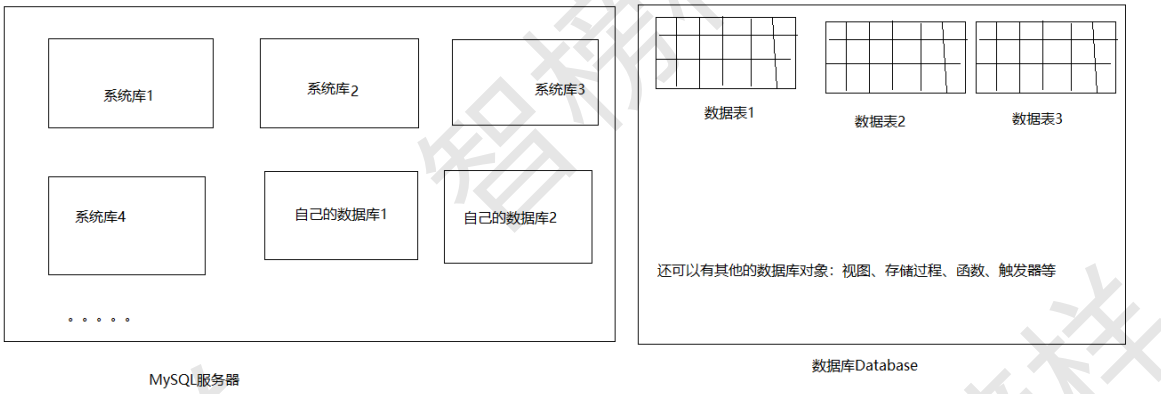
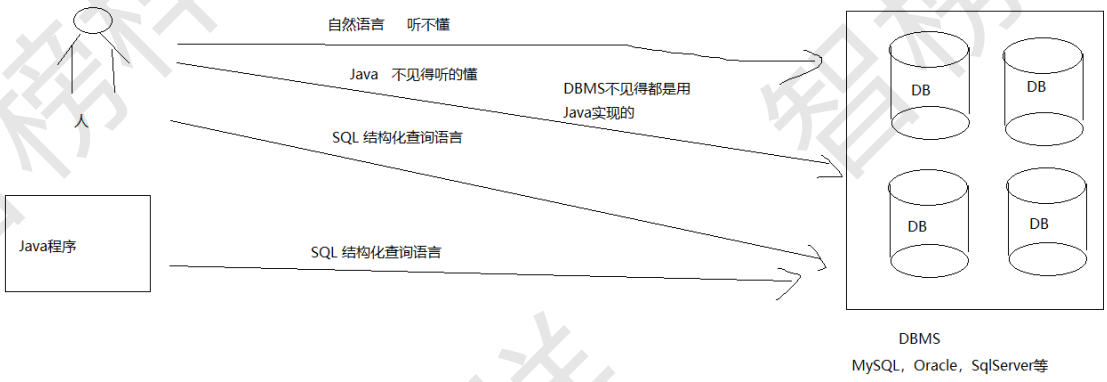
第1章 MySQL数据库概述

1.1 基本概念

- 数据库是什么?
  - 存储数据的地方
  - DB：数据库（Database）
- 为什么要用数据库?
  - 因为应用程序产生的数据是在内存中的，如果程序退出或者是断电了，则数据就会消失。使用数据库是为了能够永久保存数据。当然这里指的是非内存数据库。
- 用普通文件存储行不行?
  - 把数据写入到硬盘上的文件中，当然可以实现持久化的目标，但是不利于后期的检索和管理等。
- MySQL、Oracle、SqlServer是什么?
  - MySQL、Oracle、SqlServer都是数据库管理系统（DBMS，Database Management System）是一种操纵和管理数据库的大型软件，例如建立、使用和维护数据库。
- SQL是什么?
  - SQL是结构化查询语言（Structure Query Language），专门用来操作/访问数据库的通用语言。





1.2 MySQL数据库管理系统

在互联网行业，MySQL数据库毫无疑问已经是最常用的数据库。MySQL数据库由瑞典MySQL AB公司开发。公司名中的“AB”是瑞典语“aktiebolag”股份公司的首字母缩写。该公司于2008年1月16号被Sun（Stanford University Network）公司收购。然而2009年，SUN公司又被Oracle收购。因此，MySQL数据库现在隶属于Oracle（甲骨文）公司。MySQL中的“My”是其发明者（Michael Widenius，通常称为Monty）根据其女儿的名字来命名的。对这位发明者来说，MySQL数据库就仿佛是他可爱的女儿。

- 什么是关系型数据库和非关系数据库。
  - MySQL、Oracle、SqlServer等是关系型数据库管理系统。
  - MongoDB、Redis、Elasticsearch等是非关系型数据库管理系统。

表 -- 类

字段、属性　列

学号	姓名	年龄	性别	专业
161228001	张三	20	男	JavaEE
161228002	李四	19	女	H5
161228003	王五	21	男	Android
161228004	赵六	20	女	大数据
161228005	钱七	23	男	Python

行，记录　对象，实体

属性值

关系型数据库，采用关系模型来组织数据，简单来说，**关系模型指的就是二维表格模型**。类似于Excel工作表。**非关系型数据库**，可看成传统关系型数据库的功能阉割版本，基于键值对存储数据，通过减少很少用的功能，来提高性能。

以下是2021年《DB-Engines Ranking》对各数据库受欢迎程度进行调查后的统计结果（查看数据库最新排名：https://db-engines.com/en/ranking）

373 systems in ranking, July 2021

Rank Jul 2021	Rank Jun 2021	Rank Jul 2020	DBMS	Database Model	Score Jul 2021	Score Jun 2021	Score Jul 2020
1.	1.	1.	Oracle	Relational, Multi-model	1262.66	-8.28	-77.59
2.	2.	2.	MySQL	Relational, Multi-model	1228.38	+0.52	-40.13
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	981.95	-9.12	-77.77
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.15	+8.64	+50.15
5.	5.	5.	MongoDB	Document, Multi-model	496.16	+7.95	+52.68
6.	↑ 7.	↑ 8.	Redis	Key-value, Multi-model	168.31	+3.06	+18.26
7.	↓ 6.	↓ 6.	IBM Db2	Relational, Multi-model	165.15	-1.88	+1.99
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	155.76	+1.05	+4.17
9.	9.	9.	SQLite	Relational	130.20	-0.33	+2.75
10.	↑ 11.	10.	Cassandra	Wide column	114.00	-0.11	-7.08
11.	↓ 10.	11.	Microsoft Access	Relational	113.45	-1.49	-3.09
12.	12.	12.	MariaDB	Relational, Multi-model	97.98	+1.19	+6.86
13.	13.	13.	Splunk	Search engine	90.05	-0.22	+1.78
14.	14.	14.	Hive	Relational	82.68	+2.98	+6.25
15.	15.	↑ 18.	Microsoft Azure SQL Database	Relational, Multi-model	75.22	+0.43	+22.59
16.	16.	16.	Amazon DynamoDB	Multi-model	75.20	+1.43	+10.62
17.	17.	↓ 15.	Teradata	Relational, Multi-model	68.95	-0.39	-7.02
18.	18.	↑ 22.	Neo4j	Graph	57.16	+1.41	+8.24
19.	19.	↑ 20.	SAP HANA	Relational, Multi-model	53.81	-0.29	+2.48
20.	20.	↓ 19.	Solr	Search engine, Multi-model	51.79	-0.30	+0.15

MySQL的优点有很多，其中主要的优势有如下几点：

- 可移植性：MySQL数据库几乎支持所有的操作系统，如Linux、Solaris、FreeBSD、Mac和Windows。
- 免费：MySQL的社区版完全免费，一般中小型网站的开发都选择 MySQL 作为网站数据库。
- 开源：2000 年，MySQL公布了自己的源代码，并采用GPL（GNU General Public License）许可协议，正式进入开源的世界。开源意味着可以让更多人审阅和贡献源代码，可以吸纳更多优秀人才的代码成果。
- 关系型数据库：MySQL可以利用标准SQL语法进行查询和操作。

- 速度快、体积小、容易使用：与其他大型数据库的设置和管理相比，其复杂程度较低，易于学习。MySQL的早期版本（主要使用的是MyISAM引擎）在高并发下显得有些力不从心，随着版本的升级优化（主要使用的是InnoDB引擎），在实践中也证明了高压力下的可用性。从2009年开始，阿里的“去IOE”备受关注，淘宝DBA团队再次从Oracle转向MySQL，其他使用MySQL数据库的公司还有Facebook、Twitter、YouTube、百度、腾讯、去哪儿、魅族等等，自此，MySQL在市场上占据了很大的份额。
- 安全性和连接性：十分灵活和安全的权限和密码系统，允许基于主机的验证。连接到服务器时，所有的密码传输均采用加密形式，从而保证了密码安全。由于MySQL是网络化的，因此可以在因特网上的任何地方访问，提高数据共享的效率。
- 丰富的接口：提供了用于C、C++、Java、PHP、Python、Ruby和Eiffel、Perl等语言的API。
- 灵活：MySQL并不完美，但是却足够灵活，能够适应高要求的环境。同时，MySQL既可以嵌入到应用程序中，也可以支持数据仓库、内容索引和部署软件、高可用的冗余系统、在线事务处理系统等各种应用类型。
- MySQL最重要、最与众不同的特性是它的存储引擎架构，这种架构的设计将查询处理（Query Processing）及其他系统任务（Server Task）和数据的存储/提取相分离。这种处理和存储分离的设计可以在使用时根据性能、特性，以及其他需求来选择数据存储的方式。MySQL中同一个数据库，不同的表格可以选择不同的存储引擎。其中使用最多的是InnoDB 和MyISAM，MySQL5.5之后InnoDB是默认的存储引擎。

针对不同用户，MySQL提供三个不同的版本。

（1）MySQL Enterprise Server（企业版）：能够以更高的性价比为企业提供数据仓库应用，该版本需要付费使用，官方提供电话技术支持。

（2）MySQL Cluster（集群版）：MySQL 集群是 MySQL 适合于分布式计算环境的高可用、高冗余版本。它采用了 NDB Cluster 存储引擎，允许在 1 个集群中运行多个 MySQL 服务器。它不能单独使用，需要在社区版或企业版基础上使用。

（3）MySQL Community Server（社区版）：在开源GPL许可证之下可以自由的使用。该版本完全免费，但是官方不提供技术支持。本书是基于社区版讲解和演示的。在MySQL 社区版开发过程中，同时存在多个发布系列，每个发布处在不同的成熟阶段。

- MySQL5.7（RC）是当前稳定的发布系列。RC（Release Candidate候选版本）版只针对严重漏洞修复和安全修复重新发布，没有增加会影响该系列的重要功能。从MySQL 5.0、5.1、5.5、5.6直到5.7都基于5这个大版本，升级的小版本。5.0版本中加入了存储过程、服务器端游标、触发器、视图、分布式事务、查询优化器的显著改进以及其他的一些特性。这也为MySQL 5.0之后的版本迈向高性能数据库的发展奠定了基础。
- MySQL8.0.26（GA）是最新开发的稳定发布系列。GA（General Availability正式发布的版本）是包含新功能的正式发布版本。这个版本是MySQL数据库又一个开拓时代的开始。

第2章 客户端使用演示

2.1.1 数据库

1、查看所有的数据库

```
show databases;
```

2、创建自己的数据库

```
create database 数据库名;

#创建zbydb数据库
create database zbydb;
```

3、删除数据库

```
drop database 数据库名;
```

```
#删除zbydb数据库
drop database zbydb;
```

4、使用自己的数据库

```
use 数据库名;

#使用zbydb数据库
use zbydb;
```

说明：如果没有使用use语句，后面针对数据库的操作也没有加“数据名”的限定，那么会报“ERROR 1046 (3D000): No database selected”（没有选择数据库）

使用完use语句之后，如果接下来的SQL都是针对一个数据库操作的，那就不用重复use了，如果要针对另一个数据库操作，那么要重新use。

2.1.2 数据表

1、查看某个库的所有表格

```
show tables;  #要求前面有use语句

show tables from 数据库名;
```

2、创建新的表格

```
create table 表名称(
    字段名  数据类型,
    字段名 数据类型
);
```

说明：如果是最后一个字段，后面就用加逗号，因为逗号的作用是分割每个字段。

```
#创建学生表
create table student(
    id int,
    name varchar(20)  #说名字最长不超过20个字符
);
```

3、查看定义好的表结构

```
desc 表名称;
```

```
desc student;
```

4、添加一条记录

```
insert into 表名称 values(值列表);

#添加两条记录到student表中
insert into student values(1,'张三');
insert into student values(2,'李四');
```

5、查看一个表的数据

```
select * from 表名称;
```

6、删除表

```
drop table 表名称;
```

```
#删除学生表
drop table student;
```

2.2 可视化客户端

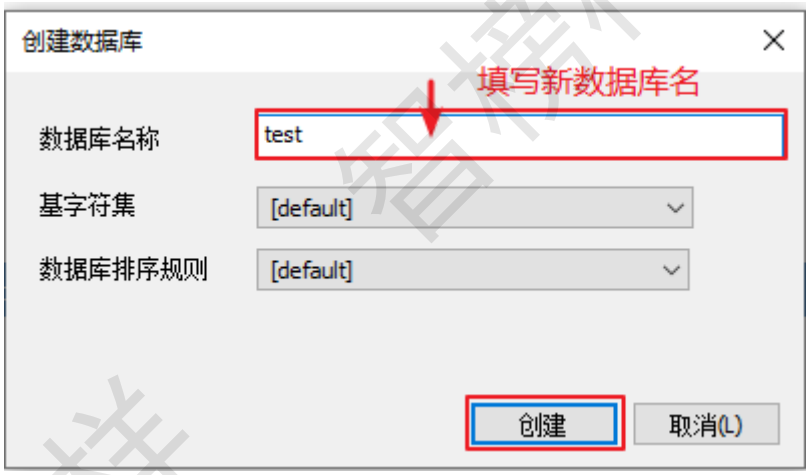
前面介绍了通过命令行来创建数据库和数据表，除此之外，还可以借助MySQL图形化工具，而且这种方式更加简单、方便。下面以SQLyog图形化工具为代表展开介绍。

2.2.1 数据库

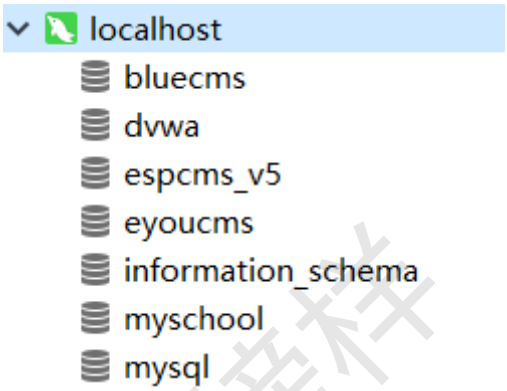
以查看所有数据库的方式登录连接成功后，进入主界面，接下来正式创建数据库。

步骤1：在主界面左边“数据库对象导航窗口”的空白处右键单击鼠标弹出快捷菜单，选择“新建数据库”菜单项。

步骤2：填写新数据库的基本信息。一般只需填写数据库名称，例如“test”，字符集和排序规则有默认选项。如果有特殊要求，也可以选择自己需要的字符集和校对规则，然后点击“创建”按钮。



步骤3：此时数据库创建成功。

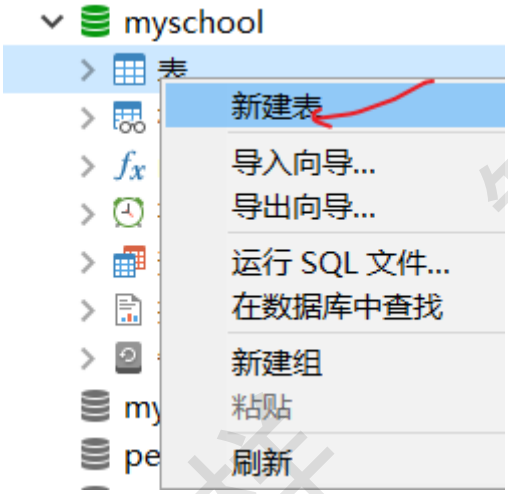


步骤4：数据库创建成功之后，可以查看或修改数据库属性。选择“myschool”数据库，右键单击鼠标弹出快捷菜单，选择“编辑数据库”菜单选项，就可以查看或修改数据库属性。

2.2.2 数据表

数据库创建成功之后，就可以在这个数据库下面创建表了。

步骤1：选择“myschool”数据库下的“表”对象，右键单击鼠标弹出快捷菜单，选择“新建表”菜单项。



步骤2：点击“新建表”后，在新表的创建页面填写表名称和表的字段等属性信息。例如表名称为“tb_student”，并添加两个字段，一个是int类型的sid，一个是varchar(20)类型的sname，分别表示学生编号和学生姓名。

第3章 SQL语句

SQL：结构化查询语言，（Structure Query Language），专门用来操作/访问数据库的通用语言。

3.1 SQL的分类

DDL语句：数据定义语句（Data Define Language），例如：创建（create），修改（alter），删除（drop）等

DML语句：数据操作语句，例如：增（insert），删（delete），改（update），查（select）

因为查询语句使用的非常的频繁，所以很多人把查询语句单拎出来一类，DQL（数据查询语言），DR（获取）L

DCL语句：数据控制语句，例如：grant，commit，rollback等

其他语句：USE语句，SHOW语句，SET语句等。这类的官方文档中一般称为命令。

3.2 SQL语法规范

（1）mysql的sql语法不区分大小写

A：数据库的表中的数据是否区分大小写。这个的话要看表格的字段的数据类型、编码方式以及校对规则。

ci（大小写不敏感），cs（大小写敏感），_bin（二元，即比较是基于字符编码的值而与language无关，区分大小写）

B：sql中的关键字，比如：create,insert等，不区分大小写。但是大家习惯上把关键字都“大写”。

（2）命名时：尽量使用26个英文字母大小写，数字0-9，下划线，不要使用其他符号

（3）建议不要使用mysql的关键字等来作为表名、字段名、数据库名等，如果不小心使用，请在SQL语句中使用`（飘号）引起来

（4）数据库和表名、字段名等对象名中间不要包含空格

（5）同一个mysql软件中，数据库不能同名，同一个库中，表不能重名，同一个表中，字段不能重名

3.3 SQL脚本中如何加注释

SQL脚本中如何加注释

- 单行注释：#注释内容（mysql特有的）
- 单行注释：--空格注释内容　其中--后面的空格必须有
- 多行注释：/* 注释内容 */

```
create table tt(
    id int, #编号
    `name` varchar(20), -- 姓名
    gender enum('男','女')
    /*
    性别只能从男或女中选择一个，
    不能两个都选，或者选择男和女之外的
    */
);
```

3.4 mysql脚本中的标点符号

mysql脚本中标点符号的要求如下：

- 本身成对的标点符号必须成对，例如：()，''，""。
- 所有标点符号必须英文状态下半角输入方式下输入。

第4章 DQL

因为查询语句使用的非常的频繁，所以很多人把查询语句单拎出来一类，DQL（数据查询语言），DR（获取）L。

4.1 SELECT语句

SELECT语句是用于查看计算结果、或者查看从数据表中筛选出的数据的。

SELECT语句的基本语法：

```
SELECT 常量;
SELECT 表达式;
SELECT 函数;
```

例如：

```
SELECT 1;
SELECT 9/2;
SELECT NOW();
```

如果要从数据表中筛选数据，需要加FROM子句。FROM指定数据来源。字段列表筛选列。

```
SELECT 字段列表 FROM 表名称;
```

如果要从数据表中根据条件筛选数据，需要加FROM和WHERE子句。WHERE筛选行。

```
SELECT 字段列表 FROM 表名称 WHERE 条件;
```

完整的SELECT语句后面可以跟7个子句，后面会逐一讲解。

4.2 使用别名

在当前select语句中给某个字段或表达式计算结果，或表等取个临时名称，便于当前select语句的编写和理解。这个临时名称称为别名。

```
select 字段名1 as "别名1", 字段名2 as "别名2" from 表名称 as 别名;
```

- 列的别名有空格时，请加双引号。**列的别名中没有空格时，双引号可以加也可以不加。**
- **表的别名不能加双引号**，表的别名中间不能包含空格。
- as大小写都可以，as也完全可以省略。

```
mysql> select * from student;
```

```
+------+------+
| id   | name |
+------+------+
|    1 | 张三 |
|    2 | 李四 |
+------+------+
2 rows in set (0.00 sec)

mysql> select id "学号",name "姓名" from student;
+------+------+
| 学号 | 姓名 |
+------+------+
|    1 | 张三 |
|    2 | 李四 |
+------+------+
2 rows in set (0.00 sec)

mysql> select id 学号,name 姓名 from student;
+------+------+
| 学号 | 姓名 |
+------+------+
|    1 | 张三 |
|    2 | 李四 |
+------+------+
2 rows in set (0.00 sec)
```

```
#查询薪资高于15000的员工姓名和薪资
select ename,salary from t_employee where salary>15000;

mysql> select ename,salary from t_employee where salary>15000;
+--------+--------+
| ename  | salary |
+--------+--------+
| 孙洪亮 |  28000 |
| 贾宝玉 |  15700 |
| 黄冰茹 |  15678 |
| 李冰冰 |  18760 |
| 谢吉娜 |  18978 |
| 舒淇格 |  16788 |
| 章嘉怡 |  15099 |
+--------+--------+
7 rows in set (0.00 sec)

#查询薪资正好是9000的员工姓名和薪资
select ename,salary from t_employee where salary = 9000;
select ename,salary from t_employee where salary == 9000;#错误，不支持==  #注意Java中判断用==，mysql判断用=

mysql> select ename,salary from t_employee where salary == 9000;
ERROR 1064 (42000): You have an error in your SQL syntax;
 check the manual that corresponds to your MySQL server version for the right
syntax to use near '== 9000' at line 1
```

```
#查询籍贯native_place不是北京的
select * from t_employee where native_place != '北京';
select * from t_employee where native_place <> '北京';

#查询员工表中部门编号不是1
select * from t_employee where did != 1;
select * from t_employee where did <> 1;

#查询奖金比例是NULL
select * from t_employee where commission_pct = null;

mysql> select * from t_employee where commission_pct = null; #无法用=null判断
Empty set (0.00 sec)
#mysql中只要有null值参与运算和比较，结果就是null，底层就是0，表示条件不成立。

#查询奖金比例是NULL
select * from t_employee where commission_pct <=> null;
select * from t_employee where commission_pct is null;

#查询“李冰冰”、“周旭飞”、“李易峰”这几个员工的信息
select * from t_employee where ename in ('李冰冰','周旭飞','李易峰');

#查询部门编号为2、3的员工信息
select * from t_employee where did in(2,3);

#查询部门编号不是2、3的员工信息
select * from t_employee where did not in(2,3);

#查询薪资在[10000,15000]之间
select * from t_employee where salary between 10000 and 15000;

#查询姓名中第二个字是'冰'的员工
select * from t_employee where ename like '冰'; #这么写等价于 ename='冰'
select * from t_employee where ename like '_冰%';
#这么写匹配的是第二个字是冰，后面可能没有第三个字，或者有好几个字

update t_employee set ename = '王冰' where ename = '李冰冰';

select * from t_employee where ename like '_冰_';
#这么写匹配的是第二个字是冰，后面有第三个字，且只有三个字

#查询员工的姓名、薪资、奖金比例、实发工资
#实发工资 = 薪资 + 薪资 * 奖金比例
select ename as 姓名,
salary as 薪资,
commission_pct as 奖金比例,
salary + salary * commission_pct as 实发工资
from t_employee;

#NULL在mysql中比较和计算都有特殊性，所有的计算遇到的null都是null。
#实发工资 = 薪资 + 薪资 * 奖金比例
select ename as 姓名,
salary as 薪资,
commission_pct as 奖金比例,
salary + salary * ifnull(commission_pct,0) as 实发工资
```

```
from t_employee;
```

4.3 区间或集合范围比较运算符（掌握）

```
区间范围：between x  and  y
        not between x  and  y
集合范围：in (x,x,x)
        not  in(x,x,x)
```

```
#查询薪资在[10000,15000]
select * from t_employee where salary>=10000 && salary<=15000;
select * from t_employee where salary between 10000 and 15000;

#查询籍贯在这几个地方的
select * from t_employee where native_place in ('北京', '浙江', '江西');

#查询薪资不在[10000,15000]
select * from t_employee where salary not between 10000 and 15000;

#查询籍贯不在这几个地方的
select * from t_employee where native_place not in ('北京', '浙江', '江西');
```

4.4 模糊匹配比较运算符（掌握）

%：代表任意个字符

_：代表一个字符，如果两个下划线代表两个字符

```
#查询名字中包含'冰'字
select * from t_employee where ename like '%冰%';

#查询名字以‘雷'结尾的
select * from t_employee where ename like '%雷';

#查询名字以'李'开头
select * from t_employee where ename like '李%';

#查询名字有冰这个字，但是冰的前面只能有1个字
select * from t_employee where ename like '_冰%';
```

```
#查询当前mysql数据库的字符集情况
show variables like '%character%';
```

4.5 逻辑运算符（掌握）

```
逻辑与：&& 或 and
逻辑或：|| 或 or
逻辑非：! 或 not
```

```
#查询薪资高于15000，并且性别是男的员工
select * from t_employee where salary>15000 and gender='男';
```

```
select * from t_employee where salary>15000 && gender='男';

select * from t_employee where salary>15000 & gender='男';#错误 &按位与
select * from t_employee where (salary>15000) & (gender='男');

#查询薪资高于15000，或者did为1的员工
select  * from t_employee where salary>15000 or did = 1;
select  * from t_employee where salary>15000 || did = 1;

#查询薪资不在[15000,20000]范围的
select  * from t_employee where salary not between 15000 and 20000;
select  * from t_employee where !(salary between 15000 and 20000);
```

4.6 关于null值的问题（掌握）

```
#（1）判断时
xx is null
xx is not null
xx <=> null

#(2)计算时
ifnull(xx,代替值)  当xx是null时，用代替值计算
```

```
#查询奖金比例为null的员工
select * from t_employee where commission_pct = null;  #失败
select * from t_employee where commission_pct = NULL; #失败
select * from t_employee where commission_pct = 'NULL'; #失败

select * from t_employee where commission_pct is null;   #成功
select * from t_employee where commission_pct <=> null;  #成功  <=>安全等于
```

```
#查询员工的实发工资，实发工资 = 薪资 + 薪资 * 奖金比例
select ename , salary + salary * commission_pct "实发工资" from t_employee; #失败，
当commission_pct为null，结果都为null

select ename ,salary , commission_pct, salary + salary *
ifnull(commission_pct,0) "实发工资" from t_employee;
```

第5章 系统预定义函数

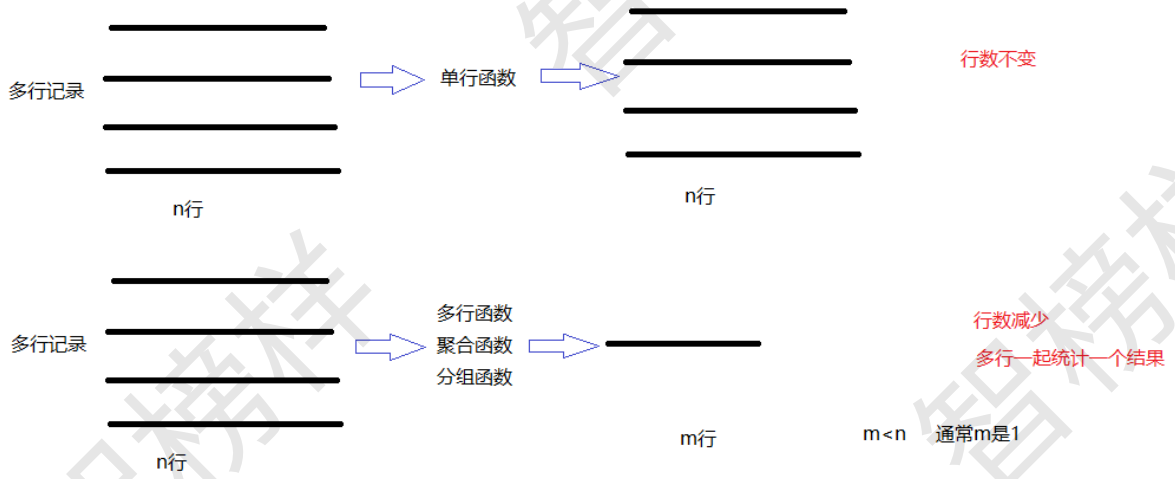
函数：代表一个独立的可复用的功能。

和Java中的方法有所不同，不同点在于：MySQL中的函数必须有返回值，参数可以有可以没有。

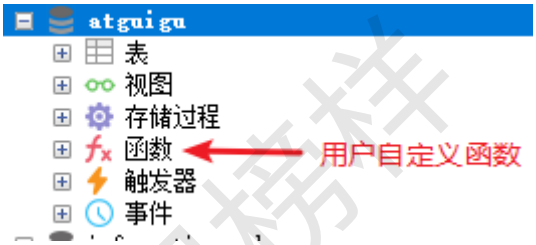
MySQL中函数分为：

（1）系统预定义函数：MySQL数据库管理软件给我提供好的函数，直接用就可以，任何数据库都可以用公共的函数。

- 分组函数：或者又称为聚合函数，多行函数，表示会对表中的多行记录一起做一个“运算”，得到一个结果。
  - 求平均值的avg，求最大值的max，求最小值的min，求总和sum，求个数的count等
- 单行函数：表示会对表中的每一行记录分别计算，有n行得到还是n行结果
  - 数学函数、字符串函数、日期时间函数、条件判断函数、窗口函数等



（2）用户自定义函数：由开发人员自己定义的，通过CREATE FUNCTION语句定义，是属于某个数据库的对象。



5.1 分组函数

调用完函数后，结果的行数变少了，可能得到一行，可能得到少数几行。

分组函数有合并计算过程。

常用分组函数类型

- **AVG(x)**：求平均值
- **SUM(x)**：求总和
- **MAX(x)**：求最大值
- **MIN(x)**：求最小值
- **COUNT(x)**：统计记录数
- ....

```
#演示分组函数，聚合函数，多行函数
#统计t_employee表的员工的数量
SELECT COUNT(*) FROM t_employee;
SELECT COUNT(1) FROM t_employee;
SELECT COUNT(eid) FROM t_employee;
SELECT COUNT(commission_pct) FROM t_employee;

/*
count(*)或count(常量值)：都是统计实际的行数。
count(字段/表达式)：只统计“字段/表达式”部分非NULL值的行数。
```

```
*/

#找出t_employee表中最高的薪资值
SELECT MAX(salary) FROM t_employee;

#找出t_employee表中最低的薪资值
SELECT MIN(salary) FROM t_employee;

#统计t_employee表中平均薪资值
SELECT AVG(salary) FROM t_employee;

#统计所有人的薪资总和，财务想看一下，一个月要准备多少钱发工资
SELECT SUM(salary) FROM t_employee; #没有考虑奖金
SELECT SUM(salary+salary*IFNULL(commission_pct,0)) FROM t_employee;

#找出年龄最小、最大的员工的出生日期
SELECT MAX(birthday),MIN(birthday) FROM t_employee;

#查询最新入职的员工的入职日期
SELECT MAX(hiredate) FROM t_employee;
```

第6章 关联查询（联合查询）

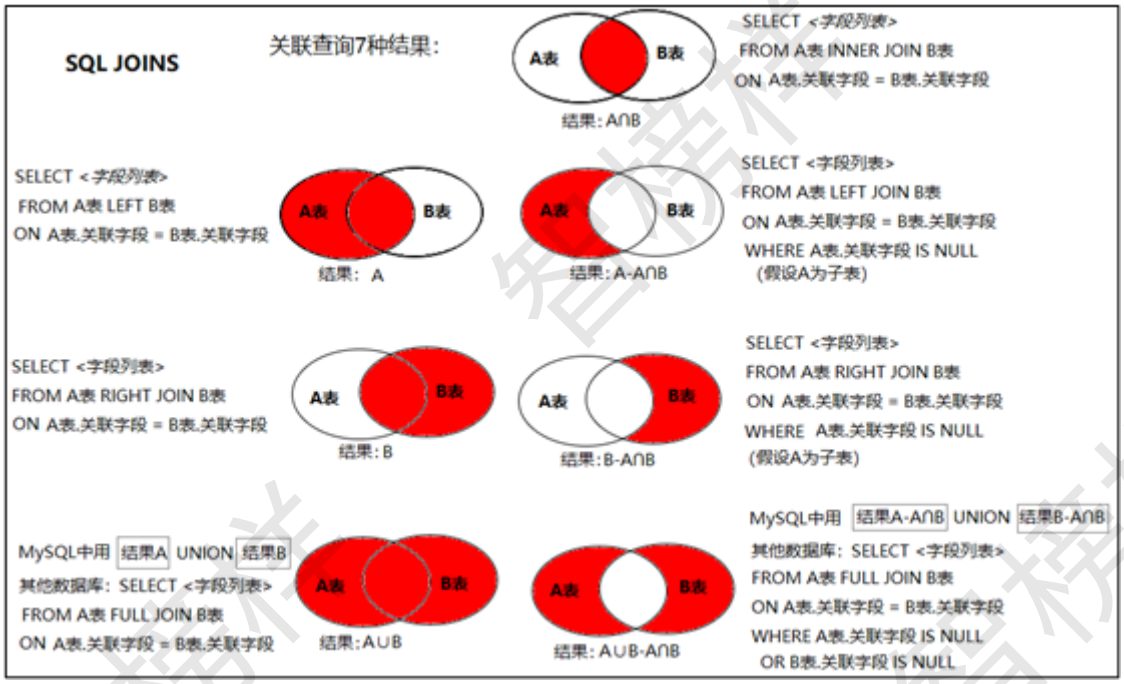
6.1 什么是关联查询

关联查询：两个或更多个表一起查询。

前提条件：这些一起查询的表之间是有关系的（一对一、一对多），它们之间一定是有关联字段，这个关联字段可能建立了外键，也可能没有建立外键。

比如：员工表和部门表，这两个表依靠“部门编号”进行关联。

6.2 关联查询结果分为几种情况



6.3 关联查询的SQL有几种情况

1、内连接：inner join ... on

结果：A表 ∩ B表

2、左连接：A left join B on

（2）A表全部

（3）A表- A∩B

3、右连接：A right join B on

（4）B表全部

（5）B表-A∩B

4、全外连接：full outer join ... on，但是mysql不支持这个关键字，mysql使用union（合并）结果的方式代替

（6）A表∪B表：　（2） A表结果 union （4）B表的结果

（7）A∪B - A∩B　（3）A表- A∩B结果 union （5）B表-A∩B结果

A表

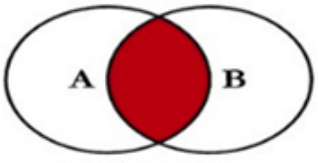
员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

A∩B结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1001	张三	15000	D1001	技术部	负责技术研发工作
S1002	李四	13000	D1001	技术部	负责技术研发工作
S1003	如意	14000	D1002	人事部	负责人事管理工作
S1004	吉祥	15000	D1003	市场部	负责市场推广工作



A∩B

A表 INNER JION B表

A表

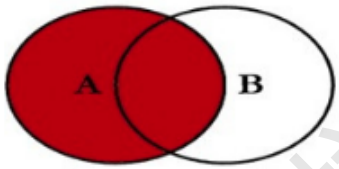
员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

A结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1001	张三	15000	D1001	技术部	负责技术研发工作
S1002	李四	13000	D1001	技术部	负责技术研发工作
S1003	如意	14000	D1002	人事部	负责人事管理工作
S1004	吉祥	15000	D1003	市场部	负责市场推广工作
S1005	王五	16000	NULL	NULL	NULL



A表 LEFT JOIN B表

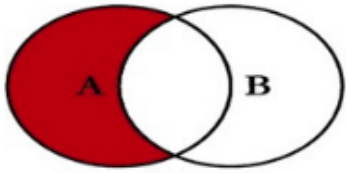
A表

员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

A表（员工表是子表）



A表 LEFT JOIN B表

WHERE A表.部门编号 IS NULL

A-(A∩B)结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1005	王五	16000	NULL	NULL	NULL

A表

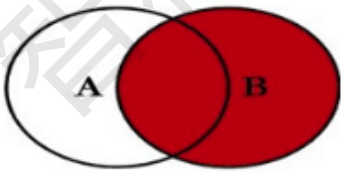
员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

B结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1001	张三	15000	D1001	技术部	负责技术研发工作
S1002	李四	13000	D1001	技术部	负责技术研发工作
S1003	如意	14000	D1002	人事部	负责人事管理工作
S1004	吉祥	15000	D1003	市场部	负责市场推广工作
NULL	NULL	NULL	D1004	测试部	负责测试检查工作



A表 RIGHT JOIN B表

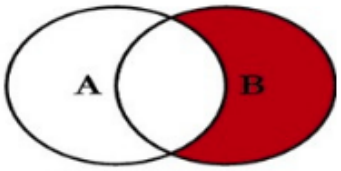
A表

员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

A表（员工表是子表）



A表 RIGHT JOIN B表
WHERE A表.部门编号 IS NULL

B表 - （A表∩B表）结果

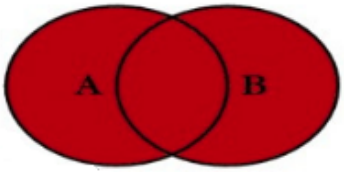
员工编号	姓名	薪资	部门编号	部门名称	部门职责
NULL	NULL	NULL	D1004	测试部	负责测试检查工作

A表

员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作



查询A结果的SQL
UNION
查询B结果的SQL

A∪B结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1001	张三	15000	D1001	技术部	负责技术研发工作
S1002	李四	13000	D1001	技术部	负责技术研发工作
S1003	如意	14000	D1002	人事部	负责人事管理工作
S1004	吉祥	15000	D1003	市场部	负责市场推广工作
S1005	王五	16000	NULL	NULL	NULL
NULL	NULL	NULL	D1004	测试部	负责测试检查工作

A表

员工编号	姓名	薪资	部门编号
S1001	张三	15000	D1001
S1002	李四	13000	D1001
S1003	如意	14000	D1002
S1004	吉祥	15000	D1003
S1005	王五	16000	NULL

B表

部门编号	部门名称	部门职责
D1001	技术部	负责技术研发工作
D1002	人事部	负责人事管理工作
D1003	市场部	负责市场推广工作
D1004	测试部	负责测试检查工作

查询A-A∩B的SQL
UNION
查询B-A∩B的SQL

（A∪B） - （A∩B）结果

员工编号	姓名	薪资	部门编号	部门名称	部门职责
S1005	王五	16000	NULL	NULL	NULL
NULL	NULL	NULL	D1004	测试部	负责测试检查工作

```
/*
（1）凡是联合查询的两个表，必须有“关联字段”，
关联字段是逻辑意义一样，数据类型一样，名字可以一样也可以不一样的两个字段。
比如：t_employee （A表）中did和t_department（B表）中的did。

发现关联字段其实就是“可以”建外键的字段。当然联合查询不要求一定建外键。

（2）联合查询必须写关联条件，关联条件的个数 = n - 1.
n是联合查询的表的数量。
```

```
如果2个表一起联合查询，关联条件数量是1，
如果3个表一起联合查询，关联条件数量是2，
如果4个表一起联合查询，关联条件数量是3，
。。。。
否则就会出现笛卡尔积现象，这是应该避免的。

（3）关联条件可以用on子句编写，也可以写到where中。
但是建议用on单独编写，这样呢，可读性更好。

每一个join后面都要加on子句
A inner|left|right join  B on 条件
A inner|left|right join  B on 条件 inner|left|right jon C on 条件
```

1、内连接

```
#演示内连接，结果是A∩B
/*
观察数据：
t_employee  看成A表
t_department 看成B表
此时t_employee （A表）中有 李红和周洲的did是NULL，没有对应部门，
    t_department（B表）中有 测试部，在员工表中找不到对应记录的。
*/

#查询所有员工的姓名，部门编号，部门名称
#如果员工没有部门的，不要
#如果部门没有员工的，不要
/*
员工的姓名在t_employee （A表）中
部门的编号，在t_employee （A表）和t_department（B表）都有
部门名称在t_department（B表）中
所以需要联合两个表一起查询。
*/
SELECT ename,did,dname
FROM t_employee INNER JOIN t_department;
#错误Column 'did' in field list is ambiguous
#因为did在两个表中都有，名字相同，它不知道取哪个表中字段了
#有同学说，它俩都是部门编号，随便取一个不就可以吗？
#mysql不这么认为，有可能存在两个表都有did，但是did的意义不同的情况。
#为了避免这种情况，需要在编写sql的时候，明确指出是用哪个表的did

SELECT ename,t_department.did,dname
FROM t_employee INNER JOIN t_department;
#语法对，结果不太对
#结果出现“笛卡尔积”现象， A表记录 * B表记录
/*
（1）凡是联合查询的两个表，必须有“关联字段”，
关联字段是逻辑意义一样，数据类型一样，名字可以一样也可以不一样的两个字段。
比如：t_employee （A表）中did和t_department（B表）中的did。

发现关联字段其实就是可以建外键的字段。当然联合查询不要求一定建外键。

（2）联合查询必须写关联条件，关联条件的个数 = n - 1.
```

```
n是联合查询的表的数量。
如果2个表一起联合查询，关联条件数量是1，
如果3个表一起联合查询，关联条件数量是2，
如果4个表一起联合查询，关联条件数量是3，
。。。。
否则就会出现笛卡尔积现象，这是应该避免的。

（3）关联条件可以用on子句编写，也可以写到where中。
但是建议用on单独编写，这样呢，可读性更好。

每一个join后面都要加on子句
A inner|left|right join  B on 条件
A inner|left|right join  B on 条件 inner|left|right jon C on 条件
*/


SELECT ename,t_department.did,dname
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;

SELECT *
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;


#查询部门编号为1的女员工的姓名、部门编号、部门名称、薪资等情况
SELECT ename,gender,t_department.did,dname,salary
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did
WHERE t_department.did = 1 AND gender = '女';

#查询部门编号为1的员工姓名、部门编号、部门名称、薪资、职位编号、职位名称等情况
SELECT ename,gender,t_department.did,dname,salary,job_id,jname
FROM t_employee INNER JOIN t_department ON t_employee.did = t_department.did
 INNER JOIN t_job ON t_employee.`job_id` = t_job.`jid`
WHERE t_department.did = 1;
```

2、左连接

```
#演示左连接
/*
(2)A
（3）A-A∩B
*/
/*
观察数据：
t_employee  看成A表
t_department 看成B表
此时t_employee （A表）中有 李红和周洲的did是NULL，没有对应部门，
    t_department（B表）中有 测试部，在员工表中找不到对应记录的。
*/
#查询所有员工，包括没有指定部门的员工，他们的姓名、薪资、部门编号、部门名称
SELECT ename,salary,t_department.did,dname
FROM t_employee LEFT JOIN t_department
```

```
ON t_employee.did = t_department.did;
#查询的是A结果  A left join B

#查询没有部门的员工信息
SELECT ename,salary,t_department.did,dname
FROM t_employee LEFT JOIN t_department
ON t_employee.did = t_department.did
WHERE t_employee.did IS NULL;
#查询的结果是A - A∩B
#此时的where条件，建议写子表的关联字段is null，这样更准确一点。
#如果要建外键，它们肯定有子表和父表的角色，写子表的关联字段is null
#因为父表中这个字段一般是主键，不会为null。
```

3、右连接

```
/*
右连接
（4）B
（5）B - A∩B
*/
#演示右连接
/*
观察数据：
t_employee  看成A表
t_department 看成B表
此时t_employee （A表）中有 李红和周洲的did是NULL，没有对应部门，
    t_department（B表）中有 测试部，在员工表中找不到对应记录的。
*/
#查询所有部门，包括没有对应员工的部门，他们的姓名、薪资、部门编号、部门名称
SELECT ename,salary,t_department.did,dname
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did;
#查询的是B结果  A RIGHT join B

#查询没有员工部门的信息
SELECT ename,salary,t_department.did,dname
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
WHERE t_employee.did IS NULL;
#查询的结果是B - A∩B
#此时的where条件，建议写子表的关联字段is null，这样更准确一点。
#如果要建外键，它们肯定有子表和父表的角色，写子表的关联字段is null
#因为父表中这个字段一般是主键，不会为null。


#查询所有员工，包括没有指定部门的员工，他们的姓名、薪资、部门编号、部门名称
SELECT ename,salary,t_department.did,dname
FROM t_department RIGHT JOIN t_employee
ON t_employee.did = t_department.did;
#查询的是B结果  A RIGHT join B

#查询没有部门的员工信息
SELECT ename,salary,t_department.did,dname
```

```
FROM t_department RIGHT JOIN t_employee
ON t_employee.did = t_department.did
WHERE t_employee.did IS NULL;
#查询的结果是B - A∩B  A right join B
#此时的where条件，建议写子表的关联字段is null，这样更准确一点。
#如果要建外键，它们肯定有子表和父表的角色，写子表的关联字段is null
#因为父表中这个字段一般是主键，不会为null。
```

4、union

```
/*
union实现
（6）A∪B
（7）A∪B - A∩B
    A-A∩B  ∪  B-A∩B
*/
#演示用union合并两个查询结果实现A∪B 和A∪B - A∩B
/*
union合并时要注意：
（1）两个表要查询的结果字段是一样的
（2）UNION ALL表示直接合并结果，如果有重复的记录一并显示
   ALL去掉表示合并结果时，如果有重复记录，去掉。
（3）要实现A∪B的结果，那么必须是合并 查询是A表结果和查询是B表结果的select语句。
同样要实现A∪B - A∩B的结果，那么必须是合并查询是A-A∩B结果和查询是B-A∩B的select语句。
*/

#查询所有员工和所有部门，包括没有指定部门的员工和没有分配员工的部门。
SELECT *
FROM t_employee LEFT JOIN t_department
ON t_employee.did = t_department.did

UNION

SELECT *
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did;


#以下union会报错
SELECT * FROM t_employee
UNION
SELECT * FROM t_department;
/*
错误代码： 1222
The used SELECT statements have a different number of columns
两个Select语句的列数是不同的。

column：列，表中的字段。
columns：很多的字段，即字段列表
select 字段列表 from 表名称;
*/


#联合 查询结果是A表的select 和查询结果是A∩B的select语句，是得不到A∪B
SELECT *
```

```
FROM t_employee LEFT JOIN t_department
ON t_employee.did = t_department.did

UNION

SELECT *
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;

#查询那些没有分配部门的员工和没有指定员工的部门，即A表和B表在对方那里找不到对应记录的数据。
SELECT *
FROM t_employee LEFT JOIN t_department
ON t_employee.did = t_department.did
WHERE t_employee.did IS NULL

UNION

SELECT *
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
WHERE t_employee.did IS NULL;
```

6.4 联合查询字段列表问题

```
#查询字段的问题
#查询每一个员工及其所在部门的信息
#要求：显示员工的编号，姓名，部门编号，部门名称
SELECT eid,ename,did,dname
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;
/*
错误代码： 1052
Column 'did' in field list is ambiguous（模糊不清的；引起歧义的）
*/

SELECT eid,ename,t_employee.did,dname
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;


#查询每一个员工及其所在部门的信息
#要求，显示员工的编号，姓名，部门表的所有字段
SELECT eid,ename,t_department.*
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;
```

第7章 select的7大子句

7.1 7大子句顺序

（1）from：从哪些表中筛选

（2）on：关联多表查询时，去除笛卡尔积

（3）where：从表中筛选的条件

（4）group by：分组依据

（5）having：在统计结果中再次筛选（with rollup）

（6）order by：排序

（7）limit：分页

必须按照（1）-（7）的顺序编写子句。

7.2 演示

7.2.1 from子句

```
#1、from子句
SELECT *
FROM t_employee; #表示从某个表中筛选数据
```

7.2.2 on子句

```
#2、on子句
/*
（1）on必须配合join使用
（2）on后面只写关联条件
所谓关联条件是两个表的关联字段的关系
（3）有n张表关联，就有n-1个关联条件
两张表关联，就有1个关联条件
三张表关联，就有2个关联条件
*/
SELECT *
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did;  #1个关联条件

#查询员工的编号，姓名，职位编号，职位名称，部门编号，部门名称
#需要t_employee员工表，t_department部门表，t_job职位表
SELECT eid,ename,t_job.job_id,t_job.job_name,
`t_department`.`did`,`t_department`.`dname`
FROM t_employee INNER JOIN t_department INNER JOIN t_job
ON t_employee.did = t_department.did AND t_employee.job_id = t_job.job_id;
```

7.2.3 where子句

```
#3、where子句，在查询结果中筛选
#查询女员工的信息，以及女员工的部门信息
SELECT *
FROM t_employee INNER JOIN t_department
ON t_employee.did = t_department.did
WHERE gender = '女';
```

7.2.4 group by子句

```
#4、group by分组
#查询所有员工的平均薪资
SELECT AVG(salary) FROM t_employee;

#查询每一个部门的平均薪资
SELECT did,ROUND(AVG(salary),2 )
FROM t_employee
GROUP BY did;

#查询每一个部门的平均薪资，显示部门编号，部门的名称，该部门的平均薪资
SELECT t_department.did,dname,ROUND(AVG(salary),2 )
FROM t_department LEFT JOIN t_employee
ON t_department.did = t_employee.did
GROUP BY t_department.did;

#查询每一个部门的平均薪资，显示部门编号，部门的名称，该部门的平均薪资
#要求，如果没有员工的部门，平均薪资不显示null，显示0
SELECT t_department.did,dname,IFNULL(ROUND(AVG(salary),2),0)
FROM t_department LEFT JOIN t_employee
ON t_department.did = t_employee.did
GROUP BY t_department.did;

#查询每一个部门的女员工的平均薪资，显示部门编号，部门的名称，该部门的平均薪资
#要求，如果没有员工的部门，平均薪资不显示null，显示0
SELECT t_department.did,dname,IFNULL(ROUND(AVG(salary),2),0)
FROM t_department LEFT JOIN t_employee
ON t_department.did = t_employee.did
WHERE gender = '女'
GROUP BY t_department.did;
```

问题1：合计，WITH ROLLUP，加在group by后面

```
#问题1：合计，WITH ROLLUP，加在group by后面
#按照部门统计人数
SELECT did, COUNT(*) FROM t_employee GROUP BY did;
#按照部门统计人数，并合计总数
SELECT did, COUNT(*) FROM t_employee GROUP BY did WITH ROLLUP;
SELECT IFNULL(did,'合计'), COUNT(*) FROM t_employee GROUP BY did WITH ROLLUP;
SELECT IFNULL(did,'合计') AS "部门编号" , COUNT(*)  AS "人数" FROM t_employee GROUP
BY did WITH ROLLUP;
```

问题2：是否可以按照多个字段分组统计

```
#问题2：是否可以按照多个字段分组统计
#按照不同的部门，不同的职位，分别统计男和女的员工人数
SELECT did, job_id, gender, COUNT(*)
FROM t_employee
GROUP BY did, job_id, gender;
```

问题4：分组统计时，select后面字段列表的问题

```
#问题4：分组统计时，select后面字段列表的问题
SELECT eid,ename, did, COUNT(*) FROM t_employee;
#eid,ename, did此时和count(*)，不应该出现在select后面

SELECT eid,ename, did, COUNT(*) FROM t_employee GROUP BY did;
#eid,ename此时和count(*)，不应该出现在select后面

SELECT  did, COUNT(*) FROM t_employee GROUP BY did;
#分组统计时，select后面只写和分组统计有关的字段，其他无关字段不要出现，否则会引起歧义
```

7.2.5 having子句

```
#5、having
/*
having子句也写条件
where的条件是针对原表中的记录的筛选。where后面不能出现分组函数。
having子句是对统计结果（分组函数计算后）的筛选。having可以加分组函数。
*/
#查询每一个部门的女员工的平均薪资，显示部门编号，部门的名称，该部门的平均薪资
#要求，如果没有员工的部门，平均薪资不显示null，显示0
#最后只显示平均薪资高于12000的部门信息
SELECT t_department.did,dname,IFNULL(ROUND(AVG(salary),2),0)
FROM t_department LEFT JOIN t_employee
ON t_department.did = t_employee.did
WHERE gender = '女'
GROUP BY t_department.did
HAVING IFNULL(ROUND(AVG(salary),2),0) >12000;


#查询每一个部门的男和女员工的人数
SELECT did,gender,COUNT(*)
FROM t_employee
GROUP BY did,gender;

#查询每一个部门的男和女员工的人数，显示部门编号，部门的名称，性别，人数
SELECT t_department.did,dname,gender,COUNT(eid)
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
GROUP BY t_department.did,gender;

#查询每一个部门薪资超过10000的男和女员工的人数，显示部门编号，部门的名称，性别，人数
#只显示人数低于3人
SELECT t_department.did,dname,gender,COUNT(eid)
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
WHERE salary > 10000
GROUP BY t_department.did,gender
HAVING COUNT(eid) < 3;
```

7.2.6 order by子句

```
#6、排序 order by
/*
升序和降序，默认是升序
asc代表升序
desc 代表降序
*/
#查询员工信息，按照薪资从高到低
SELECT * FROM t_employee
ORDER BY salary DESC;

#查询每一个部门薪资超过10000的男和女员工的人数，显示部门编号，部门的名称，性别，人数
#只显示人数低于3人，按照人数升序排列
SELECT t_department.did,dname,gender,COUNT(eid)
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
WHERE salary > 10000
GROUP BY t_department.did,gender
HAVING COUNT(eid) < 3
ORDER BY COUNT(eid);


#查询员工的薪资，按照薪资从低到高，薪资相同按照员工编号从高到低
SELECT *
FROM t_employee
ORDER BY salary ASC , eid DESC;
```

7.2.7 limit子句

```
#演示limit子句
/*
limit子句是用于分页显示结果。
limit m,n
n：表示最多该页显示几行
m：表示从第几行开始取记录，第一个行的索引是0
m = (page-1)*n  page表示第几页

每页最多显示5条，n=5
第1页，page=1，m = (1-1)*5 = 0;  limit 0,5
第2页，page=2，m = (2-1)*5 = 5;  limit 5,5
第3页，page=3，m = (3-1)*5 = 10;  limit 10,5
*/
#查询员工表的数据，分页显示，每页显示5条记录
#第1页
SELECT * FROM t_employee LIMIT 0,5;
#第2页
SELECT * FROM t_employee LIMIT 5,5;
#第3页
SELECT * FROM t_employee LIMIT 10,5;
#第4页
SELECT * FROM t_employee LIMIT 15,5;
#第5页
SELECT * FROM t_employee LIMIT 20,5;
```

```
#第6页
SELECT * FROM t_employee LIMIT 25,5;


#查询所有的男员工信息，分页显示，每页显示3条，第2页
#limit m,n  n=3,page=2,m=(page-1)*n=3
SELECT *
FROM t_employee
WHERE gender ='男'
LIMIT 3,3

#查询每一个编号为偶数的部门，显示部门编号，名称，员工数量，
#只显示员工数量>=2的结果，按照员工数量升序排列，
#每页显示2条，显示第1页
SELECT t_department.did,dname,COUNT(eid)
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
WHERE t_department.did%2=0
GROUP BY t_department.did
HAVING COUNT(eid)>=2
ORDER BY COUNT(eid)
LIMIT 0,2;
```

第8章 子查询

8.1 SELECT的SELECT中嵌套子查询

```
/*
子查询：嵌套在另一个SQL语句中的查询。
SELECT语句可以嵌套在另一个SELECT中，UPDATE，DELETE，INSERT，CREATE语句等。

(1)SELECT的SELECT中嵌套子查询
*/

#（1）在“t_employee”表中查询每个人薪资和公司平均薪资的差值，
#并显示员工薪资和公司平均薪资相差5000元以上的记录。
SELECT ename AS "姓名",
    salary AS "薪资",
 ROUND((SELECT AVG(salary) FROM t_employee),2) AS "全公司平均薪资",
 ROUND(salary-(SELECT AVG(salary) FROM t_employee),2) AS "差值"
FROM t_employee
WHERE ABS(ROUND(salary-(SELECT AVG(salary) FROM t_employee),2))>5000;

#（2）在“t_employee”表中查询每个部门平均薪资和公司平均薪资的差值。
SELECT did,AVG(salary),
AVG(salary)-(SELECT AVG(salary) FROM t_employee)
FROM t_employee
GROUP BY did;
```

8.2 SELECT的WHERE或HAVING中嵌套子查询

当子查询结果作为外层另一个SQL的过滤条件，通常把子查询嵌入到WHERE或HAVING中。根据子查询结果的情况，分为如下三种情况。

- 当子查询的结果是单列单个值，那么可以直接使用比较运算符，如“<”、“<=”、“>”、“>=”、“=”、“!=”等与子查询结果进行比较。
- 当子查询的结果是单列多个值，那么可以使用比较运算符IN或NOT IN进行比较。
- 当子查询的结果是单列多个值，还可以使用比较运算符, 如“<”、“<=”、“>”、“>=”、“=”、“!=”等搭配ANY、SOME、ALL等关键字与查询结果进行比较。

```
/*
子查询嵌套在where后面。
在where或having后面的子查询结果是：
（1）单个值，那么可以用=，>,<,>=,<=,!=这样的运算符和子查询的结果做比较
（2）多个值，那么需要用in,not in, >all,>any....形式做比较
 如“<”、“<=”、“>”、“>=”、“=”、“!=”等搭配ANY、SOME、ALL等关键字与查询结果进行比较

*/
#（1）在“t_employee”表中查询薪资最高的员工姓名（ename）和薪资（salary）。
#SELECT ename,MAX(salary) FROM t_employee;#错误
#取表中第一行员工的姓名和全公司最高的薪资值一起显示。

SELECT ename,salary
FROM t_employee
WHERE salary = (SELECT MAX(salary) FROM t_employee);

#（2）在“t_employee”表中查询比全公司平均薪资高的男员工姓名和薪资。
SELECT ename,salary
FROM t_employee
WHERE salary > (SELECT AVG(salary) FROM t_employee) AND gender = '男';

#（3）在“t_employee”表中查询和“白露”，“谢吉娜”同一部门的员工姓名和电话。
SELECT ename,tel,did
FROM t_employee
WHERE did IN(SELECT did FROM t_employee WHERE ename='白露' || ename='谢吉娜');

SELECT ename,tel,did
FROM t_employee
WHERE did =ANY(SELECT did FROM t_employee WHERE ename='白露' || ename='谢吉娜');


#（4）在“t_employee”表中查询薪资比“白露”，“李诗雨”，“黄冰茹”三个人的薪资都要高的员工姓名和薪资。
SELECT ename,salary
FROM t_employee
WHERE salary >ALL(SELECT salary FROM t_employee WHERE ename IN('白露','李诗雨','黄冰茹'));


#（5）查询“t_employee”和“t_department”表，按部门统计平均工资，
#显示部门平均工资比全公司的总平均工资高的部门编号、部门名称、部门平均薪资，
#并按照部门平均薪资升序排列。
SELECT t_department.did,dname,AVG(salary)
FROM t_employee RIGHT JOIN t_department
ON t_employee.did = t_department.did
GROUP BY t_department.did
```

```
HAVING AVG(salary) >(SELECT AVG(salary) FROM t_employee)
ORDER BY AVG(salary);
```

8.3 SELECT中的EXISTS型子查询

EXISTS型子查询也是存在外层SELECT的WHERE子句中，不过它和上面的WHERE型子查询的工作模式不相同，所以这里单独讨论它。

如果EXISTS关键字后面的参数是一个任意的子查询，系统将对子查询进行运算以判断它是否返回行，如果至少返回一行，那么EXISTS的结果为true，此时外层查询语句将进行查询；如果子查询没有返回任何行，那么EXISTS的结果为false，此时外层查询语句不进行查询。EXISTS和NOT EXISTS的结果只取决于是否返回行，而不取决于这些行的内容，所以这个子查询输入列表通常是无关紧要的。

如果EXISTS关键字后面的参数是一个关联子查询，即子查询的WHERE条件中包含与外层查询表的关联条件，那么此时将对外层查询表做循环，即在筛选外层查询表的每一条记录时，都看这条记录是否满足子查询的条件，如果满足就再用外层查询的其他WHERE条件对该记录进行筛选，否则就丢弃这行记录。

```
#exist型子查询
/*
（1）exists()中的子查询和外面的查询没有联合的情况下，
如果exists()中的子查询没有返回任何行，那么外面的子查询就不查了。
（2）exists()中的子查询与外面的查询有联合工作的情况下，
循环进行把外面查询表的每一行记录的值，代入()中子查询，如果可以查到结果，
就留下外面查询的这条记录，否则就舍去。
*/

#（1）查询“t_employee”表中是否存在部门编号为NULL的员工，
#如果存在，查询“t_department”表的部门编号、部门名称。
SELECT * FROM t_department
WHERE EXISTS(SELECT * FROM t_employee  WHERE did IS NULL);

#（2）查询“t_department”表是否存在与“t_employee”表相同部门编号的记录，
#如果存在，查询这些部门的编号和名称。
SELECT * FROM t_department
WHERE EXISTS(SELECT * FROM t_employee WHERE t_employee.did = t_department.did);

#查询结果等价于下面的sql
SELECT DISTINCT t_department.*
FROM t_department INNER JOIN t_employee
ON t_department.did = t_employee.did;
```

8.4 SELECT的FROM中嵌套子查询

当子查询结果是多列的结果时，通常将子查询放到FROM后面，然后采用给子查询结果取别名的方式，把子查询结果当成一张“动态生成的临时表”使用。

```
#子查询嵌套在from后面
/*
当一个查询要基于另一个查询结果来筛选的时候，
另一个查询还是多行多列的结果，那么就可以把这个查询结果当成一张临时表，
放在from后面进行再次筛选。

*/
```

```
#（1）在“t_employee”表中，查询每个部门的平均薪资，
#然后与“t_department”表联合查询
#所有部门的部门编号、部门名称、部门平均薪资。

SELECT did,AVG(salary) FROM t_employee GROUP BY did;

+------+-------------+
| did  | AVG(salary) |
+------+-------------+
|    1 |  11479.3125 |
|    2 |       13978 |
|    3 |    37858.25 |
|    4 |       12332 |
|    5 |       11725 |
+------+-------------+
5 ROWS IN SET (0.00 sec)

#用上面的查询结果，当成一张临时表，与t_department部门表做联合查询
#要给这样的子查询取别名的方式来当临时表用，不取别名是不可以的。
#而且此时的别名不能加""
#字段的别名可以加""，表的别名不能加""

SELECT t_department.did ,dname,AVG(salary)
FROM t_department LEFT JOIN (SELECT did,AVG(salary) FROM t_employee GROUP BY
did) temp
ON t_department.did = temp.did;
#错误，from后面的t_department和temp表都没有salary字段，
#SELECT t_department.did ,dname,AVG(salary)出现AVG(salary)是错误的

SELECT t_department.did ,dname,pingjun
FROM t_department LEFT JOIN (SELECT did,AVG(salary) AS pingjun FROM t_employee
GROUP BY did) temp
ON t_department.did = temp.did;


#（2）在“t_employee”表中查询每个部门中薪资排名前2的员工姓名、部门编号和薪资。
SELECT * FROM (
SELECT ename,did,salary,
DENSE_RANK() over (PARTITION BY did ORDER BY salary DESC) AS paiming
FROM t_employee) temp
WHERE temp.paiming <=2;
```

SQL示例演示

```
#演示和数据表相关的DDL语句
#为了方便接下来的演示，最好在前面确定针对哪个数据库的表格演示
#使用数据库
use test;

#查看当前登录用户在本库下能够看到的所有表格
show tables;

#如果前面没有use语句，或者在当前use语句下，要查看另一个数据库的表格。
show tables from 数据库名;
```

```
#例如：查看当前数据库的表格
show tables;
#例如：在当前use atguigu;下面，查看mysql库的表格
show tables from mysql;

#创建表格
create table 表名称(
    字段名1 数据类型1,
    字段名2 数据类型2  #如果后面没有其他字段或约束的定义，后面就不用加,
);

#例如：创建一个teacher表
/*
包含编号、姓名、性别、出生日期、薪资、电话号码
*/
create table teacher(
    id int,
    name varchar(20),
    gender enum('男','女'),
    birthday date,
    salary double,
    tel varchar(11)
);

#查看表结构
desc 表名称;
describe 表名称;

#例如：查看teacher表的结构
desc teacher;
describe teacher;

mysql> describe teacher;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| id       | int             | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| gender   | enum('男','女') | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| salary   | double          | YES  |     | NULL    |       |
| tel      | varchar(11)     | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

#查看表格的详细定义
show create table 表名称;

#例如：查看teacher表的定义语句
show create table teacher;

mysql> show create table teacher\G
*************************** 1. row ***************************
       Table: teacher
Create Table: CREATE TABLE `teacher` (
```

```
  `id` int DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL,
  `gender` enum('男','女') DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `salary` double DEFAULT NULL,
  `tel` varchar(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.00 sec)

#修改表结构
#增加一个字段
alter table 表名称 add column 字段名 数据类型;
#column表示列，字段，可以省略

#例如：给teacher表增加一个address varchar(100)字段
alter table teacher add column address varchar(100);

mysql> desc teacher;
+---------+-----------------+------+-----+---------+-------+
| Field   | Type            | Null | Key | Default | Extra |
+---------+-----------------+------+-----+---------+-------+
| id      | int             | YES  |     | NULL    |       |
| name    | varchar(20)     | YES  |     | NULL    |       |
| gender  | enum('男','女') | YES  |     | NULL    |       |
| birthday| date            | YES  |     | NULL    |       |
| salary  | double          | YES  |     | NULL    |       |
| tel     | varchar(11)     | YES  |     | NULL    |       |
| address | varchar(100)    | YES  |     | NULL    |       |
+---------+-----------------+------+-----+---------+-------+
7 rows in set (0.00 sec)


#在某个字段后面增加一个字段
alter table 表名称 add column 字段名 数据类型 after 另一个字段;
#column表示列，字段，可以省略

#例如：给teacher表增加一个cardid char(18)字段，增加到name后面
alter table teacher add column cardid char(18) after name;

mysql> desc teacher;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| id       | int             | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| cardid   | char(18)        | YES  |     | NULL    |       |
| gender   | enum('男','女') | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| salary   | double          | YES  |     | NULL    |       |
| tel      | varchar(11)     | YES  |     | NULL    |       |
| address  | varchar(100)    | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

#增加一个字段，称为第一个字段
```

```
alter table 表名称 add column 字段名 数据类型 first;
#column表示列，字段，可以省略

#例如：给teacher表增加一个age int字段，增加到id前面
alter table teacher add column age int first;

mysql> desc teacher;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| age      | int             | YES  |     | NULL    |       |
| id       | int             | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| cardid   | char(18)        | YES  |     | NULL    |       |
| gender   | enum('男','女') | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| salary   | double          | YES  |     | NULL    |       |
| tel      | varchar(11)     | YES  |     | NULL    |       |
| address  | varchar(100)    | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
9 rows in set (0.01 sec)


#删除字段
alter table 表名称 drop column 字段名;
#column可以省略

#例如：删除teacher表的age字段
alter table teacher drop column age;

mysql> desc teacher;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| id       | int             | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| cardid   | char(18)        | YES  |     | NULL    |       |
| gender   | enum('男','女') | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| salary   | double          | YES  |     | NULL    |       |
| tel      | varchar(11)     | YES  |     | NULL    |       |
| address  | varchar(100)    | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

#修改字段的数据类型
alter table 表名称 modify column 字段名 新的数据类型;

#例如：修改teacher表的salary字段，数据类型修改为double(10,2)
alter table teacher modify column salary double(10,2);

mysql> desc teacher;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
```

```
| id       | int             | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| cardid   | char(18)        | YES  |     | NULL    |       |
| gender   | enum('男','女') | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| salary   | double(10,2)    | YES  |     | NULL    |       |
| tel      | varchar(11)     | YES  |     | NULL    |       |
| address  | varchar(100)    | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

#修改字段的名称
alter table 表名称 change column 旧字段名 新的字段名 数据类型;

#例如：修改teacher表的tel字段，字段名修改为telphone
alter table teacher change column tel telphone char(18);

mysql> desc teacher;
+---------+-----------------+------+-----+---------+-------+
| Field   | Type            | Null | Key | Default | Extra |
+---------+-----------------+------+-----+---------+-------+
| id      | int             | YES  |     | NULL    |       |
| name    | varchar(20)     | YES  |     | NULL    |       |
| cardid  | char(18)        | YES  |     | NULL    |       |
| gender  | enum('男','女') | YES  |     | NULL    |       |
| birthday| date            | YES  |     | NULL    |       |
| salary  | double(10,2)    | YES  |     | NULL    |       |
| telphone| char(18)        | YES  |     | NULL    |       |
| address | varchar(100)    | YES  |     | NULL    |       |
+---------+-----------------+------+-----+---------+-------+
8 rows in set (0.01 sec)

#修改字段的顺序
alter table 表名称 modify column 字段名 数据类型 after 另一个字段;
alter table 表名称 modify column 字段名 数据类型 first;

#例如，把teacher表的salary调整到telphone后面
alter table teacher modify column salary double(10,2) after telphone;

mysql> desc teacher;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| id       | int             | YES  |     | NULL    |       |
| name     | varchar(20)     | YES  |     | NULL    |       |
| cardid   | char(18)        | YES  |     | NULL    |       |
| gender   | enum('男','女') | YES  |     | NULL    |       |
| birthday | date            | YES  |     | NULL    |       |
| telphone | char(18)        | YES  |     | NULL    |       |
| salary   | double(10,2)    | YES  |     | NULL    |       |
| address  | varchar(100)    | YES  |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

```
#修改表名称
rename table 旧表名称 to 新表名称;
alter table 表名称 rename 新表名称;


#把teacher表重命名为jiaoshi
rename table teacher to jiaoshi;

#把jiaoshi表重命名为teacher
alter table jiaoshi rename teacher;


#删除表结构（数据一并删除）
drop table 表名称;

#删除teacher表格
drop table teacher;
```

第9章 DML

9.1 添加语句

（1）添加一条记录到某个表中

```
insert into 表名称 values(值列表); #值列表中的值的顺序、类型、个数必须与表结构一一对应
```

```
mysql> desc teacher;
+----------+-----------------------+------+-----+---------+-------+
| Field    | Type                  | Null | Key | Default | Extra |
+----------+-----------------------+------+-----+---------+-------+
| tid      | int(11)               | YES  |     | NULL    |       |
| tname    | varchar(5)            | YES  |     | NULL    |       |
| salary   | double                | YES  |     | NULL    |       |
| weight   | double                | YES  |     | NULL    |       |
| birthday | date                  | YES  |     | NULL    |       |
| gender   | enum('男','女')       | YES  |     | NULL    |       |
| blood    | enum('A','B','AB','O') | YES  |     | NULL    |       |
| phone    | char(11)              | YES  |     | NULL    |       |
+----------+-----------------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

```
insert into teacher values(1,'张三',15000,120.5,'1990-5-
1','男','O','13789586859');
```

```
insert into teacher values(2,'李四',15000,'1990-5-1','男','O','13789586859'); #缺体重weight的值

ERROR 1136 (21S01): Column（列） count（数量） doesn't match（不匹配） value（值） count（数量） at row 1
```

(2) 添加一条记录到某个表中

```
insert into 表名称 (字段列表) values(值列表); #值列表中的值的顺序、类型、个数必须与(字段列表)一一对应
```

```
insert into teacher(tid,tname,salary,phone) values(3,'王五',16000,'15789546586');
```

(3) 添加多条记录到某个表中

```
insert into 表名称 values(值列表),(值列表),(值列表); #值列表中的值的顺序、类型、个数必须与表结构一一对应
```

```
insert into 表名称 (字段列表) values(值列表),(值列表),(值列表); #值列表中的值的顺序、类型、个数必须与(字段列表)一一对应
```

```
insert into teacher (tid,tname,salary,phone)
values(4,'赵六',16000,'15789546586'),
(5,'汪飞',18000,'15789548886'),
(6,'天琪',19000,'15909546586');
```

(4) 示例演示

```
#演示基本的，简单的DML语句
#基于tempdb数据库演示
create database tempdb;
use tempdb;

#创建teacher表
create table teacher(
    id int,
    name varchar(20),
    gender enum('m','f'),
    birthday date,
    salary double,
    tel varchar(11)
);

#查看teacher表结构
mysql> desc teacher;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| id       | int           | YES  |     | NULL    |       |
| name     | varchar(20)   | YES  |     | NULL    |       |
| gender   | enum('m','f') | YES  |     | NULL    |       |
| birthday | date          | YES  |     | NULL    |       |
| salary   | double        | YES  |     | NULL    |       |
| tel      | char(18)      | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

```
#添加数据
#（1）第一种情况，给所有字段赋值
insert into 表名称 values(值列表);
#这种情况要求(值列表)的每一个值的类型、顺序与表结构一一对应
#表中有几个字段，(值列表)必须有几个值，不能多也不能少
#值如果是字符串或日期类型，需要加单引号

#例如：添加一条记录到teacher表
insert into teacher values
(1,'张三','m','1998-7-8',15000.0,'18256953685');

#例如：添加一条记录到teacher表
insert into teacher values
(2,'李四','f','1998-7-8',15000.0); #少了电话号码

mysql> insert into teacher values
    -> (2,'李四','f','1998-7-8',15000.0);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
#(值列表)中值的数量和表结构中column列的数量不一致。

#例如：添加一条记录到teacher表
insert into teacher values
(2,'李四','f','北京宏福苑',15000.0,'18256953685'); #把生日写称为地址

mysql> insert into teacher values
    -> (2,'李四','f','北京宏福苑',15000.0,'18256953685');
ERROR 1292 (22007): Incorrect date value: '北京宏福苑' for column 'birthday' at
row 1
#日期格式不对

#（2）第二种情况，给部分字段赋值
insert into 表名称 (部分字段列表) values(值列表);
#此时(值列表)中的值的数量、格式、顺序与(部分字段列表)对应即可

#例如：添加一条记录到teacher表，只给id和name字段赋值
insert into teacher (id,name) values (2,'李四');

mysql> select * from teacher;
+------+------+--------+------------+--------+-------------+
| id   | name | gender | birthday   | salary | tel         |
+------+------+--------+------------+--------+-------------+
|    1 | 张三 | m      | 1998-07-08 |  15000 | 18256953685 |
|    2 | 李四 | NULL   | NULL       |   NULL | NULL        |
+------+------+--------+------------+--------+-------------+
2 rows in set (0.00 sec)
#没有赋值的字段都是默认值，此时默认值是NULL
#这种情况，当某个字段设置了“非空NOT NULL”约束，又没有提前指定“默认值”，
#那么在添加时没有赋值的话，会报错。明天演示非空约束。

#（3）一次添加多条记录
insert into 表名称  values(值列表1),(值列表2)...;
insert into 表名称 (部分字段列表) values(值列表),(值列表2)...;
#上面一个insert语句有几个(值列表)就表示添加几行记录。
#每一个值列表直接使用逗号分隔
```

```
#添加多条记录到teacher表
insert into teacher (id,name) values
 (3,'王五'),
 (4,'宋鑫'),
 (5,'赵志浩'),
 (6,'杨业行'),
 (7,'牛钰琪');

 #查看数据
 mysql> select * from teacher;
+------+--------+--------+------------+--------+-------------+
| id   | name   | gender | birthday   | salary | tel         |
+------+--------+--------+------------+--------+-------------+
|    1 | 张三   | m      | 1998-07-08 |  15000 | 18256953685 |
|    2 | 李四   | NULL   | NULL       |   NULL | NULL        |
|    3 | 王五   | NULL   | NULL       |   NULL | NULL        |
|    4 | 宋鑫   | NULL   | NULL       |   NULL | NULL        |
|    5 | 赵志浩 | NULL   | NULL       |   NULL | NULL        |
|    6 | 杨业行 | NULL   | NULL       |   NULL | NULL        |
|    7 | 牛钰琪 | NULL   | NULL       |   NULL | NULL        |
+------+--------+--------+------------+--------+-------------+
7 rows in set (0.00 sec)
```

9.2 修改语句

修改所有行

```
update 表名称 set 字段名 = 值, 字段名 = 值; #给所有行修改
```

```
#修改所有人的薪资，都涨了1000
update teacher set salary = salary + 1000 ;
```

修改部分行

```
update 表名称 set 字段名 = 值, 字段名 = 值 where 条件; #给满足条件的行修改
```

```
#修改天琪的薪资降低5000
update teacher set salary = salary-5000 where tname = '天琪';
```

9.3 删除

删除部分行的数据

```
delete from 表名称 where 条件;
```

```
delete from teacher where tname = '天琪';
```

删除整张表的数据，但表结构留下

```
delete from 表名称;
```

```
delete from teacher;
```

截断表，清空表中的数据，只有表结构

```
truncate 表名称;
```

```
truncate teacher;
```

truncate表和delete表的区别：

delete是一条一条删除记录的。如果在事务中，事务提交之前支持回滚。（后面会讲事务）

truncate是把整个表drop，新建一张，效率更高。就算在事务中，也无法回滚。

```
#同学问：是否可以删除salary字段的值，字段留着，值删掉
#可以实现，但是不是用delete，用update

#同学问：是否可以删除salary字段，连同字段和这个字段的数据都是删除
#可以实现，但是不是用delete，用alter table 表名称 drop column 字段名;

#同学问：只删除某个单元格的值
#可以实现，但是不是用delete，用update
```

####

###

###