年

FROM Students

3）统计成年未成年学生的个数

要求结果

|  |  |
| --- | --- |
| **成年** | **未成年** |
| 23 | 6 |

SQL语句

SELECT SUM(CASE WHEN Age>=**18** THEN **1** ELSE **0** END) AS '成年',SUM(CASE WHEN Age<**18** THEN **1** ELSE **0** END) AS '未成年'

FROM Students

 4）行列转换。统计男女生中未成年、成年的人数

结果如下：

|  |  |  |
| --- | --- | --- |
| **性别** | **未成年** | **成年** |
| 男 | 3 | 13 |
| 女 | 2 | 18 |

SQL语句:

SELECT CASE WHEN Sex=**0** THEN '男' ELSE '女' END AS '性别',

SUM(CASE WHEN Age<**18** THEN **1** ELSE **0** END) AS '未成年',

SUM(CASE WHEN Age>=**18** THEN **1** ELSE **0** END) AS '成年'

FROM Students

GROUP BY Sex

## 6. 数据库的多表查询：（连表查询）

连接查询包括合并、内连接、外连接和交叉连接，如果涉及多表查询，了解这些连接的特点很重要。

### 1. INNER JOIN（内连接）

INNER JOIN（内连接），也成为自然连接

作用：根据两个或多个表中的列之间的关系，从这些表中查询数据。

重点：内连接，只能查到匹配行。

语法：(INNER可省略)

SELECT fieldlist

FROM table1 [INNER] join table2

ON table1.column=table2.column

学生表信息（Students）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Name** | **Age** | **City** | **MajorID** |
| 101 | Tom | 20 | BeiJing | 10 |
| 102 | Lucy | 18 | ShangHai | 11 |

专业信息表（Majors）：

|  |  |
| --- | --- |
| **ID** | **Name** |
| 10 | English |
| 12 | Computer |

 实例：查询学生信息，包括ID,姓名、专业名称

SELECT Students.ID, Students.Name, Majors.Name AS MajorName

FROM Students INNER JOIN Majors

ON Students.MajorID = Majors.ID

 查询结果：

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **MajorName** |
| 101 | Tom | English |

根据结果可以清晰看到，确实只有匹配的行。学生Lucy的信息丢失了。

### 2.  外连接

与内连接相比，即使没有匹配行，也会返回一个表的全集。

外连接分为三种：左外连接，右外连接，全外连接。对应SQL：LEFT/RIGHT/FULL OUTER JOIN。通常我们省略outer 这个关键字。写成：LEFT/RIGHT/FULL JOIN。

重点：至少有一方保留全集，没有匹配行用NULL代替。

**1）LEFT OUTER JOIN，简称LEFT JOIN，左外连接（左连接）**

结果集保留左表的所有行，但只包含第二个表与第一表匹配的行。第二个表相应的空行被放入NULL值。

Eg：

SELECT Students.ID,Students.Name,Majors.Name AS MajorName

FROM Students LEFT JOIN Majors

ON Students.MajorID = Majors.ID

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **MajorName** |
| 101 | Tom | English |
| 102 | Lucy | NULL |

**2）RIGHT JOIN(right outer join)右外连接(右连接)**

右外连接保留了第二个表的所有行，但只包含第一个表与第二个表匹配的行。第一个表相应空行被入NULL值。右连接与左连接思想类似。只是第二张保留全集，如果第一张表中没有匹配项，用NULL代替

Eg：

SELECT Students.ID,Students.Name,Majors.Name AS MajorName

FROM Students RIGHT JOIN Majors

ON Students.MajorID = Majors.ID

查询结果：

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **MajorName** |
| 101 | Tom | English |
| NULL | NULL | Computer |

通过结果可以看到，包含了第二张表Majors的全集，Computer在Students表中没有匹配项，就用NULL代替。

**3）FULL JOIN (FULL OUTER JOIN，全外连接）**

全外连接，简称：全连接。会把两个表所有的行都显示在结果表中

Eg：

SELECT Students.ID,Students.Name,Majors.Name AS MajorName

FROM Students FULL JOIN Majors

ON Students.MajorID = Majors.ID

查询结果：

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **MajorName** |
| 101 | Tom | English |
| 102 | Lucy | NULL |
| NULL | NULL | Computer |

包含了两张表的所有记录，没有记录丢失，没有匹配的行用NULL代替。

### 3. CROSS JOIN（交叉连接）

假设集合A={a,b}，集合B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1), (b,2)}。

交叉连接返回左表中的所有行，左表中的每一行与右表中的所有行做笛卡尔积。

Eg：

SELECT Students.ID, Students.Name, Majors.Name AS MajorName

FROM Students CROSS JOIN Majors

查询结果：

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **MajorName** |
| 101 | Tom | English |
| 102 | Lucy | English |
| 101 | Tom | Computer |
| 102 | Lucy | Computer |

 查询结果与INNER JOIN一样，但是其效率就慢很多了。

### 4. 外键sql语句

**1、删除表中的外键：**

alter table tableName drop foreign key foreignKeyName;

（注意：foreignKeyName是类似于 xxx\_FK）

**2. 建立级联的外键约束：**

（注：级联指的是删除主表中的数据，从表中的对应数据自动删除）

create table A1(

id int not null primary key

)

create table B1(

id int not null primary key,

A1\_id int,

constraint A1\_id\_FK foreign key(A1\_id) references A1(id) on delete cascade

)

**3. mysql表创建好后添加外键**

alter table 需加外键的表 add constraint 外键名 foreign key(需加外键表的字段名) referencnes 关联表名(关联字段名);

eg：alter table B1 add constraint A1\_id\_FK foreign key(A1\_id) references A1(id) on delete cascade;

（注意：从表中不能有无效的数据，【主表中没有的数据】）

## 7.  [Apache—DBUtils工具介绍：](http://blog.csdn.net/fengdongkun/article/details/8236216)

### 1. 介绍：

是 Apache 组织提供的一个开源 JDBC工具类库，它是对JDBC的简单封装，学习成本极低，并且使用dbutils能极大简化jdbc编码的工作量，同时也不会影响程序的性能。因此dbutils成为很多不喜欢hibernate的公司的首选。

### 2. API介绍：

* + org.apache.commons.dbutils.QueryRunner
  + org.apache.commons.dbutils.ResultSetHandler
  + 工具类

org.apache.commons.dbutils.DbUtils

### 3. QueryRunne类：

该类简单化了SQL查询，它与ResultSetHandler组合在一起使用可以完成大部分的数据库操作，能够大大减少编码量

#### 3.1 构造方法

（1）默认的构造方法

（2）需要一个 javax.sql.DataSource 来作参数的构造方法

#### 3.2 主要方法：

**增删改：**

* public int update(Connection conn, String sql, Object[] params) throws SQLException:用来执行一个更新（插入、更新或删除）操作。
* public int update(Connection conn, String sql) throws SQLException：用来执行一个不需要置换参数的更新操作。

**查：**

* public Object query(Connection conn, String sql, Object[] params, ResultSetHandler rsh) throws SQLException：执行一个查询操作，在这个查询中，对象数组中的每个元素值被用来作为查询语句的置换参数。该方法会自行处理 PreparedStatement 和 ResultSet 的创建和关闭。
* public Object query(String sql, Object[] params, ResultSetHandler rsh) throws SQLException:　几乎与第一种方法一样；唯一的不同在于它不将数据库连接提供给方法，并且它是从提供给构造方法的数据源(DataSource) 或使用的setDataSource 方法中重新获得 Connection。
* public Object query(Connection conn, String sql, ResultSetHandler rsh) throws SQLException : 执行一个不需要置换参数的查询操作。

**批处理：**

* batch**(sql, params)**

### 4. ResultSetHandler接口

该接口用于处理 java.sql.ResultSet，将数据按要求转换为另一种形式

（略）

**防止sql注入：**

‘ or 1=1 or username=’ 这句话会导致sql注入，没有用户名密码也可以登录。

使用perpareStatement语句，预编译。

Sql注入的原理：Select \* from users where true 是可以返回值的。