# 第10章 Sql，聚合函数，基本ddl，数据库管理方式

# 数据操作语言DML

-- 创建表

CREATE TABLE student(

id INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(10),

age INT,

gender VARCHAR(1)

);

## 数据插入（添加数据）

有3种形式

形式1：

insert into 表名（字段名1，字段名2，....）values (值a1，值a2， .....)， (值b1，值b2， .....)，..... ；

形式2：

insert into 表名1（字段名1，字段名2，....）select 字段名1，字段名2，.... from 表名2；

形式3：

insert into 表名 set 字段名1=值1，字段名2=值2， ..... ；

-- 1、插入所有字段

INSERT INTO student(id,NAME,age,gender) VALUES(1,'zhangsan',20,'男');

-- Duplicate entry '1' for key 'PRIMARY'

INSERT INTO student(id,NAME,age,gender) VALUES(2,'lisi',20,'女');

INSERT INTO student VALUES(3,'wangwu',20,'女');

-- 插入部分字段值(必须把前面的字段名都写上)

INSERT INTO student(NAME,age,gender) VALUES('zhaoliu',28,'女');

## 删除数据

基本语法形式：

delete from 表名 [where条件] [order排序] [limit限定]；

-- 删除表里面所有数据（很少使用，非常危险）

-- delete删除表里面所有的数据，自增的id不会初始化为初始值，会继续往下递增.

DELETE FROM student;

DELETE FROM student WHERE id=2;

-- TRUNCATE删除表里面所有的数据，自增的id会重新初始化为初始值

TRUNCATE TABLE student;

truncate [table] 表名；用于直接删除整个表（结构）并重新创建该表。

1，删除整个表（数据和结构都没有了）

2，重新创建该表（全新表）。

3，跟delete 语句不带where条件，有什么区别？

主要影响的是类似：auto\_increment这种类型的字段值：

truncate结果会重新计算，delete还能继续增长。

## 修改数据

基本语法：

update 表名 set 字段名1=值表达式1，字段名2=值表达式2，....[where条件] [order排序] [limit限定]；

UPDATE student SET gender='女';

UPDATE student SET gender='男' WHERE id=2;

# 数据查询语言DQL

## 基本查询语法形式

select [all | distinct] 字段或表达式列表 [from子句] [where子句] [group by子句] [having子句] [order by子句] [limit子句]；

## select 查询模型

数据库中以表为组织单位存储数据。

表类似我们的Java类，每个字段都有对应的数据类型。

那么用我们熟悉的java程序来与关系型数据对比，就会发现以下对应关系。

类----------表

类中属性----------表中字段（列）

对象----------记录（行）

列就是变量（类中属性就是变量）

变量是可以计算（操作）

where是表达式，值为真假（true或false）

这六句话是整个select查询模型的核心。

## [all | distinct]

用于设定所select出来的数据是否允许出现重复行（完全相同的数据行）

all：允许出现——默认不写就是All（允许的）。

distinct：不允许出现——就是所谓的“消除重复行”。

-- 显示所有数据

SELECT \* FROM student;

-- 查询指定列

SELECT NAME,gender FROM student;

-- 查询的时候添加常量列

SELECT NAME,age,gender,'java1711' AS '班级' FROM student;

-- 查询时合并列

SELECT NAME,age,gender,(php+java) AS '总成绩' FROM student;

-- 查询时候去掉重复的记录

SELECT DISTINCT address FROM student;

## 条件查询 where

SELECT \* FROM student WHERE NAME='zhangsan';

-- 逻辑条件: and(同时成立) or(只要有一个成立)

SELECT \* FROM student WHERE NAME='zhangsan' AND address='上海';

SELECT \* FROM student WHERE NAME='zhangsan' OR address='上海';

## 比较条件: > < >= <= <> !=不等于

SELECT \* FROM student WHERE php>=50 AND php<=70;

-- between and (等价于 >= and <= )

SELECT \* FROM student WHERE php BETWEEN 50 AND 70;

-- 查询地址不是济南的学生信息

SELECT \* FROM student WHERE address!='济南';

## 判空条件（null空字符串）(联想基本数据类型的包装类)

-- null判断：is null, is not null

-- 空字符串: ='' <>'' !=''

-- null:表示没有值，初始化的时候就是null

-- 空字符串:已经赋值，有值是空

SELECT \* FROM student WHERE address='';

SELECT \* FROM student WHERE address IS NULL;

## 聚合查询

聚合查询函数：sum(),avg(),max(),min(),count()

-- 统计学生php的总成绩(sum求和)

SELECT SUM(php) AS 'php总成绩' FROM student;

-- 查询php学生平均分

SELECT AVG(php) AS 'php平均分' FROM student;

-- 查询php最高分max(php)

SELECT MAX(php) AS 'php最高分' FROM student;

-- 统计学生表里面一共有多少学生

SELECT COUNT(\*) AS '总人数' FROM student;

SELECT COUNT(id) AS '总分数' FROM student;

SELECT COUNT(address) AS '总人数' FROM student;

-- 注意：count()函数统计的是不包含null的数据个数

-- 使用count统计表中的记录数，要使用不包含null的字段

## 查询排序

-- 语法：order by 字段 asc/desc 默认就是asc升序

SELECT \* FROM student ORDER BY php ASC;

SELECT \* FROM student ORDER BY php;

SELECT \* FROM student ORDER BY php DESC;

-- 多个条件排序

-- 需求：先按照php升序，java逆序（整体按照php升序，如果php相同就要找java逆序排序）

SELECT \* FROM student ORDER BY php ASC, java DESC;

## group by 分组子句 having

形式：

group by 字段1 排序方式1，字段2 排序方式2， .....

通常都只进行一个字段的分组。

什么叫分组？就是将数据以某个字段的值为“依据”，分到不同的“组别”里。

分组的结果通常：

1，数据结果只能是“组”——没有数据本身的个体

2，数据结果就可能“丢失”很多特性，比如没有性别，身高，姓名，等等。

3，实际上，结果中通常只剩下“组”作为整体的信息：

首先是该组的本身依据值，

另外，这几个可能的值：组内成员的个数，组内某些字段的最大值，最小值，平均值，总和值。

其他字段，通常就不能用了。

4，如果是2个字段或以上分组，则其实是相当于对前一分组的组内，再进行后一依据的分组。

having子句其实概念跟where子句完全一样：

where是针对表的字段的值进行“条件判断”

having是只针对groupby之后的“组”数据进行条件判断

-- 需求：查询男女分别有多少人

SELECT gender AS '性别', COUNT(id) AS '总人数' FROM student GROUP BY gender;

-- select 后面的查询都是基于group by 分组之后

-- 分组查询后筛选

-- 需求：address大于2的

-- group by 之后的条件查询需要用 having

SELECT address AS '地址', COUNT(address) AS '总人数' FROM student GROUP BY address HAVING COUNT(address)>2;

## limit子句

形式：

limit [起始行号start]， 要取出的行数num

说明：

表示将前面取得的数据并前面排好之后（如果有），对之指定取得“局部连续的若干条”数据。

起始行号start：第一行的行号为０，可以省略，则为默认行号（０）。

要取得的行数：如果结果集中从指定的行号开始到最后没有这么多行，则就只取到最后。

此子句非常有用——主要用于网页上最常见的一个需求（现象）：分页。

分页原理：

分页的前提：人为指定每页显示的条数，pageSize = 3;

显示（取得）第1页数据：select \* from 表名 limit 0, pageSize;

显示（取得）第2页数据：select \* from 表名 limit 3, pageSize;

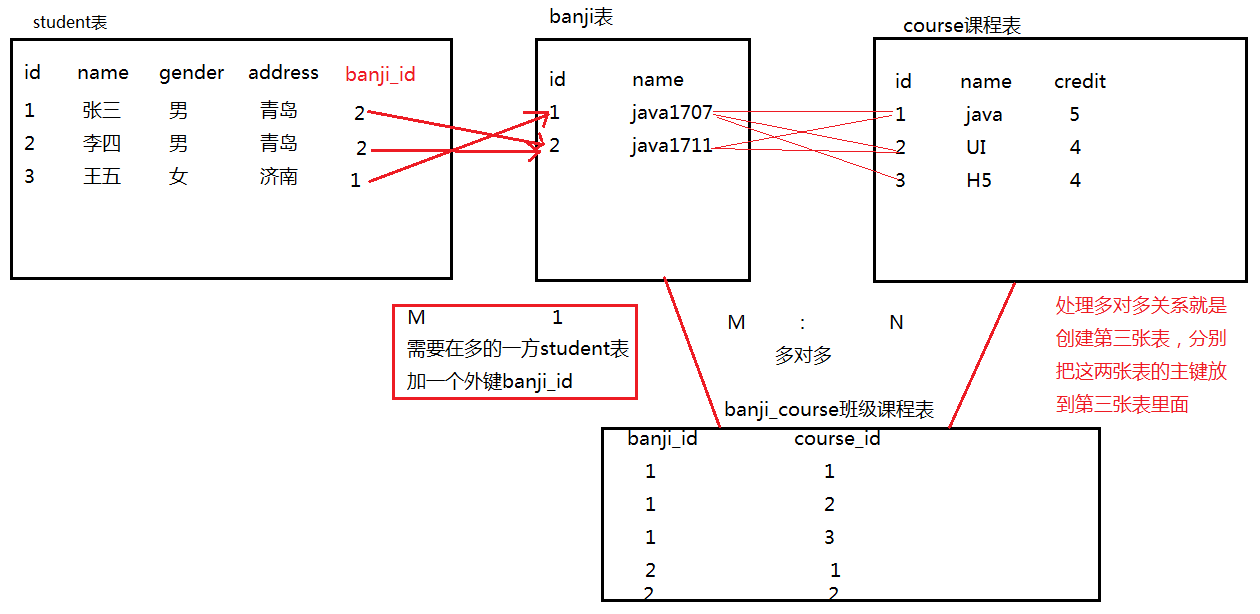
显示（取得）第3页数据：select \* from 表名 limit 6, pageSize;

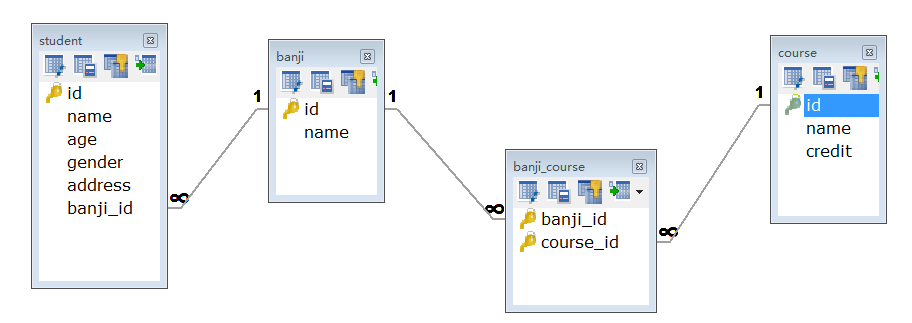
..................................................

显示（取得）第n页数据：select \* from 表名 limit (n-1)\*pageSize, pageSize;

# 多表查询：子查询、连接查询、联合查询

学生表、班级表、课程表、班级课程表





## 子查询

一个查询，通常就是一个select语句（即出现一次select关键字）

但，如果在一个select查询语句中，又出现了select查询语句，此时就称后者为“子查询”，前者就是“主查询”

形式：

select 字段或表达式或(子查询1) [as 别名] from 表名或(子查询2) where 字段或表达式或(子查询3) 的条件判断。

### 使用in子查询

in的基本语法形式为：

where 操作数 in (值1，值2， ....）

则in子查询就是：

where 操作数 in ( 列子查询 ）；

含义：

表示该操作数（字段值） 等于 该子查询的其中任意一个只，就算满足条件

### 使用any子查询

使用形式：

where 操作数 比较运算符 any ( 列子查询 );

说明：

1．操作数通常仍然是字段名

2．比较运算符就是常规的>、>＝、<、<=、=、<>

3．列子查询也可以是标量子查询，都表示“若干个数据值”

含义：

表示该操作数的值只要跟列子查询的任意一个值满足给定的比较运算，就算满足了条件——就是只要有一个成就成。

子查询：嵌套查询（内部语句）。一个查询语句是另外一个查询语句的条件。子查询的语句必须放在小括号之内

select \* from t1 where id=(select id from t2);

-- 班级是java707班的所有学生信息

SELECT \* FROM student WHERE banji\_id=(SELECT id FROM banji WHERE NAME='java1707');

-- SELECT \* FROM student WHERE banji\_id=1;

-- 总结：只能从一张表里面查询数据，另一张表提供条件

-- 班级是java1707班或者是java1711班的学生信息

SELECT \* FROM student WHERE banji\_id=1 OR banji\_id=2;

SELECT \* FROM student WHERE banji\_id IN(SELECT id FROM banji WHERE NAME='java1707' OR NAME='java1711');

-- "=":要求子查询只有一个结果 "in"子查询可能有多个结果

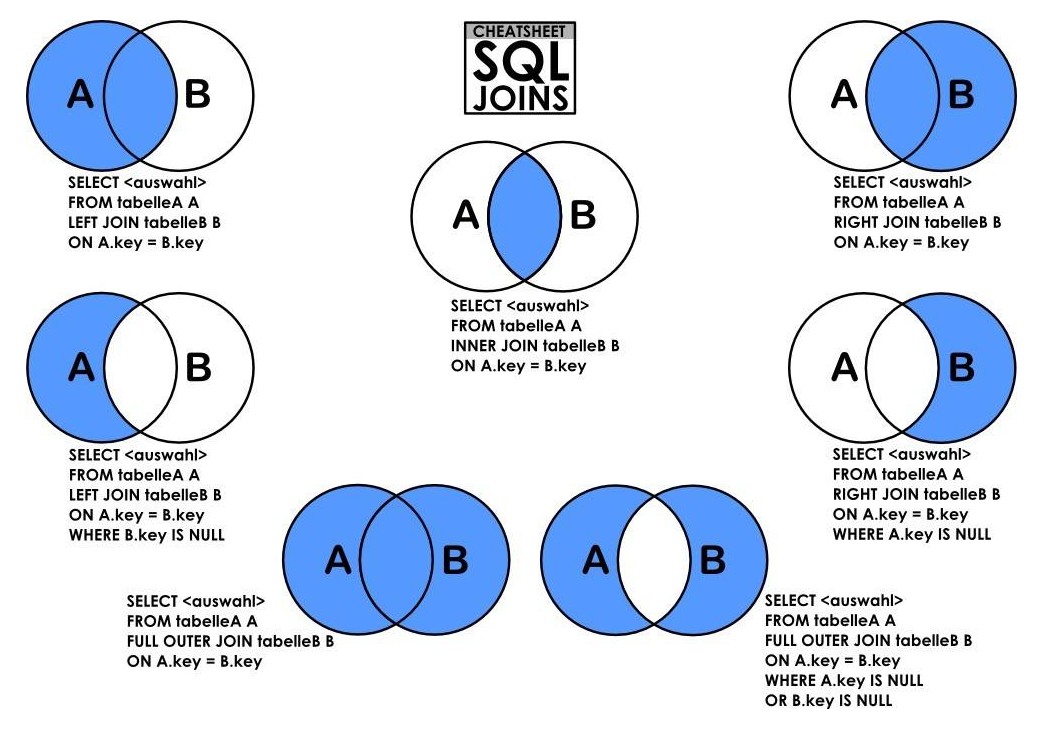
## 连接查询

连接就是指两个或2个以上的表（数据源）“连接起来成为一个数据源”。

实际上，两个表的完全的连接是这样的一个过程：

左边的表的每一行，跟右边的表的每一行，两两互相“横向对接”后所得到的所有数据行的结果。

注意：连接之后，并非形成了一个新的数据表，而只是一种“内存形态”



### 交叉连接

实际上，交叉连接是将两个表不设定任何条件的连接结果。

交叉连接通常也被叫做“笛卡尔积”——数学上可能比较多。

语法：

from 表1 [cross] join 表2 ; //可见交叉连接只是没有on条件而已。

cross这个词也可以省略，还可以使用inner这个词代替

### 内连接

语法：

from 表1 [inner] join 表2 on 表1.字段1=表2.字段2；

含义：找出（过滤）在交叉连接的结果表中的表1的字段1的值等于表2的字段2的值的那些行。

没有条件就是交叉连接cross join，也叫作笛卡尔积

### 左［外］连接

形式：

from 表1 left [outer] join 表2 on 连接条件。

说明：

1，这里，left是关键字。

2，连接条件跟内连接一样。

3，含义是：内连接的结果基础上，加上左边表中所有不符合连接条件的数据，相应本应放右边表的字段的位置就自动补为“null”值。

### 右［外］连接

右连接跟左连接恰恰相反：

形式：

from 表1 right [outer] join 表2 on 连接条件。

说明：

1，这里，right是关键字。

2，连接条件跟内连接一样。

3，含义是：在内连接的结果基础上，加上右边表中所有不符合连接条件的数据，相应本应放左边表的字段的位置就自动补为“null”值。

### 全［外］连接

形式：

from 表1 full [outer] join 表2 on 连接条件；

说明：

1，含义：其实是左右连接的“并集”（消除重复项），即内连接的结果，加上左表中不满足条件的所有行（右边对应补null），再加上，右表中不满足条件的所有行（左边对应补null）。

2，mysql中其实不认识全[外]连接语法，即mysql这个软件本身不支持全连接的语法。

3，此概念在其他数据库有的存在，了解就可以。

### 举例

-- 关联查询：inner join

-- 如果想把班级信息和学生信息都展示出来，就需要使用关联查询。

-- 需求：在上面展示学生信息的基础上将班级的名称也展示出来，就要用到连接查询。

SELECT s.name AS '姓名',age,gender,address,b.name AS '班级'

FROM student AS s INNER JOIN banji AS b ON s.banji\_id=b.id;

-- inner join on 只有左右两个表有关联才查询出来

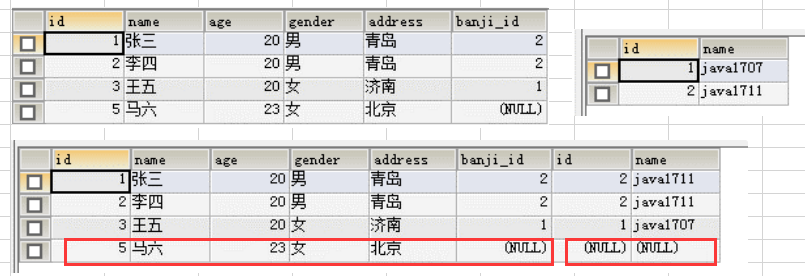
-- left join on 左表中都显示，右表没有显示空。

-- right join on 右表中都显示，左表没有显示空。

SELECT \*

FROM student AS s LEFT JOIN banji AS b

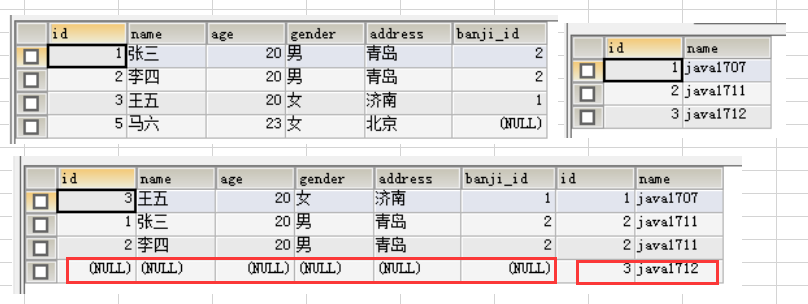
ON s.banji\_id=b.id;



SELECT \*

FROM student AS s RIGHT JOIN banji AS b

ON s.banji\_id=b.id;



-- 列出所有班级学习的课程名称

-- 学生名字 班级名字 课程名字 学分

-- 首先看要查询显示数据是来自单表还是多张表，如果是多表要用关联查询。

-- 用关联查询inner(left/right)join on 关联成一张更大的表，最终就是

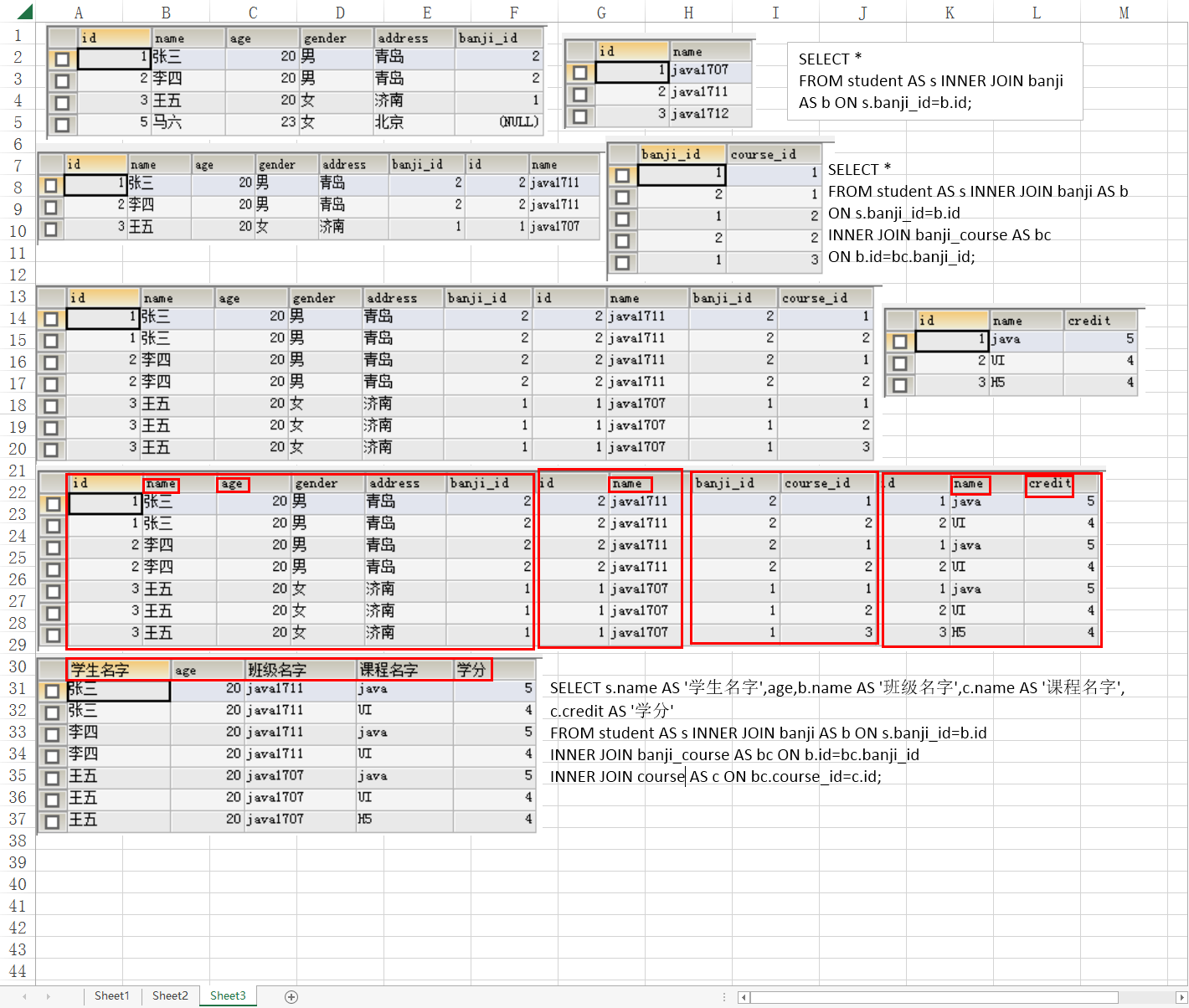
-- 在更大的表里面筛选你想显示的数据

SELECT s.name AS '学生名字',age,b.name AS '班级名字',c.name AS '课程名字',c.credit AS '学分'

FROM student AS s INNER JOIN banji AS b ON s.banji\_id=b.id

INNER JOIN banji\_course AS bc ON b.id=bc.banji\_id

INNER JOIN course AS c ON bc.course\_id=c.id;



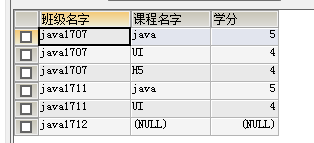
-- 列出所有的班级学习的所有课程名称，即使这个班级没有分配课程也要显示。

-- 班级名称 课程名称 学分

SELECT b.name AS '班级名字',c.name AS '课程名字',c.credit AS '学分'

FROM banji AS b LEFT JOIN banji\_course AS bc ON b.id=bc.banji\_id

LEFT JOIN course AS c ON bc.course\_id=c.id;

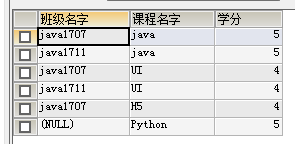


-- 列出所有的课程被哪些班级学习，即使这个课程没有任何班级学习。

SELECT b.name AS '班级名字',c.name AS '课程名字',c.credit AS '学分'

FROM banji AS b RIGHT JOIN banji\_course AS bc ON b.id=bc.banji\_id

RIGHT JOIN course AS c ON bc.course\_id=c.id;



总结：多表查询非常简单，只要把握住两点：

1、整个查询涉及到几张表，涉及到几张表就链接几张表。

2、如果涉及到几张表中的关系搞不清楚，画一下ER图，也就是表与表之间关系的图。

## 联合查询

联合查询就是将两个select语句的查询结果“层叠”到一起成为一个“大结果”。

两个查询结果的能够进行“联合”的先觉条件是：结果字段数相等。

UNION 用于合并两个或多个 SELECT 语句的结果集，并消去表中任何重复行。

UNION 内部的 SELECT 语句必须拥有相同数量的列，列也必须拥有相似的数据类型。

同时，每条 SELECT 语句中的列的顺序必须相同.

SQL UNION 语法：

复制代码 代码如下:

SELECT column\_name FROM table1

UNION

SELECT column\_name FROM table2

注释：默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。

当 ALL 随 UNION 一起使用时（即 UNION ALL），不消除重复行

SQL UNION ALL 语法

复制代码 代码如下:

SELECT column\_name FROM table1

UNION ALL

SELECT column\_name FROM table2

注释：另外，UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

注意：1、UNION 结果集中的列名总是等于第一个 SELECT 语句中的列名

2、UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同

# 思途心得

数据库的查询语句是数据库开发中的非常重要的知识，因为我们很多时候不会频繁的创建或者修改表，更多的操作是将数据以不同的形式展示给用户。

要重点掌握select查询模型：列是变量，变量可以计算。where是表达式，值为真假。

对于分组操作group by，要注意select后的查询数据是分组之后在每一个分组里面查询的，而不是在没有分组的所有数据里面查询的。

# 作业

首先在数据库中创建如下表：

create table goods (

goods\_id mediumint(8) unsigned primary key auto\_increment,

goods\_name varchar(120) not null default '',

cat\_id smallint(5) unsigned not null default '0',

brand\_id smallint(5) unsigned not null default '0',

goods\_sn char(15) not null default '',

goods\_number smallint(5) unsigned not null default '0',

shop\_price decimal(10,2) unsigned not null default '0.00',

market\_price decimal(10,2) unsigned not null default '0.00',

click\_count int(10) unsigned not null default '0'

) engine=myisam default charset=utf8;

insert into `goods` values (1,'kd876',4,8,'ecs000000',1,1388.00,1665.60,9),

(4,'诺基亚n85原装充电器',8,1,'ecs000004',17,58.00,69.60,0),

(3,'诺基亚原装5800耳机',8,1,'ecs000002',24,68.00,81.60,3),

(5,'索爱原装m2卡读卡器',11,7,'ecs000005',8,20.00,24.00,3),

(6,'胜创kingmax内存卡',11,0,'ecs000006',15,42.00,50.40,0),

(7,'诺基亚n85原装立体声耳机hs-82',8,1,'ecs000007',20,100.00,120.00,0),

(8,'飞利浦9@9v',3,4,'ecs000008',1,399.00,478.79,10),

(9,'诺基亚e66',3,1,'ecs000009',4,2298.00,2757.60,20),

(10,'索爱c702c',3,7,'ecs000010',7,1328.00,1593.60,11),

(11,'索爱c702c',3,7,'ecs000011',1,1300.00,0.00,0),

(12,'摩托罗拉a810',3,2,'ecs000012',8,983.00,1179.60,13),

(13,'诺基亚5320 xpressmusic',3,1,'ecs000013',8,1311.00,1573.20,13),

(14,'诺基亚5800xm',4,1,'ecs000014',1,2625.00,3150.00,6),

(15,'摩托罗拉a810',3,2,'ecs000015',3,788.00,945.60,8),

(16,'恒基伟业g101',2,11,'ecs000016',0,823.33,988.00,3),

(17,'夏新n7',3,5,'ecs000017',1,2300.00,2760.00,2),

(18,'夏新t5',4,5,'ecs000018',1,2878.00,3453.60,0),

(19,'三星sgh-f258',3,6,'ecs000019',12,858.00,1029.60,7),

(20,'三星bc01',3,6,'ecs000020',12,280.00,336.00,14),

(21,'金立 a30',3,10,'ecs000021',40,2000.00,2400.00,4),

(22,'多普达touch hd',3,3,'ecs000022',1,5999.00,7198.80,16),

(23,'诺基亚n96',5,1,'ecs000023',8,3700.00,4440.00,17),

(24,'p806',3,9,'ecs000024',100,2000.00,2400.00,35),

(25,'小灵通/固话50元充值卡',13,0,'ecs000025',2,48.00,57.59,0),

(26,'小灵通/固话20元充值卡',13,0,'ecs000026',2,19.00,22.80,0),

(27,'联通100元充值卡',15,0,'ecs000027',2,95.00,100.00,0),

(28,'联通50元充值卡',15,0,'ecs000028',0,45.00,50.00,0),

(29,'移动100元充值卡',14,0,'ecs000029',0,90.00,0.00,0),

(30,'移动20元充值卡',14,0,'ecs000030',9,18.00,21.00,1),

(31,'摩托罗拉e8 ',3,2,'ecs000031',1,1337.00,1604.39,5),

(32,'诺基亚n85',3,1,'ecs000032',4,3010.00,3612.00,9);

create table category (

cat\_id smallint unsigned auto\_increment primary key,

cat\_name varchar(90) not null default '',

parent\_id smallint unsigned

)engine myisam charset utf8;

INSERT INTO `category` VALUES

(1,'手机类型',0),

(2,'CDMA手机',1),

(3,'GSM手机',1),

(4,'3G手机',1),

(5,'双模手机',1),

(6,'手机配件',0),

(7,'充电器',6),

(8,'耳机',6),

(9,'电池',6),

(11,'读卡器和内存卡',6),

(12,'充值卡',0),

(13,'小灵通/固话充值卡',12),

(14,'移动手机充值卡',12),

(15,'联通手机充值卡',12);

1. 取出第4栏目或第11栏目的商品。
2. 取出100<=价格<=500的商品。
3. 取出不属于第3栏目且不属于第11栏目的商品(and,或not in分别实现)。
4. 取出价格大于100且小于300,或者大于4000且小于5000的商品()。
5. 取出第1个栏目下面的商品(注意:1栏目下面没商品,但其子栏目下有)。
6. 取出名字以"诺基亚"开头的商品。
7. 取出名字不以"诺基亚"开头的商品。
8. 取出第3个栏目下面价格在1000到3000之间,并且点击量>5 "诺基亚"开头的系列商品。
9. 查询该店的商品比市场价所节省的价格。
10. 查询比市场价省钱200元以上的商品及该商品所省的钱(where和having分别实现)。
11. 查询积压货款超过2W元的栏目,以及该栏目积压的货款。
12. 查询出编号为19的商品的栏目名称(用左连接查询和子查询分别)。

用exists型子查询,查出所有有商品的栏目（有些栏目下面没有商品）。