# 数据库是什么？

数据库就是数据仓库，用来存放数据的。

# 为什么要使用数据库？

方便操作数据（便于管理数据）

数据安全

# 常见数据库

Mysql

SQLServer

Oracle

DB2

# 常见概念

DB Database 数据库 ---保存数据

DBMS Database Manager System 数据库管理系统 ---操作数据库

SQL Structrue Query Language 结构化查询语言 --- 与数据库管理系统交互

DBA DataBase Administractor 数据库管理员

# Mysql如何使用？

安装mysql，参考Mysql安装.docx

# Mysql安装之后的细节



表示没有配置环境变量

需要手动配置mysql环境变量

将mysql安装目录复制：E:\MySQL\MySQL Server 5.5\bin

# SQL语法要求

SQL语句可以单行或多行书写，以分号结尾；

可以用空格和缩进来来增强语句的可读性；

SQL不区别大小写，关键字建议使用大写；

#第一种注释方式

-- 第二种注释方式

# SQL分类

DDL Data Defined Language 数据定义语言 （数据库，表的操作）

DML Data Manipulation Language 数据操作语言 （增，删，改）

DQL Data Query Language 数据查询语言 （查询数据）

DCL Data Control Language 数据控制语言 （权限控制）

# DDL

#关于数据库的操作

-- 创建数据库

/\*

语法：

create database 【if not exists】数据库名;

\*/

CREATE DATABASE IF NOT EXISTS mydb;

-- 删除数据库

/\*

语法：

drop database 【if exists】 数据库名;

\*/

DROP DATABASE IF EXISTS mydb;

-- 修改数据库名称

/\*

数据库名称的修改，mysql没有提供语法支持

需要修改，则直接修改数据库文件夹对应的文件名即可！！

一般在：C:\ProgramData\MySQL\MySQL Server 5.5\data

\*/

-- 显示所有的数据库列表

SHOW DATABASES;

-- 切换数据库

USE test;

#关于表的操作

-- 创建表

/\*

语法：

create table 【if not exists】 表名(

字段名1（列名） 数据类型(长度),

字段名2（列名） 数据类型(长度),

.....

);

\*/

CREATE TABLE IF NOT EXISTS student(

stuNo INT,-- 默认int类型长度为11

stuName VARCHAR(30),

age INT(3)

);

-- 删除表

/\*

语法：

drop table 【if exists】 表名;

\*/

DROP TABLE IF EXISTS student;

# 查看当前数据库中所有表名称：SHOW TABLES；

USE mysql;-- 切换数据库

SHOW TABLES;-- 显示该数据库所有的表

#查看指定表的创建语句：SHOW CREATE TABLE emp，看emp表的创建语句 (了解)

SHOW CREATE TABLE student;

#查看表结构：DESC emp，查看emp表结构；

DESC student;

#修改表：

-- 1. 修改之添加列：给stu表添加classname列：

ALTER TABLE student ADD COLUMN(gender CHAR(1));

DESC student;

-- 2. 修改之修改列类型：修改stu表的gender列类型为CHAR(2)：

ALTER TABLE student MODIFY age FLOAT(4,1);

-- 3. 修改之修改列名：修改stu表的gender列名为sex：

ALTER TABLE student CHANGE gender sex CHAR(1);

-- 4. 修改之删除列：删除stu表的classname列：

ALTER TABLE student DROP COLUMN sex;

-- 5. 修改之修改表名称：修改stu表名称为student：

ALTER TABLE student RENAME TO stu;

# DML

# DML 数据操作语言 （增删改）

-- 增 （插入）

/\*

语法：

insert into 表名(列名)

values(值)

\*/

-- 插入一条数据

/\*

注意：

①插入的值必须与列一一对应

②可以不插入所有的列

\*/

INSERT INTO stu(stuNo,stuname,age)

VALUES(1001,"张三",21);

-- 插入多条数据

INSERT INTO stu(stuNo,stuname,age)

VALUES(1002,"李四",21),(1003,"王五",22),(1004,"赵柳",24);

-- 如果插入所有的列，则列名可以省略

/\*

注意：

①值得个数必须与列的个数一致

②值应该与表的字段一一对应

\*/

INSERT INTO stu

VALUES(1009,"东哥",33,"学前班");

-- 默认值

-- 如果是主键，且设置了自动增长，则可以使用null赋值，实际上使用的是自动生成的主键

INSERT INTO mystu

VALUES(NULL,"aa",NULL);

-- 如果有默认值得列没有赋值，则自动使用默认值

INSERT INTO mystu(NAME)

VALUES("bb");

-- 自增长值或者默认值得列，一旦被赋值，则不使用默认值或者自增长的值

INSERT INTO mystu

VALUES(5,"cc",NULL);

-- 可以给有设置默认值的列赋值为default，则直接使用默认值

INSERT INTO mystu

VALUES(NULL,"cc",DEFAULT);

#修改数据

/\*

语法：

update 表名 set 列名=值,... where 条件;

\*/

UPDATE mystu SET NAME="东哥" WHERE id = 6;

-- 注意，修改数据需要加上过滤条件，否则全表都会被修改

UPDATE mystu SET NAME="东少";

UPDATE mystu SET NAME="东骚",score=99 WHERE id>4;

# 删除数据

/\*

语法：

delete from 表名 where 条件;

\*/

DELETE FROM mystu WHERE id=6;

/\*

truncate 清空表

语法：

truncate table 表名;

\*/

TRUNCATE TABLE mystu;

/\*

delete与truncate的区别：

①truncate 不能添加过滤条件，delete可以添加过滤条件

②delete会将删除的数据数量返回，truncate不会

③delete删除的数据可以回滚，但是truncate删除的数据就是永久删除

\*/

TRUNCATE TABLE stu;

SHOW VARIABLES LIKE "%autocommit%";

-- 设置mysql不允许自动提交事务

SET autocommit= 0;

DELETE FROM stu;

DELETE FROM stu WHERE stuno =1008;

# 提交事务

COMMIT;

# 回滚事务

ROLLBACK;

SELECT \* FROM stu;

# DCL(了解)

USE mysql;

SELECT \* FROM USER;

# 创建用户

/\*

语法：

create user 用户名 identified by 密码;

\*/

CREATE USER linsir IDENTIFIED BY '123456';

-- 将所有数据库和表的操作权限赋予

GRANT ALL ON \*.\* TO linsir;

-- 刷新权限表，让设置权限起效

FLUSH PRIVILEGES;

SHOW DATABASES;

SHOW TABLES;

-- 显示用户的权限

SHOW GRANTS FOR linsir;

-- 撤销权限

REVOKE ALL ON \*.\* FROM linsir;

-- 删除用户

DROP USER linsir;

-- 设置用户密码 其中PASSWORD()是将密码进行加密处理

UPDATE USER SET PASSWORD=PASSWORD('111') WHERE USER='linsir';

-- 刷新密码权限

FLUSH PRIVILEGES;

# DQL(重点)

# DQL 数据查询语言

-- 基础查询

/\*

CREATE TABLE `mystu` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(30) DEFAULT NULL,

`score` float(4,2) DEFAULT '0.00',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

插入数据

insert into mystu(`name`,score) values('aa',88),('bb',89),('cc',55),('dd',77),('ff',98);

/\*

语法：

select 查询列表 from 表名;

解释：

查询列表可以是常量，单个字段，多个字段，函数，以及上述组合

\*/

-- 案例1：查询常量

/\*

注意：查询是字符串需要使用单引号或者双引号引起来，

如果是数字，则不需要使用引号

\*/

SELECT 'afdafs';

SELECT "hello";

SELECT 123;

/\*

注意：sql语句可以进行运算

+的特殊运算：

在sql语句中+只表示加法运算：

如果是数值与数值进行加法，则直接加法运算

如果是数值与字符串进行加法，如果字符串可以转化为数值，

则转化为数值再相加，如果转化不了，则当成0进行相加，

如果与Null进行加法运算，则结果为NULL

\*/

SELECT 123\*3;

SELECT 123+'123';-- 246

SELECT 123+'abc';-- 123

SELECT 123+123+NULL;-- NULL

SELECT 'aa'+'bb';-- 0

-- 单个字段

SELECT NAME FROM mystu;

-- 多个字段

SELECT NAME,score

FROM mystu;

-- 查询表中所有的字段

-- 方式一

SELECT id,NAME,score

FROM mystu;

SELECT id,NAME,score

FROM mystu;

-- 方式二

SELECT \* FROM mystu;

-- 给字段取别名

/\*

要求：查询学生成绩，显示为姓名，成绩

\*/

-- 方式一

SELECT NAME '姓名',score 成绩

FROM mystu;

-- 方式二

SELECT NAME AS '姓名2',score AS 成绩

FROM mystu;

-- 查询当前用户

SELECT USER();

-- 组合

SELECT id,NAME,score,'三年级2班' AS 班级,USER() '当前用户'

FROM mystu;

# 条件查询

/\*

drop table if exists stu;

create table if not exists stu(

sid varchar(20),

sname varchar(20),

age int,

gender varchar(10)

);

insert into stu

values('S\_1001','zhangsan',20,'man'),

('S\_1002','lisi',21,'man'),

('S\_1003','wangwu',22,'women'),

('S\_1004','wangliu',18,'women'),

('S\_1005','longwang',23,'man'),

('S\_1006','tianqi',18,'women'),

('S\_1007','heh',28,'man'),

('S\_1008','haha',38,'women');

\*/

/\*

语法：

select 查询列表

from 表名

where 条件表达式;

条件表达式可以是比较表达式，逻辑表达式，模糊查询。

比较表达式使用比较运算符的表达式：

比较运算符：> < >= <= = 不推荐【!=】 推荐【<>】

逻辑表达式就是使用逻辑运算符的表达式：

逻辑运算符：and or not 不推荐【& | !】

条件表达式还可以是,in between..and,is null表达式

\*/

-- 案例1：查询年龄小于23岁的学生

SELECT \*

FROM stu

WHERE age < 23;

-- 案例2：查询sid为s\_1004的学生

SELECT \*

FROM stu

WHERE sid = 's\_1004';

-- 案例3：查询sid不为s\_1004的学生

SELECT \*

FROM stu

WHERE sid <> 's\_1004';

SELECT \*

FROM stu

WHERE sid != 's\_1004';

-- 案例4：查询学生sid为s\_1004或者s\_1005的学生信息

SELECT \*

FROM stu

WHERE sid = 's\_1004'

OR sid = 's\_1005';

-- 案例5：查询学生gender为women的，且年龄小于20岁的女孩

SELECT \*

FROM stu

WHERE gender = 'women'

AND age < 20;

-- in表达式 在...里面 简化or运算

-- 案例6：查询sid为s\_1004,s\_1005,s\_1006的学生信息

SELECT \*

FROM stu

WHERE sid = 's\_1004'

OR sid = 's\_1005'

OR sid = 's\_1006';

SELECT \*

FROM stu

WHERE sid IN ('s\_1004','s\_1005','s\_1006');

-- between...and表达式 在...之间

/\*

注意：between...and表达式 小的值必须在前面，大的值必须在后面，不能交换顺序

包含两端的端点值

\*/

-- 案例7：查询年龄在21-25岁之间的学生信息

SELECT \*

FROM stu

WHERE age >=21

AND age <= 25;

SELECT \*

FROM stu

WHERE age BETWEEN 21 AND 22;

-- 案例8：查询性别为null的学生信息

SELECT \*

FROM stu

WHERE gender IS NULL;

-- 案例9: 查询性别不为null的学生信息

SELECT \*

FROM stu

WHERE gender IS NOT NULL;

-- like模糊查询

/\*

%: 0或任意个字符

\_: 一个任意字符

\*/

-- 案例10：查询以wang开头的学生信息

SELECT \*

FROM stu

WHERE sname LIKE 'wang%';

-- 案例11：查询学生姓名第三个字母为a的学生信息

SELECT \*

FROM stu

WHERE sname LIKE '\_\_a%';

-- 案例12：查询学生姓名中包含an的学生信息

SELECT \*

FROM stu

WHERE sname LIKE '%an%';

-- 查询结果去重

-- 案例13：查询学生的所有性别

SELECT DISTINCT gender

FROM stu;

-- ifnull函数 通用函数，如果对应字段结果是NULL，则可以使用后面的值进行替换

-- 案例14：查询每个学生每年花费的零花钱（包含压岁钱）,每个学生过年都会有10000的压岁钱

-- UPDATE STU SET MONEY= NULL WHERE sid = 's\_1008';

SELECT sid,sname,age,gender,money,10000 + IFNULL(money,0)\*12 "全年零花钱"

FROM stu;

-- 排序查询

/\*

语法：

select 查询列表

from 表名

【

where 条件表达式

】

order by 排序列表 asc|desc;

asc 表示升序，也是默认值，可以省略

desc 表示降序

排序列表可以是：单个字段，多个字段，表达式

\*/

-- 案例15：查询学生信息，按照学生的年龄降序排列

SELECT \*

FROM stu

ORDER BY age DESC;

-- 案例16：查询学生信息，按照学生的sname升序排列

SELECT \*

FROM stu

ORDER BY sname ASC;

-- 案例17：查询学生信息，按照年龄进行降序排列，如果年龄相同，按照姓名降序排列

SELECT \*

FROM stu

ORDER BY age DESC,sname DESC;

-- 案例18：查询学生信息和每年零花钱（压岁钱10000+每个月零花钱\*12），按照每年零花钱降序排列

SELECT sid,sname,money,10000+IFNULL(money,0)\*12 '零花钱'

FROM stu

ORDER BY 10000+IFNULL(money,0)\*12 DESC;

-- 案例19：查询女学生的学生信息，按照零花钱降序排列

SELECT \*

FROM stu

WHERE gender = 'women'

ORDER BY money DESC;

-- 聚合函数也称为分组函数

/\*

解释：将表中的数据进行分组，然后再统计，得出一个结果

max

min

avg

sum

count

\*/

-- 案例20：查询所有学生最高年龄，最低年龄，平均年龄，年龄总和，总人数

SELECT MAX(age) '最高年龄',MIN(age) 最低年龄,AVG(age) 平均年龄,

SUM(age) 年龄总和,COUNT(\*) 总人数

FROM stu;

-- 这些组函数是否都是忽略NULL值 都会忽略Null值

-- 这些组函数都可以对任意类型的数据进行统计么

-- max min count 可以是任意类型的数据

-- sum avg 只能处理数值型数据

-- count 统计函数

/\*

count(任意内容)

通常会放入一个常量或者一个字段名或者\*

count(1)

count(sid)

count(\*) 推荐使用

\*/

SELECT COUNT(\*)

FROM stu;

-- 分组查询

/\*

语法：

select 查询列表

from 表名

【

where 分组前过滤

】

group by 分组列表 可以使用别名

having 分组后过滤 可以使用别名

【

order by 排序列表

】

\*/

SELECT \* FROM emp;

-- 案例21：查询每个部门的平均工资

SELECT deptno,AVG(sal)

FROM emp

GROUP BY deptno;

-- 案例22：查询每个部门的总工资

SELECT deptno,SUM(sal)

FROM emp

GROUP BY deptno;

-- 案例23：查询部门编号为20的部门平均工资

SELECT deptno,AVG(sal)

FROM emp

WHERE deptno = 20;

-- 案例24：查询部门平均工资大于20000的部门编号和部门平均工资

SELECT deptno,AVG(sal)

FROM emp

GROUP BY deptno

HAVING AVG(sal)>20000;-- 分组后过滤

-- 案例25：查询部门平均工资大于20000的部门编号和部门平均工资，且平均工资显示为“平均工资”

SELECT deptno,AVG(sal) "平均工资"

FROM emp

GROUP BY deptno

HAVING AVG(sal)>20000;

SELECT deptno num,AVG(sal) "平均工资"

FROM emp

GROUP BY num

HAVING 平均工资 < 20000;

-- 案例26：查询所有部门不包括job=分析师的部门平均工资和部门编号,且平均工资大于16000

SELECT deptno,AVG(sal) 平均工资

FROM emp

WHERE job != '分析师'

GROUP BY deptno

HAVING 平均工资>16000;

-- 分页查询

/\*

语法：

select 查询列表

from 表名

【

where 分组前过滤

group by 分组列表

having 分组后过滤

order by 排序列表 asc|desc

】

limit m,n;

其中n就是个数，m开始的下标（从0开始）

注意：m可以省略，默认就是0

\*/

/\*

案例27：

查询员工工资前三高的员工信息

\*/

SELECT \*

FROM emp

ORDER BY sal DESC

LIMIT 3;

/\*

案例28：

查询第4,5,6高工资的员工信息

\*/

SELECT \*

FROM emp

ORDER BY sal DESC

LIMIT 3,3;

/\*

案例29：每页显示3条数据，显示第4页的数据

\*/

SELECT \*

FROM emp

LIMIT 9,3;

/\*

案例30：总结

每页显示pageSize条数据，需要显示第page页，则limit函数应该怎么写

limit (page-1)\*pageSize,pageSize;

\*/

SELECT \*

FROM emp

LIMIT 6,3;

# 多表查询

# 多表查询

-- 需求：查询所有员工的员工姓名和部门名称

/\*

笛卡尔积：

产生的原因？没有连接条件

解决：给上连接条件

\*/

SELECT ename,dname

FROM emp,dept;

SELECT ename,dname

FROM emp,dept

WHERE emp.deptno = dept.deptno;

# 表也可以取别名 取别名主要就是简化表名

SELECT e.ename,d.dname,e.deptno

FROM emp e,dept d

WHERE e.deptno = d.deptno;

-- 合并结果集

-- 案例1：查询员工姓名和工资，以及学生的姓名和年龄

SELECT ename,sal

FROM emp

UNION ALL

SELECT sname,age

FROM stu;

/\*

UNION 与UNION ALL的区别？

UNION合并结果集并去重，

UNION ALL合并结果集不去重

要求：被合并的两个结果：列数、列类型必须相同。

\*/

-- 连接查询

/\*

内连接：

SQL99语法：

select 查询列表

from 表1

【inner】 join 表2

on 连接条件

\*/

-- 需求：查询所有员工的员工姓名和部门名称

SELECT e.ename,d.dname

FROM emp e

INNER JOIN dept d

ON e.deptno = d.deptno;

-- 外连接

/\*

语法：

select 查询列表

from 表1

left|right 【outer】 join 表2

on 连接条件

\*/

-- 案例2：查询没有员工的部门编号和部门名称

SELECT d.deptno,d.dname

FROM dept d

LEFT OUTER JOIN emp e

ON d.deptno = e.deptno

WHERE e.empno IS NULL;

-- 案例3：查询没有员工的部门编号和部门名称

SELECT d.deptno,d.dname

FROM emp e

RIGHT OUTER JOIN dept d

ON d.deptno = e.deptno

WHERE e.empno IS NULL;

-- 案例4：查询没有部门的员工姓名

SELECT e.ename

FROM emp e

LEFT JOIN dept d

ON e.deptno = d.deptno

WHERE d.deptno IS NULL;

-- 自然连接（了解） 一般不使用

-- 案例5：查询员工的姓名和部门编号

SELECT ename,dname

FROM emp

NATURAL JOIN dept;

# 子查询

# 子查询 （难点）

/\*

子查询解释：

查询里面嵌套查询，外面的查询称为主查询，

里面嵌套的查询就是子查询

语法：

select 查询列表(子查询)

from 表名(子查询)

where 条件表达式（子查询） -- 常用

group by 分组列表

having 分组后过滤（子查询） -- 常用

exists （子查询）

注意：

①子查询一般会比主查询优先执行，并将结果给主查询使用

②子查询如果是作为过滤条件，则一般会放在条件右边

③子查询一般会使用()包起来

\*/

-- 案例1：查询工资比刘备高的员工信息

SELECT \*

FROM emp

WHERE sal > (SELECT sal

FROM emp

WHERE ename = '刘备');

/\*

子查询分类：

根据子查询返回的结果进行分类：

单行子查询： 返回一行一列数据的子查询称为单行子查询，

一般结果单行操作符使用：

> < >= <= = <>

多行子查询： 返回多行一列的子查询称为多行子查询

一般会结合多行操作符使用：

in any all

\*/

-- 案例2：查询工资比20号部门所有员工工资都高的员工信息

-- a.使用单行操作符实现

SELECT \*

FROM emp

WHERE sal > (SELECT MAX(sal)

FROM emp

WHERE deptno=20);

/\*

all 的使用，表示全部都

>all 等同于>最大值

<all 等同于<最小值

\*/

SELECT \*

FROM emp

WHERE sal > ALL (SELECT sal

FROM emp WHERE deptno=20);

-- 案例3：查询员工工资大于10号部门任一员工的员工信息，不包含10号部门的员工

-- 其实就是查询工资大于10号部门最低工资的员工

SELECT \*

FROM emp

WHERE sal > (SELECT MIN(sal)

FROM emp

WHERE deptno=10)

AND deptno <>10;

/\*

any 的使用，表示任一，其中一个

<any 等同于小于最大值

>any 等同于大于最小值

\*/

SELECT \*

FROM emp

WHERE sal > ANY(SELECT sal

FROM emp WHERE deptno=10);

-- 案例4： 查询工作和工资与诸葛亮完全相同的员工信息分析

SELECT \*

FROM emp

WHERE (job,sal) IN (

SELECT job,sal

FROM emp

WHERE ename = '诸葛亮'

);

-- 案例5：在from后面使用子查询

SELECT t.\*

FROM (SELECT \* FROM emp) t

WHERE t.empno>1002;

-- 案例6：在select后面使用子查询

SELECT e.\*,(SELECT d.dname FROM dept d WHERE d.deptno = e.deptno)

FROM emp e;

# 约束

# 约束

/\*

6大约束：

主键约束 Primary Key

非空约束 Not NULL

唯一约束 unique

检查约束 check mysql不支持

外键约束

默认约束 default

\*/

-- 主键约束 不能为NULL，且不能重复

CREATE TABLE testPK(

id INT PRIMARY KEY

);

-- 非空

INSERT INTO testPK VALUES(NULL);

-- 唯一

INSERT INTO testPK VALUES(100);

-- 非空约束

DROP TABLE testPK;

CREATE TABLE testPK(

id INT PRIMARY KEY,

NAME VARCHAR(20) NOT NULL

);

INSERT INTO testpk VALUES(101,'aa');

INSERT INTO testpk VALUES(101,NULL);

-- 唯一约束

DROP TABLE testPK;

CREATE TABLE testPK(

id INT PRIMARY KEY,

NAME VARCHAR(20) UNIQUE

);

INSERT INTO testpk VALUES(102,'aa');

INSERT INTO testpk VALUES(103,'aa1');

-- 外键约束

CREATE TABLE score(

sid INT PRIMARY KEY,

SUBJECT VARCHAR(20),

score INT,

stuid INT

);

-- 添加外键

ALTER TABLE score ADD CONSTRAINT student\_score\_fk FOREIGN KEY(stuid)

REFERENCES student(stuid);

CREATE TABLE student(

stuid INT PRIMARY KEY,

sname VARCHAR(20)

);

DELETE FROM student WHERE stuid=1001;

# Mysql高级（存储过程，视图，索引）

# 存储过程

/\*

解释：

将多个SQL语句封装到一起，这样的一个SQL脚本就是存储过程，

存储过程类似于函数

语法：

create procedure 存储过程名称(参数)

begin

SQL语句

end

delimiter 声明，存储过程结束符号的声明

存储过程默认是以;结束

\*/

-- 案例1：创建一个简单的存储过程

DROP PROCEDURE myp1; #删除存储过程

DELIMITER $ # 设置存储过程的结束符号

CREATE PROCEDURE myp1()

BEGIN

INSERT INTO mystu VALUES(NULL,'myprocedure',99);

SELECT \* FROM mystu;

END$

-- 调用存储过程

CALL myp1();

-- 案例2：创建带有参数的存储过程

DROP PROCEDURE myp2;

DELIMITER $ # 设置存储过程的结束符号

CREATE PROCEDURE myp2(cj INT,NAME VARCHAR(30))

BEGIN

INSERT INTO mystu VALUES(NULL,NAME,cj);

SELECT \* FROM mystu;

END$

CALL myp2(78,'lisi');

-- 案例3：加入判断条件的存储过程

DELIMITER $

# 当flag为1的时候，统计工资10000以上员工的个数

# 当flag为其它值得时候，统计工资小于或等于10000的员工个数

DELIMITER $

CREATE PROCEDURE myp4(flag INT)

BEGIN

IF flag=1 THEN

SELECT CONCAT("工资大于10000的员工有 ",COUNT(\*)," 个") FROM emp WHERE sal > 10000;

ELSE

SELECT CONCAT("工资小于等于10000的员工有 ",COUNT(\*)," 个") FROM emp WHERE sal <= 10000;

END IF;# if也需要使用end if结束

END$

CALL myp4(0);

/\*

delimiter $

create procedure myp3(page int,pageSize int)

begin

declare sta int;

set sta = (page-1)\*pageSize;

select \* from emp limit sta,pageSize;

end$

call myp3(3,3);

\*/

-- 案例4：在存储过程中加入循环

-- 计算1到N的和

DELIMITER $

CREATE PROCEDURE myp5(n INT)

BEGIN

DECLARE i INT;# 声明一个变量 用来迭代

DECLARE he INT;# 声明一个变量，用来存储和

SET i = 1;

SET he = 0;

WHILE i<=n DO

SET he = he + i;

SET i=i+1;

END WHILE;

SELECT he;#查询总和

END$

CALL myp5(10);

-- 索引

/\*

索引主要是提高检索效率（查询）

底层原理：

给数据进行分类，加上目录

\*/

-- 创建索引

CREATE INDEX ename\_index ON emp(ename(50));

ALTER TABLE emp ADD INDEX name\_index(ename(50));

-- 删除索引

DROP INDEX ename\_index ON emp;

-- 视图

/\*

视图就是将查询结果封装，但是是一个虚拟表

语法：

create view 视图名

as 查询语句;

\*/

CREATE OR REPLACE VIEW emp\_dept\_v

AS SELECT d.dname,d.loc # 注释

FROM emp e

JOIN dept d

ON e.deptno=d.deptno;

-- 查询视图和查询表的语法一致

SELECT \* FROM emp\_dept\_v;

-- 删除视图

DROP VIEW emp\_dept\_v;

-- 查看当前数据库使用的编码格式

SHOW VARIABLES LIKE 'char%';

# MYSQL导入导出数据

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。

导出数据库：

mysqldump -uroot -p密码 数据库名>SQL脚本存放的位置和文件名

导入数据库脚本:

Mysql -uroot -p密码 数据库名<SQL脚本文件