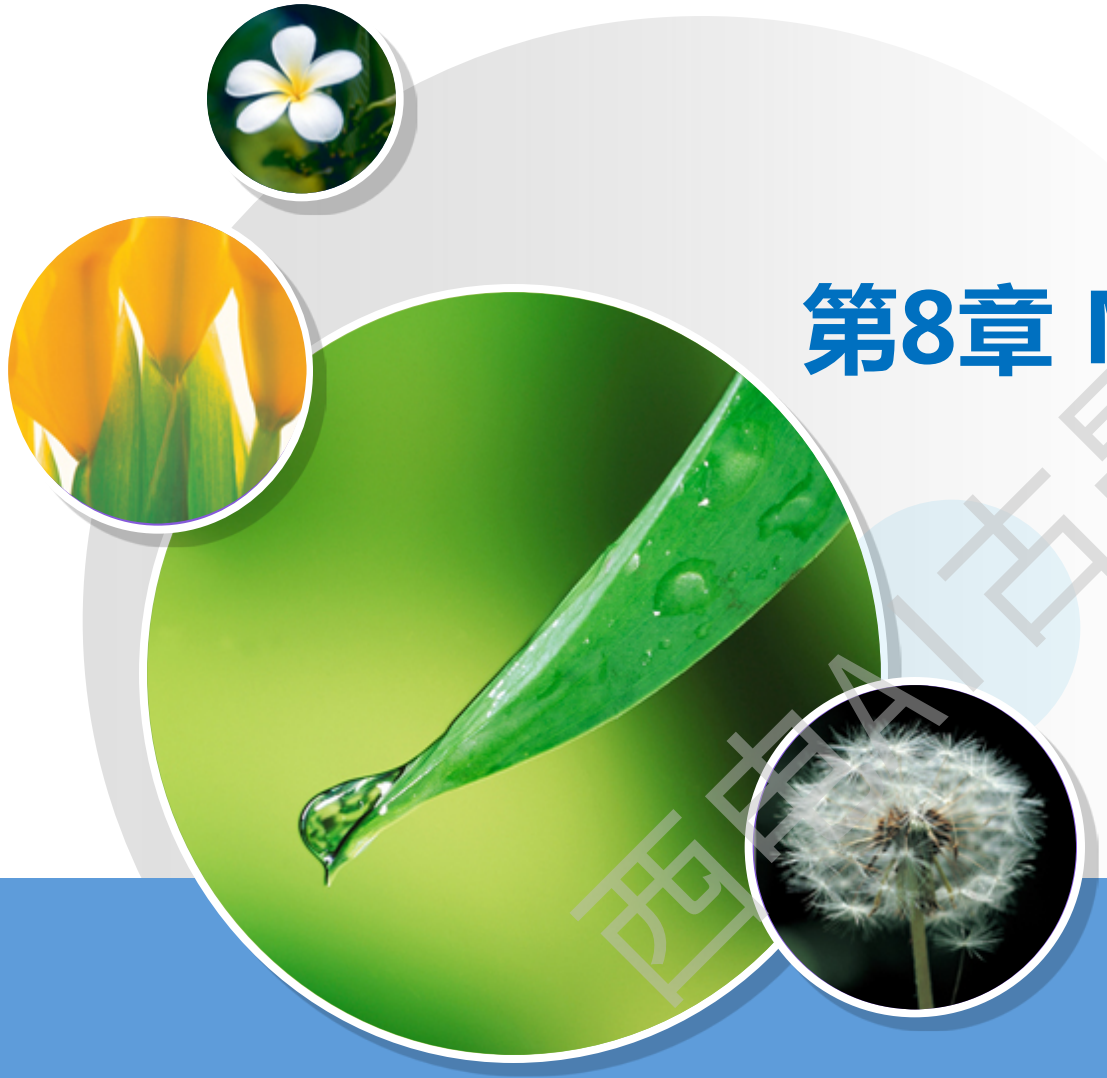


第8章 MySQL数据操作管理

人工智能学院
古　晶

西安电子科技大学

第8章 MySQL数据操作管理

- 8.1 插入数据
- 8.2 修改数据
- 8.3 删除数据
- 8.4 查询（单表查询，多表查询）
- 8.5 知识点小结
- 本章实验

第8章 MySQL数据操作管理

- MySQL**对数据的操作**主要有：
  - **添加**：向数据库表中添加不存在的记录
  - **删除**：删除数据库中已存在的记录
  - **修改**：对已经存在的记录进行更新
  - **查询**：指数据库管理系统按照数据用户指定的条件，从数据库中相关表中找到满足条件的记录过程。
- 本章将主要介绍如何查询MySQL数据库中的数据，还将介绍如何向数据库中添加记录，以及删除和修改记录。

第8章 MySQL数据操作管理

- 8.1 插入数据
- 8.2 修改数据
- 8.3 删除数据
- 8.4 查询（单表查询，多表查询）
- 8.5 知识点小结
- 本章实验

8.1 插入数据

- 插入数据是向表中插入新的记录。通过这种方式可以为表中增加新的数据。
- 具体插入数据的**形式**有
  - ✓ **不指定具体的字段名**
  - ✓ **列出指定字段**
  - ✓ **一次插入多条记录**
  - ✓ **通过SET形式插入记录**
  - ✓ **从目标表中插入值，即将查询结果插入到表中**
  - ✓ **REPLACE语句**

创建vipuser表，用于后续插入演示示例

```
CREATE TABLE IF NOT EXISTS vipuser(
    id TINYINT UNSIGNED AUTO_INCREMENT KEY,
    username VARCHAR(20) NOT NULL UNIQUE,
    password CHAR(32) NOT NULL,
    email VARCHAR(50) NOT NULL DEFAULT 'lihui@cau.edu.cn',
    age TINYINT UNSIGNED DEFAULT 18
    );
```

8.1 插入数据--不指定具体的字段名

口语法格式：INSERT [INTO] tbl_name VALUES (值...)

口示例：

✓ **插入记录：**

- INSERT INTO vipuser VALUES(1,'CAUE','123', 'CAUE@QQ. COM' ,20);
- INSERT INTO vipuser VALUES(2,'CAUW','456', 'CAUW@QQ.COM ' ,30);

✓ **查看插入记录：**

- SELECT * FROM vipuser;

8.1 插入数据--指定具体的字段名

口语法格式：INSERT [INTO] tbl_name(字段名称m,...) VALUES(值m,...);

- ✓ “字段名m”参数表示表中的字段名称，此处指定表的部分字段的名称；“值m”参数表示指定字段的值，每个值与相应的字段对应。
- ✓ 没有赋值的字段，数据库系统会为其插入默认值。这个默认值是在创建表的时候定义的。
- ✓ 可以随意的设置字段的顺序，而不需要按照表定义时的顺序。

口示例：

- ✓ **列出指定字段,插入记录：**
- INSERT vipuser(username,password) VALUES('A','AAA');
- INSERT vipuser(password,username) VALUES('BBB','B');
- ✓ **查看插入记录**
- SELECT * FROM vipuser;

8.1 插入数据--一次插入多条记录

口语法格式：INSERT [INTO] tbl_name[(字段名称...)] VALUES(值...),(值...)...

- ✓ “**tbl_name**”参数指明向哪个表中插入数据；“字段名列表”参数是可选参数，指定哪些字段插入数据，没有指定字段时向所有字段插入数据；“取值列表n”参数表示要插入的记录，每条记录之间用逗号隔开。
- ✓ 如果插入的记录很多时，一个INSERT语句插入多条记录的方式的速度会比较快。

口示例：

- ✓ **一次插入多条记录：**

```
INSERT vipuser VALUES(6,'D','DDD','D@QQ.COM',35),
    (8,'E','EEE','E@QQ.COM',9),
    (18,'F','FFF','F@QQ.COM',32);
```

8.1 插入数据--通过SET形式插入记录

口语法格式：**INSERT [INTO] tbl_name SET 字段名称=值,...**

口示例：

✓ **通过INSERT SET形式插入记录：**

INSERT INTO vipuser SET id=98,username='test',password='abc',email='123@qq.com',age=48;

✓ **只给username插入记录：**

INSERT vipuser SET username='贾玲',password='lingling' ;

8.1 插入数据--从目标表中插入值

口语法格式：**INSERT [INTO] tbl_name[(字段名称,...)] SELECT 字段名称 FROM tbl_name [WHERE 条件]**

注：SELECT语句中返回的是一个查询到的结果集，INSERT语句将这个结果插入到目标表中，结果集中记录的字段数和字段类型要与目标表完全一致。

口示例：创建一个testuser表

```
CREATE TABLE IF NOT EXISTS t_testUser(
    id TINYINT UNSIGNED AUTO_INCREMENT KEY,
    username VARCHAR(20) NOT NULL UNIQUE
    );
```

✓ **将查询结果插入表中：**

```
INSERT t_testUser SELECT id,username FROM vipuser;
```

- 字段数目不配（保证字段一致数据类型要符合，如果是“*”就会报错（错误的写法））

```
INSERT t_testUser SELECT * FROM vipuser;
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

8.1 为表的指定字段插入数据

- **语法格式**：**REPLACE [INTO] 表名 VALUES（值列表）**
- **注**：
  - ✓ 如果新行不存在，MySQL的REPLACE语句插入一个新行；
  - ✓ 如果新行已经存在，则REPLACE语句首先删除旧的行，然后插入一个新行。在某些情况下，REPLACE语句只更新现有行；
  - ✓ 使用REPLACE语句添加记录时，如果新记录的主键值或者唯一性约束的字段值与已经有记录相同，则已有记录被删除后再添加新记录。
- **示例**：插入贾玲的信息

```
REPLACE vipuser VALUES(99,'贾玲','123',
'jialing@qq.com',20);
```

第8章 MySQL数据操作管理

- 8.1 插入数据
- **8.2 修改数据**
- 8.3 删除数据
- 8.4 查询（单表查询，多表查询）
- 8.5 知识点小结
- 本章实验

8.2 修改数据

- 修改数据是更新表中已经存在的记录。通过**UPDATE语句**来修改数据。
- UPDATE语句的**基本语法形式**如下：
  - **语法格式**：

    UPDATE 表名
    SET 字段名1 = 取值1，字段名2 = 取值2，… 字段名n = 取值n
    WHERE 条件表达式
  - 其中,"字段名n"参数表示需要更新的字段的名称；"取值n"参数表示为字段更新的新数据；"条件表达式"参数指定更新满足条件的记录。
- 注意：**如果不加where条件就会更新全部记录，多个记录用逗号分隔**

8.2 修改数据

- UPDATE语句的基本语法形式如下：
  - **语法格式**：

    UPDATE 表名

    SET 字段名1 = 取值1，字段名2 = 取值2，… 字段名n = 取值n

    WHERE 条件表达式
    - 其中，“字段名n”参数表示需要更新的字段的名称；“取值n”参数表示为字段更新的新数据；“条件表达式”参数指定更新满足条件的记录。
- **注意**：**如果不加where条件就会更新全部记录，多个记录用逗号分隔**
- **示例**：
- ✓ UPDATE vipuser SET password='caue123',email='123@qq.com',age=99 WHERE id=1;
- ✓ 记录“E”恢复到默认值：UPDATE vipuser SET age=DEFAULT WHERE username='E';

第8章 MySQL数据操作管理

- 8.1 插入数据
- 8.2 修改数据
- **8.3 删除数据**
- 8.4 查询（单表查询，多表查询）
- 8.5 知识点小结
- 本章实验

8.3 删除数据

- 删除数据是删除表中已经存在的**记录**。
  - 通过DELETE语句来删除数据。
    - **基本语法**：**DELETE FROM 表名 [WHERE 条件表达式]**
    - 其中，“表名”参数指明从哪个表中删除数据；“WIHERE条件表达式”指定删除表中的哪些数据。如果没有该条件表达式，数据库系统就会删除表中的所有数据。
- **示例**：**删除**t_testUser**表中id为1 的用户**

```
DELETE FROM t_testUser WHERE id=1;
```

- 使用TRUNCATE **完全清除**某一个表。
  - **基本语法**： **TRUNCATE [TABLE] 表名**
- **示例**：**清空t_testuser表**

```
TRUNCATE t_testiser;
```

8.3 删除数据

- **TRUNCATE TABLE 与DELETE的区别**：
  - TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同：二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE **速度快**，且使用的**系统和事务日志资源少**。
  - TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。
  - **TRUNCATE，DELETE，DROP比较**：
    - TRUNCATE TABLE：删除内容、释放空间但**不删除定义**。
    - DELETE TABLE：删除内容**不删除定义**，**不释放空间**。
    - DROP TABLE：删除内容和定义，释放空间。

第8章 MySQL数据操作管理

- 8.1 插入数据
- 8.2 修改数据
- 8.3 删除数据
- **8.4 查询（单表查询，多表查询）**
- 8.5 知识点小结
- 本章实验

单表查询

SELECT语法：

- SELECT语句是在所有数据库操作中**使用频率最高的SQL语句。**
- SELECT语句的语法格式如下:

```
SELECT select_expr [, select_expr ...]
    [
        FROM table_references
        [WHERE 条件]
        [GROUP BY {col_name | position}  [ASC | DESC], ...
分组]
        [HAVING 条件 对分组结果进行二次筛选]
        [ORDER BY {col_name | position} [ASC | DESC], ...
排序]
        [LIMIT 限制显示条数]
    ]
```

- 其中[]内的内容是可选的

单表查询

- **SELECT子句**：
  - 指定要查询的列名称，列与列之间用逗号隔开。
  - 还可以为列指定新的**别名**，显示在输出的结果中。
  - **ALL关键字**表示显示所有的行，包括**重复行**，是**系统默认的。**
  - **DISTINCT**表示显示的结果要**消除重复的行**。
- **FROM子句**：指定要查询的表，可以指定两个以上的表，表与表之间用逗号隔开。
- **WHERE子句**：指定要查询的条件。
  - 如果有WHERE子句，就按照“条件表达式”指定的条件进行查询；
  - 如果没有WHERE子句，就查询所有记录。

单表查询

- **GROUP BY** ：子句用于对查询结构进行分组。
  - 按照“列名1”指定的字段进行分组；
  - 如果GROUP BY子句后带着**HAVING关键字**，那么只有满足“条件表达式2”中指定的条件的才能够输出。
  - GROUP BY子句通常和COUNT()、SUM()等**聚合函数**一起使用。
- **HAVING子句**：指定分组的条件，通常放Group by字句之后
- **ORDER BY子句**：用于对查询结果的进行**排序**。
- 排序方式由ASC和DESC两个参数指出；
- ASC参数表示**按升序进行排序。默认情况下是ASC**。
- DESC参数表示按降序的顺序进行排序。升序表示值按从小到大的顺序排列。
- **LIMIT 子句**：限制查询的输出结果的**行数**。

简单查询

- **示例：**演示查询，创建几张表：
  - 专业表（specialty）

	专业号	专业名
列名	zno	zname
数据类型	varchar	varchar
长度	4	50
是否为空	not null	not null
是否主键	主键	
是否外键		

简单查询

- 课程表（course）

	课程号	课程名称	学分	开课院系
列名	cno	cname	ccredit	cdept
数据类型	varchar	varchar	int	varchar
长度	8	50	11	20
是否为空	not null	not null	not null	not null
是否主键	主键			
是否外键				

简单查询

- 学生表（student）

	学号	姓名	性别	出生日期	班级	专业号
列名	sno	sname	ssex	sbirth	sclass	zno
数据类型	varchar	varchar	enum	date	varchar	varchar
长度	10	20	{男，女}		10	4
是否为空	not null	not null	not null	not null	not null	null
是否主键	主键					
是否外键						外键

简单查询

- 选修表（sc）

	学号	课程号	成绩
列名	sno	cno	grade
数据类型	varchar	varchar	float
长度	10	8	4
是否为空	not null	not null	not null
是否主键	是	是	
是否外键	是	是	

简单查询

1. **查询所有字段**：查询表中的所有字段的数据
   - **两种方式**：
     - 一种是列出表中的所有字段
     - 另一种是使用**通配符***来查询
   - **示例**：查询学生的所有信息
     - **方式1**：使用 **SELECT zno,sclass ,sno,sname, ssex,sbirth FROM student**;（返回的结果字段的顺序和SELECT语句中指定的顺序一致。）
     - **方式2**：使用**SELECT * FROM student**;（返回的结果字段的顺序是固定的，和建立表时指定的顺序一致。）

简单查询

方法1：

查询创建工具 查询编辑器

```
1  select  zno,sclass ,sno,sname,ssex,sbirth from student;
```

信息 结果1 概况 状态

zno	sclass	sno	sname	ssex	sbirth
1407	工商1401	1114070116	欧阳贝贝	女	1997-01-08
1407	工商1401	1114070236	欧阳贝贝	女	1997-01-08
1214	信管1201	1207040137	郑熙婷	女	1996-05-23
1214	信管1201	1207040237	郑熙婷	女	1996-05-23
1102	商务1401	1411855228	唐晓	女	1997-11-05
1102	商务1401	1411855321	蓝梅	女	1997-07-02

简单查询

方法2：

查询创建工具 | 查询编辑器

```
1  select  * from student;
```

信息 | 结果1 | 概况 | 状态

sno	sname	ssex	sbirth	zno	sclass
1114070116	欧阳贝贝	女	1997-01-08	1407	工商1401
1114070236	欧阳贝贝	女	1997-01-08	1407	工商1401
1207040137	郑熙婷	女	1996-05-23	1214	信管1201
1207040237	郑熙婷	女	1996-05-23	1214	信管1201
1411855228	唐晓	女	1997-11-05	1102	商务1401
1411855321	蓝梅	女	1997-07-02	1102	商务1401

简单查询

2. 指定字段查询

- **示例**：查询学生的学号和姓名。**SELECT sno, sname FROM student;**

查询创建工具 查询编辑器

```
1  select sno,sname from student;
```

信息 结果1 概况 状态

sno	sname
1114070116	欧阳贝贝
1114070236	欧阳贝贝
1207040137	郑熙婷
1207040237	郑熙婷
1411855228	唐晓
1411855321	蓝梅
1411855426	余小梅

简单查询

3. 避免重复数据查询

- **DISTINCT关键字**可以去除重复的查询记录。和DISTINCT相对的是all关键字，即显示所有的记录（包括重复的），而**all关键字**是系统默认的，可以省略不写。
- **示例**：
  - ✓ **查询在student表中的班级：**
    - SELECT sclass FROM student;
    - SELECT DISTINCT sclass FROM student;

简单查询

3. 避免重复数据查询

查询创建工具 查询编辑器

```
1  Select sclass from student;
```

信息 结果1 概况 状态

sclass
工商1401
工商1401
信管1201
信管1201
商务1401
商务1401
商务1401
信管1401
食品1401

查询创建工具 查询编辑器

```
1  Select DISTINCT sclass from student;
```

信息 结果1 概况 状态

sclass
工商1401
信管1201
商务1401
信管1401
食品1401
会计1401
计算1401

4. 为表和字段取别名

- 当查询数据时，MySQL会显示每个输出列的名称。默认的情况下，显示的列名是创建表时定义的列名。

SELECT [ALL | DISTINCT] <目标列表达式> [AS] [别名] [, <目标列表达式> [AS] [别名]] . . .
FROM <表名或视图名> [别名] [, <表名或视图名>[别名]] . . .

- AS可以省略，但是不建议省略。
- **示例**：查询学生的学号，成绩，并指定返回的结果中的列名为学号,成绩，而不是sno和grade。

```
SELECT sno '学号',grade '成绩' FROM sc;
```

简单查询

4. 为表和字段取别名

查询创建工具 查询编辑器

```
1  select sno '学号',grade '成绩' from sc;
```

信息 结果1 概况 状态

学号	成绩
1414855328	85
1414855406	75
1412855223	60
1114070116	65
1414855302	90
1411855228	96
1418855232	87
1414855328	96
1414855406	86
1412855223	77
1414855406	84
1114070116	90
1411855321	69
1418855232	91

简单查询

- 在使用SELECT语句对列进行查询时，**在结果集中可以输出对列值计算后的值。**
- **例**：查询sc表中学生的成绩提高10%，对显示后的成绩列，显示为“修改后成绩”。
- SELECT sno, grade,grade*1.1 AS '修改后成绩' FROM sc;

查询创建工具 查询编辑器

```
1  SELECT sno, grade,grade*1.1 as  '修改后成绩'
2  FROM sc;
```

信息 结果1 概况 状态

sno	grade	修改后成绩
1414855328	85	93.5
1414855406	75	82.5
1412855223	60	66
1114070116	65	71.5
1414855302	90	99
1411855228	96	105.6
1418855232	87	95.7
1414855328	96	105.6
1414855406	86	94.6
1412855223	77	84.7
1414855406	84	92.4
1114070116	90	99

条件查询

- 条件查询主要使用关键字**WHERE**指定查询的条件
- WHERE子句常用的查询条件如下：

查询条件	符号或关键字
比较	=、<、<=、>、>=、!=、<>、!>、!<
匹配字符	LIKE、NOT LIKE
指定范围	BETWEEN AND、NOT BETWEEN AND
是否为空值	IS NULL、IS NOT NULL

- **“<>”表示不等于**，其作用等价于“!=”；**“!>”表示不大于**，等价于“<=”；**“!<”表示不小于**，等价于“>=”；**BETWEEN AND**指定了某字段的取值范围；**“IN”**指定了某字段的取值的集合；**IS NULL**用来判断某字段的取值是否为空；**AND**和**OR**用来**连接多个查询条件。**

条件查询

1.带关系运算符和逻辑运算符的查询

- MySQL中，可以通过关系运算符和逻辑运算符来编写“条件表达式”。
- MySQL支持的比较运算符有**>，<，=！，=(<>)，>=，<=；逻辑运算符有AND(&&)， OR (||)，XOR，NOT(!)**
- **示例**：查询成绩在70分到80分之间（包含70分和80分）学生的学号和成绩。

```
SELECT sno,grade FROM sc WHERE grade>=70 and grade<=80;
```

2.带IN关键字的查询

- IN关键字可以判断某个字段的值是否在指定的**集合**中，如果字段的值在集合中，则满足查询条件，该记录将被查询出来；如果不在集合中，则不满足查询条件。
- **语法格式 [NOT] IN (元素1,元素2, 元素3, . . .)**
- 其中，NOT是可选参数，加上NOT表示不在集合内满足条件：字符型元素要加上单引号。

条件查询

示例：

查询成绩在集合(65，75，85，95)中的学生的学号和成绩

SELECT sno,grade FROM sc WHERE grade In (65，75，85，95);

查询创建工具 查询编辑器

```
1  Select sno,grade from sc where grade In (65,75,85,95);
```

信息 结果1 概况 状态

sno	grade
1414855328	85
1414855406	75
1114070116	65

3.带BETWEEN AND关键字的查询

- BETWEEN AND关键字可以判断读某个字段的值是否在指定的范围内，如果在，则满足条件，否则不满足。
- **语法规则**：**[NOT] BETWEEN 取值1 AND 取值2**
- 其中，“NOT”是可选参数，加上NOT表示不在指定范内满足条件；“取值 1”表示范围的起始值；“取值2”表示范围的终止值。

条件查询

示例：

查询成绩在70分到80分之间（包含70分和80分）学生的学号和成绩。

SELECT sno,grade FROM sc WHERE grade BETWEEN 70 AND 80;

4.带IS NULL关键字的空值查询

- IS NULL关键字可以用来判断字段的值是否为空值（NULL）.如果字段值为空值，则满足查询条件，否则不满足。

  语法规则：**IS [not] NULL**

- IS NULL 是一个**整体**，**不能将IS换成“=”**。
- IS NOT NULL 中的**IS NOT也不可以换成!= 或者<>** 。

条件查询

示例：

查询还没有分专业的学生的学号和姓名。

查询条件：没分专业说明专业号为空，即 zno IS NULL;

SELECT sno,sname,zno FROM student WHERE zno IS NULL;

5.带LIKE关键字的查询

- LIKE关键字可以匹配字符串是否相等。如果字段的值与指定的字符串相匹配，则满足条件，否则不满足。

  语法规则: **[NOT] LIKE “字符串”；**

- 通配符“%”与“_”的**区别**：

（1）“%”可以代表**任意长度**的字符串，长度可以为0。**例如**，**b%k**表示以字母b开头，以字母k结尾的任意长度的字符串。该字符串可以代表bk、buk、book、break、bedrock等字符串。

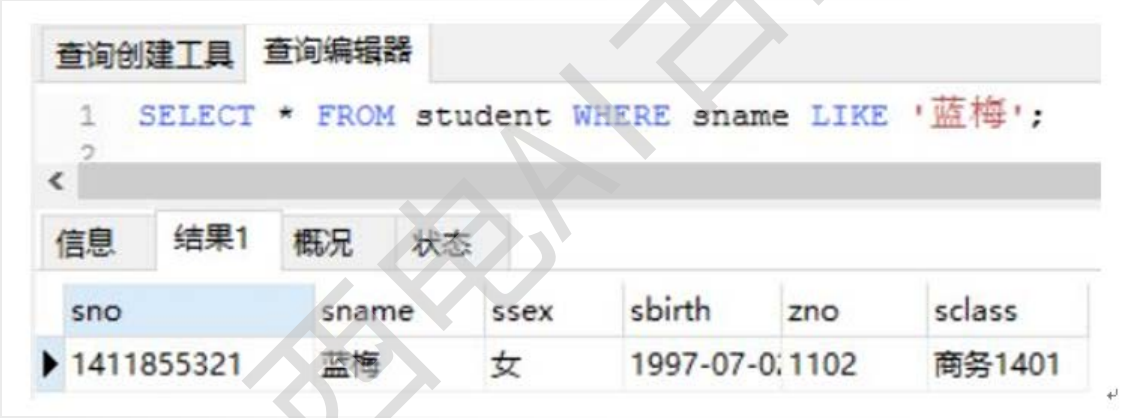
（2）“_”只能表示**单个字符**。**例如**，**b_k**表示以字母b开头，以字母k结尾的3个字符。中间的“_”可以代表任意一个字符。字符串可以代表bok、bak和buk等字符串。

条件查询

示例：

使用LIKE关键字来匹配一个完整的字符串 '蓝梅'。

SELECT * FROM student WHERE sname LIKE '蓝梅';



此处的**LIKE与等号（ = ）是等价的**。可以直接换成 “ = ” ，查询结果是一样的。

SELECT * FROM student WHERE sname ='马小梅';

条件查询

示例：

使用LIKE关键字来匹配带有通配符‘%’的字符串 ‘李%’。

SELECT语句的代码如下：

```
SELECT * FROM student WHERE sname LIKE '李%';
```

查询创建工具 查询编辑器

```
1  SELECT * FROM student WHERE sname LIKE '李%';
2
```

信息 结果1 概况 状态

sno	sname	ssex	sbirth	zno	sclass
1414855302	李壮	男	1996-01-17	1409	会计1401
1418855212	李冬旭	男	1996-06-08	1805	计算1401

条件查询

示例：使用LIKE关键字来匹配带有通配符'_'的字符串。

```
SELECT * FROM student WHERE sname LIKE '李_ _';
```

查询创建工具 查询编辑器

```
1  SELECT * FROM student WHERE sname LIKE '李__';
2
```

信息 结果1 概况 状态

sno	sname	ssex	sbirth	zno	sclass
1418855212	李冬旭	男	1996-06-08	1805	计算1401

- 需要匹配的字符串需要加引号。可以是单引号，也可以是双引号。如果要匹配姓张且名字只有两个字的人的记录，“张”字后面必须有两个“_”符号。因为一个汉字是两个字符，而一个“_”符号只能代表一个字符。因此，匹配的字符串应该为“张_ _”，必须是两个“_”符号。

条件查询

- 假若LIKE ‘字符串’ 中的要匹配的字符串上就含有通配符百分号%或者下划线 “_”,那么我们可以使用**ESCAPE<转化码>**短语，对通配符进行转移。**例如ESCAPE ‘\’表示’\’为转码字符。**这样匹配串中紧跟在 ‘\’后面的字符 “_”不在具有通配符的含义，转义为普通的 “_”字符。
- 如果要查询以 “DB_”开头的，且倒数第三个字符为i的课程的详细情况。可使用如下语句：

```
SELECT * FROM course WHERE cname like 'DB\_%i_
_'ESCAPE '\';
```

条件查询

NOT LIKE表示字符串**不匹配**的情况下满足条件。

示例：

使用NOT LIKE关键字来查询不是姓李的所有人的记录。

```
SELECT * FROM student WHERE sname NOT LIKE '李%';
```

查询创建工具 查询编辑器

```
1  SELECT * FROM student WHERE sname NOT LIKE '李%';
2
```

信息 结果1 概况 状态

sno	sname	ssex	sbirth	zno	sclass
1114070116	欧阳宝贝	女	1997-01-08	1407	工商1401
1114070236	欧阳贝贝	女	1997-01-08	1407	工商1401
1207040137	郑婷	女	1996-05-23	1214	信管1201
1207040237	郑熙婷	女	1996-05-23	1214	信管1201
1411855228	唐晓	女	1997-11-05	1102	商务1401
1411855321	蓝梅	女	1997-07-02	1102	商务1401
1411855426	余小梅	女	1997-06-18	1102	商务1401
1412855223	徐美利	女	1989-09-07	1214	信管1401
1412855313	郭爽	女	1995-02-14	1601	食品1401
1414320425	曹平	女	1997-12-14	1407	工商1401
1414855308	马琦	男	1996-06-14	1409	会计1401
1414855328	刘梅红	女	1991-06-12	1407	工商1401
1414855406	王松	男	1996-10-06	1409	会计1401
1416855305	聂鹏飞	男	1997-08-25	1601	食品1401

条件查询

- MySQL中使用正则表达式查询：REGEXP '匹配方式'
- **常用匹配方式**：

模式字符	含义
^	匹配字符开始部分
$	匹配字符结尾的部分
.	代表字符串中的任意一个字符，包括回车和换行
[字符集合]	匹配字符集合中的任和一个字符
[^字符集合]	匹配除了字符集合以外的任何一个字符
S1|S2|S3	匹配S1、S2、S3中的任意一个字符串
*	代表0个1或者多个其前的字符
+	代表1个或者多个其前的字符
String{N}	字符串出现N次
字符串{M,N}	字符串至少出现M次，最多N次

条件查询

- **正则表达式应用示例**：
  - ✓ 查询用户以t开始的用户（不区分大小写）
    SELECT * FROM cms_user WHERE username REGEXP '^t';
  - ✓ $匹配字符串结尾的部分，查询用户以g结尾的用户（不区分大小写）
    SELECT * FROM cms_user WHERE username REGEXP 'g$';
  - ✓ .代表任意字符（像写LIKE了一个%，和没写效果一样）
    SELECT * FROM cms_user WHERE username REGEXP '.';
  - ✓ 查询用户名包含 “lto”
    SELECT * FROM cms_user WHERE username REGEXP '[lto]';
  - ✓ ‘lg’中间有两位任意字符（使用下划线，错误的）
    SELECT * FROM cms_user WHERE username **REGEXP** 'l__g';
  - ✓ ‘lg’中间有两位任意字符（**使用下划线，通过like来实现**）
    SELECT * FROM cms_user WHERE username **LIKE** 'l_ _g';

高级查询

- **高级查询**：
  - 分组查询
  - 对查询结果进行排序
  - 限制查询结果数量
  - 聚合函数

分组查询

- GROUP BY关键字可以将查询结果按**某个字段或多个字段**进行分组。字段中值相等的为一组。
- **语法格式**：

GROUP BY 字段名 [HAVING 条件表达式][WITH ROLLUP]

- 其中，“字段名”是指按照该字段的值进行分组；“HAVING条件表达式”用来限制分组后的显示，满足条件表达式的结果将被显示；WITH ROLLUP关键字将会在所有记录的最后加上一条记录。该记录是上面所有记录的总和。
- 如果单独使用GROUP BY关键字，查询结果只显示一个分组的一条记录。
- GROUP BY关键字加上“**HAVING条件表达式**”，可以限制输出的结果。只有满足条件表达式的结果才会显示。

分组查询

口 示例：

按student表的ssex字段进行分组查询。

SELECT * FROM student GROUP BY ssex;

查询创建工具 查询编辑器

```
1  SELECT * FROM student GROUP BY ssex;
2
```

信息 结果1 概况 状态

sno	sname	ssex	sbirth	zno	sclass
1114070116	欧阳宝贝	女	1997-01-08	1407	工商1401
1414855302	李壮	男	1996-01-17	1409	会计1401

分组查询

- **示例：**

  按student表的ssex字段进行分组查询。然后显示记录数大于等于10的分组（COUNT()用来统计记录的条数）。

```
SELECT ssex,COUNT(ssex)
FROM student
GROUP BY ssex
HAVING COUNT(ssex)>=10;
```

- **说明：“HAVING条件表达式”与“WHERE条件表达式”都是用来限制显示的。但是，两者起作用的地方不一样。“WHERE条件表达式”作用于表或者视图，是表和视图的查询条件。“HAVING条件表达式”作用于分组后的记录，用于选择满足条件的组。**

对查询结果排序

- 使用**ORDER BY**关键字对记录进行排序。
- **语法格式**： **ORDER BY字段名[ASC｜DESC]**
- 其中，“字段名”参数表示按照该字段进行排序；
  - ASC参数表示按升序的顺序进行排序；
  - DESC参数表示按降序的顺序进行排序。
  - **默认的情况下，按照ASC方式进行排序。**

对查询结果排序

- 示例：

查询student表中所有记录，按照zno字段进行排序。

SELECT * FROM student ORDER BY zno ;

查询创建工具 查询编辑器

```
1  SELECT * FROM student ORDER BY zno ;
2
```

信息 结果1 概况 状态

sno	sname	ssex	sbirth	zno	sclass
1411855228	唐晓	女	1997-11-05	1102	商务1401
1411855321	蓝梅	女	1997-07-02	1102	商务1401
1411855426	余小梅	女	1997-06-18	1102	商务1401
1418855236	张强	男	1996-03-15	1103	商务1301
1207040137	郑婷	女	1996-05-23	1214	信管1201
1207040237	郑熙婷	女	1996-05-23	1214	信管1201
1412855223	徐美利	女	1989-09-07	1214	信管1401
1114070116	欧阳宝贝	女	1997-01-08	1407	工商1401
1114070236	欧阳贝贝	女	1997-01-08	1407	工商1401
1414320425	曹平	女	1997-12-14	1407	工商1401
1414855328	刘梅红	女	1991-06-12	1407	工商1401

对查询结果排序

- **注意**：如果存在一条记录zno字段的值为**空值(NULL)**时，这条记录将显示为**第一条记录**。因为，按**升序**排序时，含空值的记录将**最先**显示。可以理解为空值是该字段的最小值。而按**降序**排列时，zno字段为空值的记录将**最后**显示。
- MySQL中，可以指定按**多个字段**进行排序。**例如**，可以使student表按照zno字段和sno字段进行排序。排序过程中，先按照zno字段进行排序。遇到zno字段的值相等的情况时，再把zno值相等的记录按照sno字段进行排序。

对查询结果排序

- **示例：**
- 查询student表中所有记录，按照zno字段的升序方式和sno字段的降序方式进行排序。SELECT语句如下：
- SELECT * FROM student ORDER BY zno ASC,sno DESC;

查询创建工具 查询编辑器

```
1  SELECT * FROM student ORDER BY zno ASC,sno DESC;
2
```

信息 结果1 概况 状态

sno	sname	ssex	sbirth	zno	sclass
1411855426	余小梅	女	1997-06-18	1102	商务1401
1411855321	蓝梅	女	1997-07-02	1102	商务1401
1411855228	唐晓	女	1997-11-05	1102	商务1401
1418855236	张强	男	1996-03-15	1103	商务1301
1412855223	徐美利	女	1989-09-07	1214	信管1401
1207040237	郑熙婷	女	1996-05-23	1214	信管1201
1207040137	郑婷	女	1996-05-23	1214	信管1201
1414855328	刘梅红	女	1991-06-12	1407	工商1401
1414320425	曹平	女	1997-12-14	1407	工商1401
1114070236	欧阳贝贝	女	1997-01-08	1407	工商1401
1114070116	欧阳宝贝	女	1997-01-08	1407	工商1401
1414855406	王松	男	1996-10-06	1409	会计1401
1414855308	马琦	男	1996-06-14	1409	会计1401
1414855302	李杜	男	1996-01-17	1409	会计1401

限制查询结果数量

- 当使用SELECT语句返回的结果集中行数很多时，为了便于用户对结果数据的浏览和操作，可以使用**LIMIT子句**来限制被SELECT语句返回的行数。
- **语法格式：** **LIMIT {[offset，] row_count | row_count OFFSET offset}**
- 其中，offset为可选项，默认为数字0，用于指定返回数据的第一行在SELECT语句结果集中的偏移量，其必须是非负的整数常量。注意，SELECT语句结果集中第一行（初始行）的偏移量为0而不是1。row_count用于指定返回数据的行数，其也必须是非负的整数常量。若这个指定行数大于实际能返回的行数时，MySQL将只返回它能返回的数据行。
- row_count OFFSET offset是MySQL5.0开始支持的另外一种语法，即从第offset+1行开始，取row_count行。

限制查询结果数量

口 示例：

在student表中查找从第3名同学开始的3位学生的信息。

```
SELECT * FROM student ORDER BY sno LIMIT 2,3;
```

查询创建工具 | 查询编辑器

```
1  SELECT * FROM student ORDER BY sno LIMIT 2,3;
2
```

信息 | 结果1 | 概况 | 状态

sno	sname	ssex	sbirth	zno	sclass
1207040137	郑婷	女	1996-05-23	1214	信管1201
1207040237	郑熙婷	女	1996-05-23	1214	信管1201
1411855228	唐晓	女	1997-11-05	1102	商务1401

聚合函数

- 集合函数包括COUNT()、SUM()、 AVG()、 MAX()和MIN()。其中，
  - **COUNT()**用来**统计记录的条数**，Count（字段）不统计null值；
  - **SUM()**用来计算字段的值的**总和**；
  - **AVG()**用来计算字段的值的**平均值**；
  - **MAX()**用来查询字段的**最大值**；
  - **MIN()**用来查询字段的**最小值**。
- GROUP BY关键字通常需要与集合函数一起使用。

<1>如果某个给定行中的一列仅包含NULL值，则函数的值等于NULL值。

<2>如果一列中的某些值为NULL值，则函数的值等于所有非NULL值的平均值除以非NULL值的数量（不是除以所有值）。

<3>对于必须计算的SUM和AVG函数，如果中间结果为空，则函数的值等于NULL值。

聚合函数

（1）COUNT()函数

- COUNT用于统计组中满足条件的行数或总行数。
- **格式如下**：

  COUNT（{[ALL I DISTINCT]<表达式>}I* ）

- ALL、DISTINCT的含义及默认值与SUM/AVG函数相同。
- 选择*时将统计总行数。
- COUNT用于计算列中非NULL值的数量。
- 如果要统计student表中有多少条记录，可以使用COUNT()函数。

聚合函数

口 示例：

使用COUNT()函数统计student表不同zno值的记录数。COUNT()函数与GROUP BY关键字一起使用。

SELECT zno,COUNT(*) AS '专业人数' FROM student GROUP BY zno;

查询创建工具 查询编辑器

```
1  SELECT zno,COUNT(*) as '专业人数'
2  FROM student
3  GROUP BY zno;
```

信息 结果1 概况 状态

zno	专业人数
1102	3
1103	1
1214	3
1407	5
1409	3
1601	4
1805	2

聚合函数

（2）SUM()函数

- SUM()函数是求和函数。使用SUM()函数可以求表中某个字段取值的总和。
- **示例：**

使用SUM()函数统计sc表中学号为1414855328的同学的总成绩。

```
SELECT sno,SUM(grade)
FROM sc
WHERE sno='1414855328';
```

查询创建工具 查询编辑器

```
1  SELECT sno,SUM(grade)
2  FROM sc
3  WHERE sno='1414855328';
```

信息 结果1 概况 状态

sno	SUM(grade)
1414855328	181

聚合函数

- SUM()函数通常和GROUP BY关键字一起使用。这样可以计算出不同分组中某个字段取值的总和。
- **示例：**将sc表按照sno字段进行分组。然后，使用SUM()函数统计各分组的总成绩。

```
SELECT sno,SUM(grade)
FROM sc
GROUP BY sno ;
```

- SUM()函数只能计算数值类型的字段:INT、FLOAT、DOUBLE、DECIMAL等。

信息 | 结果1 | 概况 | 状态

sno	SUM(grade)
1114070116	155
1411855228	96
1411855321	69
1412855223	137
1414855302	90
1414855328	181
1414855406	245
1418855232	178

聚合函数

（3）AVG()函数

- AVG()函数是求平均值的函数。使用AVG()函数可以求出表中某个字段取值的平均值。

示例：

使用AVG()函数计算sc表中平均成绩。

```
SELECT AVG(grade)
FROM sc;
```

查询创建工具 查询编辑器

```
1  SELECT AVG(grade)
2  FROM sc;
```

信息 结果1 概况 状态

AVG(grade)
82.2143

聚合函数

口 示例：使用AVG()函数计算sc表中不同科目的平均成绩。

```
SELECT cno,AVG(grade)
FROM sc
GROUP BY cno;
```

查询创建工具 查询编辑器

```
1  SELECT cno,AVG(grade)
2  FROM sc
3  GROUP BY cno;
```

信息 结果1 概况 状态

cno	AVG(grade)
11110140	90.0000
11110470	77.5000
11110930	65.0000
18110140	81.0000
18130320	82.0000
18132220	90.0000
58130540	84.3333

聚合函数

（4）MAX()函数

- MAX()函数是求最大值的函数。使用MAX()函数可以求出表中某个字段取值的最大值。
- MAX()不仅仅适用于**数值**类型，也适用于**字符**类型。
- MAX()函数是使用字符对应的**ASCII码**进行计算的。
  - 在MySQL表中，字母a最小，字母z最大。因为a的ASCII码值最小。在使用MAX()函数进行比较时，先比较第一个字母。如果第一个字母相等时，再继续往下一个字母进行比较。

（5）MIN()函数

MIN()函数是求最小值的函数。使用MIN()函数可以求出表中某个字段取值的最小值。

聚合函数

口 示例：使用MAX()函数查询sc表中不同科目的最高成绩。

```
SELECT sno,cno,MAX(grade)
FROM sc
GROUP BY cno;
```

查询创建工具 查询编辑器

```
1  SELECT sno,cno,MAX(grade)
2  FROM sc
3  GROUP BY cno;
```

信息 结果1 概况 状态

sno	cno	MAX(grade)
1414855302	11110140	90
1414855406	11110470	86
1114070116	11110930	65
1414855406	18110140	87
1412855223	18130320	96
1411855228	18132220	96
1414855328	58130540	91

聚合函数

- **示例：**使用MAX()函数查询student表中sname字段的最大值。SELECT语句如下：

```
SELECT MAX(sname)
FROM student;
```

多表查询

什么是连接查询？

- ✓ 连接查询是**将两个或两个以上的表按某个条件连接起来，从中选取需要的数据**。
- ✓ 连接查询是同时查询两个或两个以上的表时使用的。当不同的表中存在相同意义的字段时，可以通过该字段连接这几个表。
- ✓ 连接查询就是找到表与表之间关系，或者找到表与表之间的连接的桥梁。

内连接查询（常用的）

JOIN|CROSS JOIN INNER JOIN

通过ON 连接条件

显示两个表中符合连接条件的记录（超过3个表效率低）

多表查询

- **连接查询类型：**
  - **内连接查询**和**外连接查询**。
  - 主要区别在于，内连接仅选出两张表中互相匹配的记录，而外连接会选出其他不匹配的记录。
  - 最常用的是内连接查询。
- **子查询**操作：代替连接查询，具有更高的操作效率。

内连接查询

- 内连接查询是**最常用的一种查询**，也称为等同查询，就是在表关系的笛卡尔积数据记录中，保留表关系中所有相匹配的数据，而舍弃不匹配的数据。
- 按照**匹配条件**可以分为
  - **自然连接**
  - **等值连接**
  - **不等值连接**

等值连接（inner join）

- 用来连接两个表的条件称为连接条件。
- 如果连接条件中的连接运算符是=时，称为**等值连接**。
- **示例：**对选修表和课程表做等值连接(返回的结果限制在4条以内)

查询创建工具 | 查询编辑器

```
1  select *
2  from sc inner join course
3  on sc.cno=course.cno
4  limit 4;
```

信息 | 结果1 | 概况 | 状态

sno	cno	grade	cno1	cname	ccredit	cdept
1414855328	58130540	85	58130540	大数据技术及应用	3	信息学院
1414855406	18110140	75	18110140	C语言程序设计	3	信息学院
1412855223	18130320	60	18130320	Internet技术及应用	2	信息学院
1114070116	11110930	65	11110930	电子商务	2	经济管理学院

自然连接（natural join）

- 自然连接操作就是表关系的笛卡尔积中选取满足连接条件的行。
- 具体过程是，首先根据表关系中相同名称的字段进行记录匹配，然后去掉重复的字段。
- 还可以理解为**在等值连接中把目标列中重复的属性列去掉则为自然连接。**
- **示例：**对选修表和课程表做自然连接(返回的结果限制在4条以内)

```
SELECT *
    FROM sc natural join course
limit 4;
```

自然连接（natural join）

- **示例：**对选修表和课程表做自然连接(返回的结果限制在4条以内)
- **自然连接结果：**

查询创建工具 | 查询编辑器

```
1  select *
2  from sc natural join course
3  limit 4;
4
```

信息 | 结果1 | 概况 | 状态

cno	sno	grade	cname	ccredit	cdept
58130540	1414855328	85	大数据技术及应用	3	信息学院
18110140	1414855406	75	C语言程序设计	3	信息学院
18130320	1412855223	60	Internet技术及应用	2	信息学院
11110930	1114070116	65	电子商务	2	经济管理学院

不等值连接（inner join）

- 用来连接两个表的条件称为连接条件。
- 如果连接条件中的连接运算符是=时，称为等值连接。
- 如果是**其他的运算符**，则是**不等值连接**。
- **例：**对选修表和课程表做不等值连接(返回的结果限制在4条以内)

```
SELECT *
    FROM sc inner join course
    on sc.cno!=course.cno
limit 4;
```

不等值连接（inner join）

- **示例**：对选修表和课程表做不等值连接(返回的结果限制在4条以内)
- **不等值连接结果**：

查询创建工具 查询编辑器

```
1  select *
2  from sc inner join course
3  on sc.cno!=course.cno
4  limit 4;
```

信息 结果1 概况 状态

sno	cno	grade	cno1	cname	ccredit	cdept
1414855328	58130540	85	11110140	管理信息系统	3	经济管理学院
1414855406	18110140	75	11110140	管理信息系统	3	经济管理学院
1412855223	18130320	60	11110140	管理信息系统	3	经济管理学院
1114070116	11110930	65	11110140	管理信息系统	3	经济管理学院

外连接查询

- 外连接可以查询两个或两个以上的表，外连接查询和内连接查询非常的相似，也需要通过指定字段进行连接，当该字段取值相等时，可以查询出该表的记录。而且，该字段取值不相等的记录也可以查询出来。
- 外连接查询**分类**:

1.左外连接:LEFT [OUTER] JOIN，显示左表的全部记录及右表符合连接条件的记录（以左为主）

2.右外连接:RIGHT [OUTER] JOIN，显示右表的全部记录以及左表符合条件的记录（以右为主）

- **基本语法**如下：

SELECT 字段表 FROM 表1 LEFT | RIGHT [OUTER] JOIN 表2 ON 表1.字段=表2.字段

左外连接（left join）

- 左外连接的结果集中包含左表（JOIN关键字左边的表）中所有的记录，然后左表按照连接条件与右表进行连接。如果右表中没有满足连接条件的记录，则结果集中右表中的相应行数据填充为**NULL**。
- **例：**利用左连接方式 查询课程表和选修表

```
SELECT course.cno, course.cname,sc.cno,sc.sno
FROM course left join sc
on course.cno=sc.cno
limit 10;
```

左外连接（left join）

- 例：利用左连接方式查询课程表和选修表

信息 | 结果1 | 概况 | 状态

cno	cname	cno1	sno
11110140	管理信息系统	11110140	1414855302
11110470	统计学A	11110470	1414855406
11110470	统计学A	11110470	1411855321
11110930	电子商务	11110930	1114070116
11111260	客户关系管理	(Null)	(Null)
11140260	网站规划与运营实训	(Null)	(Null)
18110140	C语言程序设计	18110140	1414855406
18110140	C语言程序设计	18110140	1418855232
18111850	数据库原理	(Null)	(Null)
18112820	网站设计与开发	(Null)	(Null)

右外连接（right join）

- 右外连接的结果集中包含满足连接条件的所有数据和右表（JOIN关键字右边的表）中不满足条件的数据，**左表中**的相应行数据为**NULL**。
- **例：**利用右连接方式查询课程表和选修表

```
SELECT course.cno, course.cname,sc.cno,sc.sno
FROM course right join sc
on course.cno=sc.cno
limit 10;
```

右外连接（right join）

- 例：利用右连接方式查询课程表和选修表

信息 | 结果1 | 概况 | 状态

cno	cname	cno1	sno
58130540	大数据技术及应用	58130540	1414855328
18110140	C语言程序设计	18110140	1414855406
18130320	Internet技术及应用	18130320	1412855223
11110930	电子商务	11110930	1114070116
11110140	管理信息系统	11110140	1414855302
18132220	数据库技术及应用	18132220	1411855228
18110140	C语言程序设计	18110140	1418855232
18130320	Internet技术及应用	18130320	1414855328
11110470	统计学A	11110470	1414855406
58130540	大数据技术及应用	58130540	1412855223

联合查询

- **联合查询**：查询多个表的记录
- **语法格式**：
  - ✓ **UNION**
  - ✓ **UNION ALL**
- UNION和UNION ALL 区别是UNION去掉相同记录，UNION ALL 是简单的合并到一起。保证表的查询字段相同。
- **示例：**

✓ **使用UNION查询(查询多个表就可以使用它)：**

```
SELECT username FROM employee UNION SELECT username FROM cms_user;
```

✓ **使用UNION ALL查询（只是简单地合并）：**

```
SELECT username FROM employee UNION ALL SELECT username FROM cms_user;
```

子查询

- **什么是子查询？**

  子查询是**将一个查询语句嵌套在另一个查询语句中**。内层查询语句的查询结果，可以作为外层查询语句提供条件。**执行时是从内层查询到外层查询。**

- **引发子查询的情况**：

  1.使用**[NOT]IN**的子查询 ‘

  2.使用**比较运算符**的子查询，=、>、 <、>=、 <=、<>、 !=、 <=>；

  3.使用**[NOT]EXISTS**的子查询。

带IN关键字的子查询

- 一个查询语句的条件可能落在另一个SELECT语句的查询结果中。这可以通过IN关键字来判断。
- IN关键字可以判断某个字段的值是否在指定的**集合**中，如果字段的值在集合中，则满足查询条件，该记录将被查询出来；如果不在集合中，则不满足查询条件。
- **语法格式**： **[NOT] IN (元素1,元素2, 元素3, . . .)**
- 其中，NOT是可选参数，加上NOT表示不在集合内满足条件：字符型元素要加上单引号。

带IN关键字的子查询

- **示例：**查询选修过课程的student的记录。

```
SELECT *
FROM student
WHERE sno IN( SELECT sno FROM sc);
```

查询创建工具 查询编辑器

```
1  select *
2  from course
3  where exists ( select * from specialty where zname='金融' );
```

信息 结果1 概况 状态

cno	cname	ccredit	cdept
(Null)	(Null)	(Null)	(Null)

带EXISTS关键字的子查询

- exists关键字表示存在，使用exists关键字时，内查询语句不返回查询的记录。而是**返回一个真假值。**
- 如果内层查询语句查询到满足条件的记录，就会返回一个真值**true**，否则返回**false**。
- **当返回true时，外查询进行查询**，否则外查询不进行查询。

带EXISTS关键字的子查询

示例：如果存在 “计算机科学与技术”这个专业，就查询所有的课程信息。

```
SELECT *
FROM course
WHERE exists ( SELECT * FROM specialty
 WHERE zname='计算机科学与技术' )
limit 2;
```

带ANY关键字的子查询

- ANY关键字表示满足其中任何一个条件。使用ANY关键字时，只要满足内查询语句返回结果中的**一个**，就可以通过该条件来执行外层查询语句。
- **示例：**查询比其他班级（比如，计算1401班级）某一个同学年龄小的学生的姓名和年龄。

```
SELECT sname ,
(date_format(from_days(to_days(now())- to_days(sbirth)),'%Y') +
0) as '年龄'
FROM student WHERE sbirth > ANY (
SELECT sbirth FROM student
WHERE sclass='计算1401'
);
```

带ANY关键字的子查询

- **示例：**查询比其他班级（比如，计算1401班级）某一个同学年龄小的学生的姓名和年龄。

查询创建工具 查询编辑器

```
1  select sname , (date_format(from_days(to_days(now()) - to_days(sbirth)),'%Y') + 0) as '年龄'
2  from student
3  where sbirth > ANY (
4  select sbirth
5  from student
6  where sclass='计算1401'
7  );
```

信息 结果1 概况 状态

sname	年龄
欧阳宝贝	20
欧阳贝贝	20
郑婷	20
郑熙婷	20
唐晓	19
蓝梅	19
余小梅	19
曹平	19
马琦	20
王松	20

带ANY关键字的子查询

- 为了对照结果，我们给出计算1401班所有人的年龄如下。

查询创建工具 查询编辑器

```
1  select sname , (date_format(from_days(to_days(now()) - to_days(sbirth)),'%Y') + 0) as '年龄'
2  from student
3  where  sclass='计算1401';
```

信息 结果1 概况 状态

sname	年龄
李冬旭	20
王琴雪	19
李苹	21
李一苹	20

带ALL关键字的子查询

- ALL关键字表示满足所有的条件。使用ALL关键字时，只有满足内层查询语句返回的**所有结果**，才能执行外层的查询语句。 >ALL表示大于所有的值，<ALL表示小于所有的值。
- ALL关键字和ANY关键字的使用方式一样，但两者的差距很大，前者是满足所有的内层查询语句返回的所有结果，才执行外查询，后者是只需要满足其中一条记录，就执行外查询。

带ALL关键字的子查询

- **示例：**查询比其他班级（比如计算1401班级）所有同学年龄都大的学生的姓名和年龄。

```
SELECT sname ,
(date_format(from_days(to_days(now()) - to_days(sbirth)),'%Y')
+ 0) as '年龄'
FROM student
WHERE sbirth < ALL (
SELECT sbirth
FROM student
WHERE sclass='计算1401'
);
```

带ALL关键字的子查询

- 查询比其他班级（比如计算1401班级）所有同学年龄都大的学生的姓名和年龄。

查询创建工具 查询编辑器

```
1  select sname , (date_format(from_days(to_days(now()) - to_days(sbirth)),'%Y') + 0) as '年龄'
2  from student
3  where sbirth < ALL (
4  select sbirth
5  from student
6  where sclass='计算1401'
7  );
```

信息 结果1 概况 状态

sname	年龄
徐美利	27
郭爽	22
李壮	21
刘梅红	25

用查询语句插入表

- ✓ 创建t_test1表

```
CREATE TABLE t_test1 (
    id TINYINT UNSIGNED AUTO_INCREMENT KEY,
    num TINYINT UNSIGNED
    );
```

- ✓ **插入导入学员表的记录**

```
INSERT t_test1(id,num)  SELECT id,score FROM student;
```

- ✓ **创建t_test2表时导入学员表的记录**

```
CREATE TABLE t_test2 (
    id TINYINT UNSIGNED AUTO_INCREMENT KEY,
    num TINYINT UNSIGNED
    )SELECT id,score FROM student;
```

- 8.1 插入数据
- 8.2 修改数据
- 8.3 删除数据
- 8.4 查询（单表查询，多表查询）
- **8.5 知识点小结**
- 本章实验

8.6 知识点小结

本章知识小结：

- 数据库的插入、修改、删除、查询
- SELECT语句的使用方法及语法要素

第8章 MySQL数据操作管理

- 8.1 插入数据
- 8.2 修改数据
- 8.3 删除数据
- 8.4 查询（单表查询，多表查询）
- 8.5 知识点小结
- **本章实验**

本章实验

□ 实验内容：

（5）对表中记录进行插入、修改、删除：

①向student表的sno，sname，ssex，sbirth字段插入数据；

②更新student表中sno值为152011的记录，将sclass字段的值变为“人工智能05”，将zno字段的值变为“1513”；

③删除student表中sname值为“张三”的记录。

（6）对表中记录进行查询：

①查询sc表中成绩在75到85之间学生的学号和成绩，要求返回结果中列名为学号、成绩，而不是sno、grade；

②查询“人工智能02”班中所有姓王的学生记录；

③计算sc表中不同科目的最高成绩；

④使用自然连接方式查询专业号为“1502”的学生选修的所有课程的编号。

Thanks !

See you