

# MYSQL

## 数据库概述

数据库（DataBase，DB）：指长期保存在计算机的存储设备上，按照一定规则组织起来，可以被各种用户或应用共享的数据集合。(文件系统)

数据库管理系统（DataBase Management System，DBMS）：指一种操作和管理数据库的大型**软件**，用于建立、使用和维护数据库，对数据库进行统一管理和控制，以保证数据库的安全性和完整性。用户通过数据库管理系统访问数据库中的数据。

数据库软件应该为**数据库管理系统**，数据库是通过数据库管理系统创建和操作的。

数据库：存储、维护和管理数据的集合。

## 常见的数据库

### Oracle

目前比较成功的关系型数据库管理系统。运行稳定、功能齐全、性能超群、技术领先。主要应用在**大型的企业数据库领域，收费，有免费开发版**。

### DB2

IBM的产品。

### SQL Server

Microsoft的产品。软件界面友好、易学易用，在操作性和交互性方面独树一帜，跟.net c#。

### PostgreSQL

加州大学伯克利分校以教学目的开发的数据库系统，支持关系和面向对象的数据库，属于自由数据库管理系统。

### MySQL

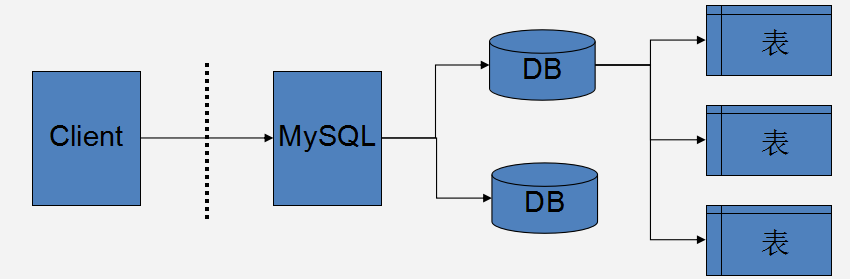
**免费**的数据库系统。**被广泛用于中小型应用系统**。体积小、速度快、总体拥有成本低，开放源代码。2008年被SUN收购，2009年SUN被Oracle收购。

## 安装MySQL

参照MySQL安装图解

## 数据库服务器、数据库和表的关系

* 所谓安装数据库服务器，只是在机器上装了一个数据库管理程序，这个管理程序可以管理多个数据库，一般开发人员会针对每一个应用创建一个数据库。
* 为保存应用中实体的数据，一般会在数据库创建多个表，以保存程序中实体的数据。
* 数据库服务器、数据库和表的关系如图所示：



## MySQL的登录

**首先确定在环境变量path里是否有mysql路径【C:\Program Files\MySQL\MySQL Server 5.7\bin】**

打开命令行cmd

输入**mysql -u root -p,然后密码，**

**接着输入show databases; 显示数据库**

**接着输入 exit; 退出数据库**

|  |
| --- |
|  |

## MySQL服务的停止与开启

在服务中找到MySQL,可在**运行中输入services.msc**

|  |
| --- |
|  |

## 修改mysql root用户密码

|  |
| --- |
|  |
| **使用mysqladmin命令：**  **mysqladmin -u root -p旧密码 password 新密码**  **mysqladmin -u root -pabcdef password abc** |

# SQL

## 概述

* SQL：Structure Query Language。（**结构化查询语言**）
* SQL被美国国家标准局（ANSI）确定为关系型数据库语言的美国标准，后来被国际化标准组织（ISO）采纳为**关系数据库语言的国际标准**。
* 各数据库厂商都支持ISO的SQL标准。普通话
* 各数据库厂商在标准的基础上做了自己的扩展。**方言**

## sql的分类

### DDL（Data Definition Language）

数据定义语言，用来定义数据库对象：库、表、列等；**CREATE、 ALTER、DROP**

### DML（Data Manipulation Language）

数据表操作语言，用来定义数据库记录（数据）；**INSERT、 UPDATE、 DELETE**

### DCL（Data Control Language）【DBA来做】

数据控制语言，用来定义**访问权限和安全级别**；

### DQL（Data Query Language）

数据查询语言，用来查询记录（数据）。**SELECT**

**注意：sql语句以;结尾**

## DDL:操作数据库、表、列等

掌握如何使用的关键字：**CREATE、 ALTER、 DROP**

### 操作数据库

**创建**

create database mydb1;

Create database mydb2 character set gbk;

Create database mydb3 character set gbk COLLATE gbk\_chinese\_ci;

COLLATE ：指排序规则

**查询**

查看当前数据库服务器中的所有数据库

show databases;

查看前面创建的mydb2数据库的定义信息

Show create database mydb2;

删除前面创建的mydb3数据库

Drop database mydb3;

**修改**

查看服务器中的数据库，并把mydb2的字符集修改为utf8;

alter database mydb2 character set utf8;

**删除**

drop database mydb3;

**其他：**

查看当前使用的数据库

select database();

切换数据库

use mydb2;

### 操作数据表

#### 语法：

|  |
| --- |
| create table 表名(  **字段1 字段类型,**  **字段2 字段类型,**  **...**  **字段n 字段类型**  ); |

#### 常用数据类型

|  |  |  |
| --- | --- | --- |
| 分类 | 数据类型 | 说明 |
| 数值类型 | BIT(M) | 位类型。M指定位数，默认值1，范围1-64 |
| TINYINT [UNSIGNED] [ZEROFILL] | 带符号的范围是-128到127  无符号0到255。 |
| BOOL，BOOLEAN | 使用0或1表示真或假 |
| SMALLINT [UNSIGNED] [ZEROFILL] | 2的16次方 |
| INT [UNSIGNED] [ZEROFILL] | 2的32次方 |
| BIGINT [UNSIGNED] [ZEROFILL] | 2的64次方 |
| FLOAT[(M,D)] [UNSIGNED] [ZEROFILL] | M指定显示长度，d指定小数位数 |
| DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL] | 表示比float精度更大的小数 |
| 文本、二进制类型 | CHAR(size) char(20) | 固定长度字符串 |
| VARCHAR(size) varchar(20) | 可变长度字符串 |
| BLOB LONGBLOB | 二进制数据 |
| TEXT(clob) LONGTEXT(longclob) | 大文本 |
| 时间日期 | DATE/DATETIME/TimeStamp | 日期类型(YYYY-MM-DD) (YYYY-MM-DD HH:MM:SS)，TimeStamp表示时间戳，它可用于自动记录insert、update操作的时间 |
| **VARCHAR、BLOB和TEXT类是变长类型。每个类型的存储需求取决于列值的实际长度。** | | |

#### 创建一个员工表

|  |
| --- |
|  |
| create table **employee**(  id int,  name varchar(20), 易洋千喜  gender bool,//true 男性 ，false 女性  gender varchar(6) male 男性 female:女性  birthday date,  Entry\_date date,  job varchar(20),  salary float,  resume text  ); |

改：gender用boolean型

#### 当前数据库中的所有表

SHOW TABLES;

#### 查看表的字段信息

DESC employee;

#### 增加一个image列。

ALTER TABLE employee ADD image blob;

#### 修改job列长度为60

ALTER TABLE employee MODIFY job varchar(60);

#### 删除image列

ALTER TABLE employee DROP image;//一次只能删除一个列

#### 表名改为user。

RENAME TABLE employee TO user;

#### 查看表格的创建细节

SHOW CREATE TABLE user;

#### 修改表的字符集为gbk

ALTER TABLE user CHARACTER SET gbk;

#### 列名name修改为username

ALTER TABLE user **CHANGE** name username varchar(100);

#### 删除表

DROP TABLE user ;

## DML

DML是对表中的数据进行增、删、改的操作。**不要与DDL混淆了**。

INSERT 、UPDATE、 DELETE

**小知识：**

在mysql中，字符串类型和日期类型都要用单引号括起来。'tom' '2015-09-04'

空值：null

### 插入操作：INSERT

|  |
| --- |
| **语法：** INSERT INTO 表名（列名1，列名2 ...）VALUES(列值1，列值2...);  注意：列名与列值的类型、个数、顺序要一一对应。  可以把列名当做java中的形参，把列值当做实参。  值不要超出列定义的长度。  如果插入空值，请使用null  插入的日期和字符一样，都使用引号括起来。 |

|  |
| --- |
| create table emp(  id int,  name varchar(100),  gender varchar(10),  birthday date,  salary float(10,2),  job varchar(20),  entry\_date date,  resume text  ); |
|  |
|  |

**插**nto emp(id,name,gender,birthday,salary,entry\_date,resume)

values (1,'gyf','female','1988-10-12',100000.00,'2000-10-12','good good');

insert into em**入数据**

insert ip(id,name,gender,birthday,salary,entry\_date,resume)

values (2,'mayun','female','1988-10-12',100000.00,'2000-10-12','good good');

insert into emp(id,name,gender,birthday,salary,entry\_date,resume)

values (3,'mht','female','1988-10-12',100000.00,'2000-10-12','good good');

**批量插入**

insert into emp(id,name,gender,birthday,salary,entry\_date,resume)

values

(4,'gyf1','female','1988-10-12',100000.00,'2000-10-12','good good'),

(5,'mayun1','female','1988-10-12',100000.00,'2000-10-12','good good'),

(6,'mht1','female','1988-10-12',100000.00,'2000-10-12','good good');

### 修改操作 UPDATE

|  |
| --- |
| 语法：UPDATE 表名 SET 列名1=列值1，列名2=列值2 。。。 WHERE 列名=值 |

#### 练习

|  |
| --- |
| 将所有员工薪水修改为5000元。  UPDATE emp SET salary=5000  将姓名为zhangsan的员工薪水修改为3000元。  UPDATE emp SET salary=3000 WHERE name=’ zhangsan’;  将姓名为lisi的员工薪水修改为4000元,job改为工程师。  UPDATE emp SET salary=4000,gender='female' WHERE name='lisi';  将wangwu的薪水在原有基础上增加1000元。  UPDATE emp SET salary=salary+1000 WHERE name='wangwu'; |

### 删除操作 DELETE

|  |
| --- |
| 语法 ： DELETE FROM 表名 【WHERE 列名=值】 |

|  |
| --- |
| 删除表中名称为’zs’的记录。  DELETE FROM emp WHERE name=‘zs’;  删除表中所有记录。  DELETE FROM emp;  使用truncate删除表中记录。  TRUNCATE TABLE emp; |

**注：**

**DELETE 删除表中的数据，表结构还在;删除后的数据可以找回**

**TRUNCATE 删除是把表直接DROP掉，然后再创建一个同样的新表。删除的数据不能找回。执行速度比DELETE快。**

## 安装mysql客户端工具

* Navicat for MySQL 11.0.10（64bit） + 破解补丁
* sqlyog82.exe

## DQL操作

DQL数据查询语言 **（重要）**

数据库执行DQL语句不会对数据进行改变，而是让数据库发送结果集给客户端。

查询返回的结果集是一张**虚拟表**。

### 查询关键字：SELECT 语法

|  |
| --- |
| SELECT selection\_list /\*要查询的列名称\*/  FROM table\_list /\*要查询的表名称\*/  WHERE condition /\*行条件\*/  GROUP BY grouping\_columns /\*对结果分组\*/  HAVING condition /\*分组后的行条件\*/  ORDER BY sorting\_columns /\*对结果排序\*/  LIMIT offset\_start, row\_count /\*结果限定\*/ |

### 创建表

#### 学生表

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **sid** | char(6) | 学生学号 |
| **sname** | varchar(50) | 学生姓名 |
| **age** | int | 学生年龄 |
| **gender** | varchar(6) | 学生性别 |

|  |
| --- |
| CREATE TABLE stu (  sid CHAR(6),  sname VARCHAR(50),  age INT,  gender VARCHAR(6)  ); |
| INSERT INTO stu VALUES('S\_1001', 'liuYi', 35, 'male');  INSERT INTO stu VALUES('S\_1002', 'chenEr', 15, 'female');  INSERT INTO stu VALUES('S\_1003', 'zhangSan', 95, 'male');  INSERT INTO stu VALUES('S\_1004', 'liSi', 65, 'female');  INSERT INTO stu VALUES('S\_1005', 'wangWu', 55, 'male');  INSERT INTO stu VALUES('S\_1006', 'zhaoLiu', 75, 'female');  INSERT INTO stu VALUES('S\_1007', 'sunQi', 25, 'male');  INSERT INTO stu VALUES('S\_1008', 'zhouBa', 45, 'female');  INSERT INTO stu VALUES('S\_1009', 'wuJiu', 85, 'male');  INSERT INTO stu VALUES('S\_1010', 'zhengShi', 5, 'female');  INSERT INTO stu VALUES('S\_1011', 'xxx', NULL, NULL);  INSERT INTO stu VALUES('S\_1009', 'wuJiu', 85, 'male');  INSERT INTO stu VALUES('S\_1010', 'zhengShi', 5, 'female');  INSERT INTO stu VALUES('S\_1011', 'xxx', NULL, NULL); |

#### 员工表

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **empno** | int | 员工编号 |
| **ename** | varchar(50) | 员工姓名 |
| **job** | varchar(50) | 员工工作 |
| **mgr** | int | 领导编号 |
| **hiredate** | date | 入职日期 |
| **sal** | decimal(7,2) | 月薪 |
| **comm** | decimal(7,2) | 奖金 |
| **deptno** | int | 部分编号 |

|  |
| --- |
| CREATE TABLE emp(  empno INT,  ename VARCHAR(50),  job VARCHAR(50),  mgr INT,  hiredate DATE,  sal DECIMAL(7,2),  comm decimal(7,2),  deptno INT  ); |
| INSERT INTO emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20);  INSERT INTO emp values(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30);  INSERT INTO emp values(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30);  INSERT INTO emp values(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20);  INSERT INTO emp values(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30);  INSERT INTO emp values(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30);  INSERT INTO emp values(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10);  INSERT INTO emp values(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20);  INSERT INTO emp values(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10);  INSERT INTO emp values(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30);  INSERT INTO emp values(7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20);  INSERT INTO emp values(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30);  INSERT INTO emp values(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20);  INSERT INTO emp values(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10); |

#### 部门表：dept

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **deptno** | int | 部分编码 |
| **dname** | varchar(50) | 部分名称 |
| **loc** | varchar(50) | 部分所在地点 |

|  |
| --- |
| CREATE TABLE dept(  deptno INT,  dname varchar(14),  loc varchar(13)  ); |
| INSERT INTO dept values(10, 'ACCOUNTING', 'NEW YORK');  INSERT INTO dept values(20, 'RESEARCH', 'DALLAS');  INSERT INTO dept values(30, 'SALES', 'CHICAGO');  INSERT INTO dept values(40, 'OPERATIONS', 'BOSTON'); |

### 基础查询

#### 查询所有列

SELECT \* FROM stu;

#### 查询指定列

SELECT sid, sname, age FROM stu;

### 条件查询

条件查询介绍

条件查询就是在查询时**给出WHERE子句**，在WHERE子句中可以使用如下运算符及关键字：

* =、!=、<>、<、<=、>、>=；
* BETWEEN…AND；
* IN(set)；
* IS NULL； IS NOT NULL
* AND；
* OR；
* NOT；

|  |
| --- |
| **查询性别为女，并且年龄50的记录**  **SELECT \* FROM stu WHERE gender='female' AND age<50;**  **查询学号为S\_1001，或者姓名为liSi的记录**  **SELECT \* FROM stu WHERE sid ='S\_1001' OR sname='liSi';**  **查询学号为S\_1001，S\_1002，S\_1003的记录**  **select \* from stu where sid in ('S\_1001','S\_1002','S\_1003');**  **查询学号不是S\_1001，S\_1002，S\_1003的记录**  **select \* from stu where sid not in ('S\_1001','S\_1002','S\_1003');**  **查询年龄为null的记录**  **select \* from stu where age is null;**  **查询年龄在20到40之间的学生记录**  **select \* from stu where age >= 20 and age <= 40;**  **select \* from stu where age between 20 and 40;**  **查询性别非男的学生记录**  **SELECT \* FROM stu WHERE gender!='male';**  **或者**  **SELECT \* FROM stu WHERE gender<>'male';**  **或者**  **SELECT \* FROM stu WHERE NOT gender='male';**  **查询姓名不为null的学生记录**  **SELECT \* FROM stu WHERE sname IS NOT NULL;**  **或者**  **SELECT \* FROM stu WHERE NOT sname IS NULL;** |
|  |

### 模糊查询

当想查询姓名中包含a字母的学生时就需要使用模糊查询了。模糊查询需要使用关键字**LIKE**。

通配符:

\_ 任意一个字符

%：任意0~n个字符

'%张%' '张\_'

**查询姓名由5个字母构成的学生记录**

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_\_\_\_\_';**

模糊查询必须使用LIKE关键字。其中 “\_”匹配任意一个字母，5个“\_”表示5个任意字母。

**查询姓名由5个字母构成，并且第5个字母为“i”的学生记录**

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_\_\_\_i';**

**查询姓名以“z”开头的学生记录**

**SELECT \***

**FROM stu**

**WHERE sname LIKE 'z%';**

其中“%”匹配0~n个任何字母。

**查询姓名中第2个字母为“i”的学生记录**

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_i%';**

**查询姓名中包含“a”字母的学生记录**

**SELECT \***

**FROM stu**

**WHERE sname LIKE '%a%';**

### 字段控制查询

**去除重复记录**

去除重复记录（两行或两行以上记录中系列的上的数据都相同），例如emp表中sal字段就存在相同的记录。当只查询emp表的sal字段时，那么会出现重复记录，那么想去除重复记录，需要使用DISTINCT：

**SELECT DISTINCT sal FROM emp;**

**查看雇员的月薪与佣金之和**

　　因为sal和comm两列的类型都是数值类型，所以可以做加运算。如果sal或comm中有一个字段不是数值类型，那么会出错。

**SELECT \*,sal+comm FROM emp;**

comm列有很多记录的值为NULL，因为任何东西与NULL相加结果还是NULL，所以结算结果可能会出现NULL。下面使用了把NULL转换成数值0的函数IFNULL：

**SELECT \*,sal+IFNULL(comm,0) FROM emp;**

**给列名添加别名**

在上面查询中出现列名为sal+IFNULL(comm,0)，这很不美观，现在我们给这一列给出一个别名，为total：

**SELECT \*, sal+IFNULL(comm,0) AS total FROM emp;**

给列起别名时，是可以省略AS关键字的：

**SELECT \*,sal+IFNULL(comm,0) total FROM emp;**

### 排序

**order by 列名 asc(默认) 升序 desc降序**

**查询所有学生记录，按年龄升序排序**

**SELECT \***

**FROM stu**

**ORDER BY sage ASC;**

或者

**SELECT \***

**FROM stu**

**ORDER BY sage;**

**查询所有学生记录，按年龄降序排序**

**SELECT \***

**FROM stu**

**ORDER BY age DESC;**

**查询所有雇员，按月薪降序排序，如果月薪相同时，按编号升序排序**

**SELECT \* FROM emp**

**ORDER BY sal DESC,empno ASC;**

### 聚合函数

聚合函数是用来做纵向运算的函数：

* COUNT()：**统计指定列不为NULL的记录行数**；
* MAX()：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；
* MIN()：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；
* SUM()：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0；
* AVG()：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0；

#### COUNT

当需要纵向统计时可以使用COUNT()。

* 查询emp表中记录数：

**SELECT COUNT(\*) AS cnt FROM emp;**

* 查询emp表中有佣金的人数：

**SELECT COUNT(comm) cnt FROM emp;**

注意，因为count()函数中给出的是comm列，那么只统计comm列非NULL的行数。

* 查询emp表中月薪大于2500的人数：

**SELECT COUNT(\*) FROM emp**

**WHERE sal > 2500;**

* 统计月薪与佣金之和大于2500元的人数：

**SELECT COUNT(\*) AS cnt FROM emp WHERE sal+IFNULL(comm,0) > 2500;**

* 查询有佣金的人数，有领导的人数：

**SELECT COUNT(comm), COUNT(mgr) FROM emp;**

#### SUM和AVG

当需要纵向求和时使用sum()函数。

* 查询所有雇员月薪和：

**SELECT SUM(sal) FROM emp;**

* 查询所有雇员月薪和，以及所有雇员佣金和：

**SELECT SUM(sal), SUM(comm) FROM emp;**

* 查询所有雇员月薪+佣金和：

**SELECT SUM(sal+IFNULL(comm,0)) FROM emp;**

* 统计所有员工平均工资：

**SELECT AVG(sal) FROM emp;**

#### MAX和MIN

* 查询最高工资和最低工资：

**SELECT MAX(sal), MIN(sal) FROM emp;**

### 分组查询

当需要分组查询时需要使用GROUP BY子句，例如查询每个部门的工资和，这说明要使用部门来分组。

注：凡和聚合函数同时出现的列名，一定要写在group by 之后

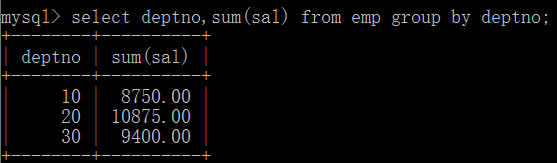
#### 分组查询

* 查询每个部门的部门编号和每个部门的工资和：

**SELECT deptno, SUM(sal)**

**FROM emp**

**GROUP BY deptno;**



* 查询每个部门的部门编号以及每个部门的人数：

**SELECT deptno,COUNT(\*)**

**FROM emp**

**GROUP BY deptno;**

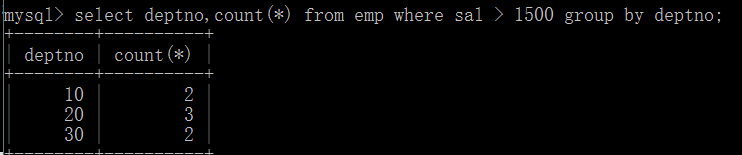
* 查询每个部门的部门编号以及每个部门工资大于1500的人数：

**SELECT deptno,COUNT(\*)**

**FROM emp**

**WHERE sal>1500**

**GROUP BY deptno;**



#### HAVING子句

* 查询工资总和大于9000的部门编号以及工资和：

**SELECT deptno, SUM(sal)**

**FROM emp**

**GROUP BY deptno**

**HAVING SUM(sal) > 9000;**

注：having与where的区别:

1.**having是在分组后对数据进行过滤**.

where是在分组前对数据进行过滤

2.having后面可以使用聚合函数(统计函数)

where后面不可以使用聚合函数。

WHERE是对分组前记录的条件，如果某行记录没有满足WHERE子句的条件，那么这行记录不会参加分组；而HAVING是对分组后数据的约束。

### LIMIT 方言

LIMIT用来限定查询结果的起始行，以及总行数。

**查询5行记录，起始行从0开始**

**SELECT \* FROM emp LIMIT 0, 5;**

注意，起始行从0开始，即第一行开始！

**查询10行记录，起始行从3开始**

**SELECT \* FROM emp LIMIT 3, 10;**

**分页查询**

如果一页记录为5条，希望查看第3页记录应该怎么查呢？

* 第一页记录起始行为0，一共查询5行；
* 第二页记录起始行为5，一共查询5行；

第三页记录起始行为10，一共查询5行；

# 数据的完整性

作用：保证用户输入的数据保存到数据库中是正确的。

确保数据的完整性 = 在创建表时给表中添加约束

## 完整性的分类：

> 实体完整性

> 域完整性

> 引用完整性

## 实体完整性

实体：即表中的一行(一条记录)代表一个实体（entity）

实体完整性的作用：标识每一行数据不重复。

**约束类型：** 主键约束（primary key） 唯一约束(unique) 自动增长列(auto\_increment)

### 主键约束（primary key）

**特点：数据唯一，且不能为null**

#### 第一种添加方式：

|  |
| --- |
| CREATE TABLE student(  Id int primary key,  Name varchar(50)  ); |

#### 第二种添加方式：

此种方式优势在于，可以创建联合主键

|  |
| --- |
| CREATE TABLE student(  id int,  name varchar(50),  primary key(id)  );  CREATE TABLE student(  id int,  name varchar(50),  primary key(id，name)  ); |

#### 第三种添加方式：

|  |
| --- |
| CREATE TABLE student(  id int,  name varchar(50)  );  ALTER TABLE student  ADD PRIMARY KEY (id); |

### 唯一约束(unique)：

|  |
| --- |
| CREATE TABLE student(  id int primary key,  name varchar(50) unique  ); |

### 自动增长列(au**t**o\_increment)

给主键添加自动增长的数值，**列只能是整数类型**，但是如果删除之前增长的序号，后面再添加的时候序号不会重新开始，而是会接着被删除的那一列的序号

|  |
| --- |
| CREATE TABLE student(  id int primary key auto\_increment,  name varchar(50)  );  INSERT INTO student(name) values(‘tom’); |

## 域完整性

域完整性的作用：限制此单元格的数据正确与否

域完整性约束：数据类型 **非空约束（not null） 默认值约束(default)**

**Check约束（mysql不支持） check();**

### 非空约束：not null

CREATE TABLE student(

id int pirmary key,

name varchar(50) not null,

sex varchar(10)

);

INSERT INTO student values(1,’tom’,null);

### 默认值约束 default

CREATE TABLE student(

Id int pirmary key,

Name varchar(50) not null,

Sex varchar(10) default ‘female’

);

insert into student1 values(1,'tom','female');

insert into student1 values(2,'jerry',default);

### 外键约束：FOREIGN KEY

* **学生（student）**和**成绩(score)**

|  |
| --- |
| CREATE TABLE student(  id int primary key,  name varchar(50) not null,  gender varchar(10) default '男'  );  create table score(  id int,  score int,  name varchar(6),  sid int,  CONSTRAINT fk\_score\_student FOREIGN KEY(sid) REFERENCES student(id)  );  第二种添加外键方式。  ALTER TABLE score1 ADD CONSTRAINT fk\_stu\_score FOREIGN KEY(sid) REFERENCES student(id);  插入数据  INSERT INTO student values(1,'jerry','女');  INSERT INTO student values(2,'tony','女');  INSERT INTO student values(3,'mia','男');  INSERT INTO score values(2,80,’数学’,4); |

## 表与表之间的关系

### 一对一：

例如t\_person表和t\_card表，即人和身份证。这种情况需要找出主从关系，即谁是主表，谁是从表。人可以没有身份证，但身份证必须要有人才行，所以人是主表，而身份证是从表。设计从表可以有两种方案：

在t\_card表中添加外键列（相对t\_user表），并且给外键添加唯一约束；

给t\_card表的主键添加外键约束（相对t\_user表），即t\_card表的主键也是外键。

### 一对多（多对一）：

最为常见的就是一对多！一对多和多对一，这是从哪个角度去看得出来的。t\_user和t\_section【部门】的关系，从t\_user来看就是一对多，而从t\_section的角度来看就是多对一！这种情况都是在多方创建外键！

### 多对多：

例如t\_stu和t\_teacher表，即一个学生可以有多个老师，而一个老师也可以有多个学生。这种情况通常需要创建中间表来处理多对多关系。例如再创建一张表t\_stu\_tea表，给出两个外键，一个相对t\_stu表的外键，另一个相对t\_teacher表的外键。

# 多表查询（重要）

## 多表查询有如下几种：

* 合并结果集；UNION 、UNION ALL
* 连接查询
* 内连接 [INNER] JOIN ON
* 外连接 OUTER JOIN ON
* 左外连接 LEFT [OUTER] JOIN
* 右外连接 RIGHT [OUTER] JOIN
* **全外连接（MySQL不支持）FULL JOIN**
* 自然连接 NATURAL JOIN
* 子查询

## 合并结果集

作用：合并结果集就是把**两个select语句的查询结果合并到一起**！

合并结果集有两种方式：

**UNION：去除重复记录**，例如：SELECT \* FROM t1 UNION SELECT \* FROM t2；

**UNION ALL：不去除重复记录**，例如：SELECT \* FROM t1 UNION ALL SELECT \* FROM t2。

要求：**被合并的两个结果：列数、列类型必须相同**

|  |
| --- |
| CREATE TABLE employee\_china(  id int,  name varchar(50)  );  CREATE TABLE employee\_usa(  id int,  name varchar(50)  );  INSERT INTO employee\_usa VALUES (1,'michal');  INSERT INTO employee\_usa VALUES (2,'lucy');  INSERT INTO employee\_usa VALUES (3,'anmy');  INSERT INTO employee\_china VALUES (1,'马云');  INSERT INTO employee\_china VALUES (2,'郭永峰');  INSERT INTO employee\_china VALUES (3,'马化腾');  INSERT INTO employee\_usa VALUES (4,'vincent');  INSERT INTO employee\_china VALUES (4,'vincent');  **SELECT \* FROM employee\_china UNION SELECT \* FROM employee\_usa;**  **SELECT \* FROM employee\_china UNION ALL SELECT \* FROM employee\_usa;** |

## 连接查询 （非常重要）

连接查询就是求出**多个表的乘积**，例如t1连接t2，那么查询出的结果就是t1\*t2。

SELECT \* FROM employee\_china,employee\_usa;

连接查询会产生**笛卡尔积**，假设集合A={a,b}，集合B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1),(b,2)}。可以扩展到多个集合的情况。

那么多表查询产生这样的结果并不是我们想要的，那么怎么去除重复的，不想要的记录呢，当然是通过条件过滤。通常要查询的多个表之间都存在关联关系，那么就通过关联关系去除笛卡尔积。

|  |
| --- |
| CREATE TABLE department(  id int,  name varchar(50)  );  INSERT INTO department VALUE(10001,'销售部');  INSERT INTO department VALUE(10002,'咨询部');  INSERT INTO department VALUE(10003,'人事部');  INSERT INTO department VALUE(10004,'技术部');  CREATE TABLE employee(  id int,  name varchar(50),  depno int  );  INSERT INTO employee VALUES(1,'tony',10001);  INSERT INTO employee VALUES(2,'lucy',10001);  INSERT INTO employee VALUES(3,'mia',10001);  INSERT INTO employee VALUES(4,'amy',10002);  INSERT INTO employee VALUES(5,'jerry',10002);  INSERT INTO employee VALUES(6,'micheal',10003);  INSERT INTO employee VALUES(7,'lily',10003);  INSERT INTO employee VALUES(8,'elain',10004);  INSERT INTO employee VALUES(9,'ruly',10004);  INSERT INTO employee VALUES(10,'kk',10004);  INSERT INTO employee VALUES(11,'cici',10004); |
|  |

|  |  |
| --- | --- |
|  |  |
|  | |
| SELECT \* FROM employee e,department d WHERE e.depno = d.id;  SELECT e.id,e.name as '名字',d.name as '部门' FROM employee e,department d WHERE e.depno = d.id;**//取别名** | |

### 内连接

上面的连接语句就是内连接，但它不是SQL标准中的查询方式，可以理解为方言！SQL标准的内连接为：

|  |
| --- |
| SELECT \* FROM employee e **INNER JOIN** department d  ON e.depno=d.id; |

### 外连接（左连接、右连接）

外连接的特点：查询出的结果存在不满足条件的可能。

#### 左连接

是先查询出左表（即以左表为主），然后查询右表，右表中满足条件的显示出来，不满足条件的显示NULL。

|  |
| --- |
| INSERT INTO department VALUE(10005,'保洁部');  INSERT INTO employee (id,name) VALUES(12,'ela');  #左连接  SELECT \* FROM employee e LEFT OUTER JOIN department d  ON e.depno=d.id; |
|  |

#### 右连接

右连接就是先把右表中所有记录都查询出来，然后左表满足条件的显示，不满足显示NULL。

|  |
| --- |
| #右连接  SELECT \* FROM employee e RIGHT OUTER JOIN department d  ON e.depno=d.id; |
|  |

#### **连接查询心得**：

连接不限与两张表，连接查询也可以是三张、四张，甚至N张表的连接查询。**通常连接查询不可能需要整个笛卡尔积，而只是需要其中一部分，那么这时就需要使用条件来去除不需要的记录。**这个条件大多数情况下都是使用主外键关系去除。

两张表的连接查询一定有一个主外键关系，三张表的连接查询就一定有两个主外键关系，所以在大家不是很熟悉连接查询时，首先要学会去除无用笛卡尔积，那么就是用主外键关系作为条件来处理。如果两张表的查询，那么至少有一个主外键条件，三张表连接至少有两个主外键条件*。*

### 自然连接

大家也都知道，连接查询会产生无用笛卡尔积，我们通常使用主外键关系等式来去除它。而自然连接无需你去给出主外键等式，它会自动找到这一等式：

**两张连接的表中名称和类型完全一致的列作为条件**，例如**emp和dept表都存在deptno**列，并且类型一致，所以会被自然连接找到！

当然自然连接还有其他的查找条件的方式，但其他方式都可能存在问题！

|  |
| --- |
| #自然连接....  CREATE TABLE t\_department(  depno varchar(10),  depname varchar(50),  primary key(depno)  );  CREATE TABLE t\_employee(  eid int,  ename varchar(50),  depno varchar(10),  foreign key(depno) references t\_department(depno)  );  insert into t\_department values('10','财务');  insert into t\_department values('20','市场');  insert into t\_department values('30','Java研发部');  insert into t\_department values('40','PHP研发部');    insert into t\_employee values(1,'gyf','10');  insert into t\_employee values(2,'ljl','20');  insert into t\_employee values(3,'hg','30');    select \* from t\_employee natural join t\_department;//自然连接    SELECT \* FROM t\_employee inner JOIN t\_department using(depno);//等同于自然连接    select \* from t\_employee natural left join t\_department;//左连接  select \* from t\_employee leftjoin t\_department using(depno);//左连接  select \* from t\_employee natural right join t\_department;//右连接  select \* from t\_employee right join t\_department using(depno);//右连接 |

### 子查询

#### 先整个测试数据

|  |
| --- |
| CREATE TABLE depart(  depno int PRIMARY KEY,  name varchar(50),#部门名称  location varchar(255),#部门地点  setuptime TIMESTAMP#部门成立时间  );  CREATE TABLE emp(  empno INT PRIMARY KEY,  ename VARCHAR(50),  job VARCHAR(50),  mgr INT,  hiredate DATE,  sal DECIMAL(7,2),  comm decimal(7,2),  depno INT,  CONSTRAINT fk\_emp\_depart FOREIGN KEY(depno) REFERENCES depart(depno)  );  INSERT INTO depart VALUE(30,'销售部','广州',NULL);  INSERT INTO depart VALUE(20,'咨询部','深圳',NULL);  INSERT INTO depart VALUE(10,'技术部','深圳',NULL);  INSERT INTO emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20);  INSERT INTO emp values(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30);  INSERT INTO emp values(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30);  INSERT INTO emp values(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20);  INSERT INTO emp values(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30);  INSERT INTO emp values(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30);  INSERT INTO emp values(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10);  INSERT INTO emp values(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20);  INSERT INTO emp values(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10);  INSERT INTO emp values(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30);  INSERT INTO emp values(7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20);  INSERT INTO emp values(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30);  INSERT INTO emp values(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20);  INSERT INTO emp values(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10); |

#### 子查询概念：

一个select语句中包含另一个完整的select语句。

子查询就是嵌套查询，即SELECT中包含SELECT，如果一条语句中存在两个，或两个以上SELECT，那么就是子查询语句了。

* 子查询出现的位置：
* where后，作为查询条件的一部分；
* from后，作表；
* 当子查询出现在where后作为条件时，还可以使用如下关键字：
* any
* all
* 子查询结果集的形式：
* 单行单列（用于条件）
* 单行多列（用于条件）
* 多行单列（用于条件）
* 多行多列（用于表）

#### 1.工资高于JONES的员工。

分析：

查询条件：工资>JONES工资，其中JONES工资需要一条子查询。

第一步：查询JONES的工资

SELECT sal FROM emp WHERE ename='JONES'

第二步：查询高于甘宁工资的员工

SELECT \* FROM emp WHERE sal > (${第一步})

结果：

SELECT \* FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename='JONES')

#### 2、查询与SCOTT同一个部门的员工。

子查询作为条件

子查询形式为单行单列

SELECT \* FROM emp WHERE deptno = (SELECT deptno FROM emp WHERE ename = 'SCOTT');

#### 3、工资高于30号部门所有人的员工信息

分析：

SELECT \* FROM emp WHERE sal>(

SELECT MAX(sal) FROM emp WHERE depno=30);

查询条件：工资高于30部门所有人工资，其中30部门所有人工资是子查询。高于所有需要使用all关键字。

第一步：查询30部门所有人工资

SELECT sal FROM emp WHERE depno=30;

第二步：查询高于30部门所有人工资的员工信息

SELECT \* FROM emp WHERE sal > ALL (${第一步})

结果：

SELECT \* FROM emp WHERE **sal > ALL** [大于所有](SELECT sal FROM emp WHERE depno=30)

子查询作为条件

子查询形式为多行单列（当子查询结果集形式为多行单列时可以使用ALL或ANY关键字）

#### 4、查询工作和工资与MARTIN（**马丁**）完全相同的员工信息

分析：

查询条件：工作和工资与MARTIN完全相同，这是子查询

第一步：查询出MARTIN的工作和工资

SELECT job,sal FROM emp WHERE ename='MARTIN'

第二步：查询出与MARTIN工作和工资相同的人

SELECT \* FROM emp WHERE (job,sal) IN (${第一步})

结果：

SELECT \* FROM emp WHERE (job,sal) IN (SELECT job,sal FROM emp WHERE ename='MARTIN')

* **子查询作为条件，子查询形式为单行多列**

#### 5、有2个以上直接下属的员工信息

SELECT \* FROM emp WHERE empno IN(

SELECT mgr FROM emp GROUP BY mgr HAVING COUNT(mgr)>=2);

* **子查询作为条件，子查询形式为多行单列**

#### 6、查询员工编号为7788的员工名称、员工工资、部门名称、部门地址

分析：（无需子查询）

查询列：员工名称、员工工资、部门名称、部门地址

条件：员工编号为7788

两种方式

SELECT e.ename,e.sal,d.name,d.location

FROM emp e,depart d

WHERE e.depno = d.depno AND e.empno = '7788';

SELECT e.ename,e.sal,d.name,d.location

FROM emp e,(SELECT depno,name,location FROM depart) d

WHERE e.depno = d.depno AND e.empno = '7788';

**子查询作为表,子查询形式为多行多列**

#### **连接查询心得**：

连接不限与两张表，连接查询也可以是三张、四张，甚至N张表的连接查询。通常连接查询不可能需要整个笛卡尔积，而只是需要其中一部分，那么这时就需要使用条件来去除不需要的记录。这个条件大多数情况下都是使用主外键关系去除。

两张表的连接查询一定有一个主外键关系，三张表的连接查询就一定有两个主外键关系，所以在大家不是很熟悉连接查询时，首先要学会去除无用笛卡尔积，那么就是用主外键关系作为条件来处理。如果两张表的查询，那么至少有一个主外键条件，三张表连接至少有两个主外键条件*。*

### 自连接:自己连接自己，起别名

*求7369员工编号、姓名、经理编号和经理姓名*

*SELECT e1.empno , e1.ename,e2.mgr,e2.ename*

*FROM emp e1, emp e2*

*WHERE e1.mgr = e2.empno AND e1.empno = 7369;*

### 练习：

求各个部门薪水最高的员工所有信息

select e.\* from emp e,

--部门最高工资

(select max(sal) maxsal,depno from emp

group by depno) a

where e.depto = a.depto

and e.sal =a.maxsal

# MySQL中的函数

## 时间日期函数

|  |  |
| --- | --- |
| **ADDTIME (date2 ,time\_interval )** | **将time\_interval加到date2** |
| **CURRENT\_DATE (  )** | **当前日期** |
| **CURRENT\_TIME (  )** | **当前时间** |
| **CURRENT\_TIMESTAMP (  )** | **当前时间戳** |
| **DATE (datetime )** | **返回datetime的日期部分** |
| **DATE\_ADD (date2 , INTERVAL d\_value d\_type )** | **在date2中加上日期或时间** |
| **DATE\_SUB (date2 , INTERVAL d\_value d\_type )** | **在date2上减去一个时间** |
| **DATEDIFF (date1 ,date2 )** | **两个日期差** |
| **NOW (  )** | **当前时间** |
| **YEAR|Month|Day(datetime )** | **年月日** |

示例：SELECT ADDTIME('2017-11-18 02:30:30','1:1:20');;

注意：字符串、时间日期的引号问题

select date\_add(entry\_date,INTERVAL 2 year) from student;／／增加两年

select addtime(time,‘1 1-1 10:09:09’) from student;　／／时间戳上增加，注意年后没有-

## 字符串相关函数

|  |  |
| --- | --- |
| **CHARSET(str)** | **返回字串字符集** |
| **CONCAT (string2  [,... ])** | **连接字串** |
| **INSTR (string ,substring )** | **返回substring在string中出现的位置,没有返回0** |
| **UCASE (string2 )** | **转换成大写** |
| **LCASE (string2 )** | **转换成小写** |
| **LEFT (string2 ,length )** | **从string2中的左边起取length个字符** |
| **LENGTH (string )** | **string长度** |
| **REPLACE (str ,search\_str ,replace\_str )** | **在str中用replace\_str替换search\_str** |
| **STRCMP (string1 ,string2 )** | **逐字符比较两字串大小,** |
| **SUBSTRING (str , position  [,length ])** | **从str的position开始,取length个字符** |
| **LTRIM (string2 ) RTRIM (string2 )  trim** | **去除前端空格或后端空格** |

## 数学函数

|  |  |
| --- | --- |
| **ABS (number2 )** | **绝对值** |
| **BIN (decimal\_number )** | **十进制转二进制** |
| **CEILING (number2 )** | **向上取整** |
| **CONV(number2,from\_base,to\_base)** | **进制转换** |
| **FLOOR (number2 )** | **向下取整** |
| **FORMAT (number,decimal\_places )** | **保留小数位数** |
| **HEX (DecimalNumber )** | **转十六进制** |
| **LEAST (number , number2  [,..])** | **求最小值** |
| **MOD (numerator ,denominator )** | **求余** |
| **RAND([seed])** | **RAND([seed])** |

# MySQL数据库的备份与恢复

## 生成SQL脚本 导出数据

在控制台使用mysqldump命令可以用来生成指定数据库的脚本文本，但要注意，脚本文本中只包含数据库的内容，而不会存在创建数据库的语句！所以在恢复数据时，还需要自已手动创建一个数据库之后再去恢复数据。

|  |
| --- |
| mysqldump –u用户名 –p密码 数据库名>生成的脚本文件路径 |



现在可以在C盘下找到mydb1.sql文件了！

注意，mysqldump命令是在Windows控制台下执行，无需登录mysql！！！

Windows10 不能直接把数据备份到C盘根目录，如果想放在根目录下，cmd使用管理来运行，方便点，就直接把数据备份到桌面

|  |
| --- |
|  |
|  |

## 执行SQL脚本 恢复数据

**前提：必须先创建数据库名**

**执行SQL脚本需要登录mysql**，然后进入指定数据库，才可以执行SQL脚本！！！

执行SQL脚本不只是用来恢复数据库，也可以在平时编写SQL脚本，然后使用执行SQL 脚本来操作数据库！大家都知道，在黑屏下编写SQL语句时，就算发现了错误，可能也不能修改了。所以我建议大家使用脚本文件来编写SQL代码，然后执行之！

|  |
| --- |
| SOURCE C:\mydb1.sql |



　　注意，在执行脚本时需要先行核查当前数据库中的表是否与脚本文件中的语句有冲突！例如在脚本文件中存在create table a的语句，而当前数据库中已经存在了a表，那么就会出错！

还可以通过下面的方式来执行脚本文件：

mysql -uroot -p123 mydb1<c:\mydb1.sql

|  |
| --- |
| mysql –u用户名 –p密码 数据库<要执行脚本文件路径 |



**这种方式无需登录mysql！**

**注意：在CMD下 命令不能加;**