

本单元目标

一、为什么要学习数据库	
二、数据库的相关概念	
DBMS、DB、SQL	
三、数据库存储数据的特点	
四、初始MySQL	
MySQL产品的介绍	
MySQL产品的安装	★
MySQL服务的启动和停止	★
MySQL服务的登录和退出	★
MySQL的常见命令和语法规范	
五、DQL语言的学习	★
基础查询	★
条件查询	★
排序查询	★
常见函数	★
分组函数	★
分组查询	★
连接查询	★
子查询	√
分页查询	★
union联合查询	√
六、DML语言的学习	★
插入语句	
修改语句	
删除语句	
七、DDL语言的学习	
库和表的管理	√
常见数据类型介绍	√
常见约束	√
八、TCL语言的学习	
事务和事务处理	
九、视图的讲解	√
十、变量	
十一、存储过程和函数	
十二、流程控制结构	

数据库的好处

1.持久化数据到本地
2.可以实现结构化查询，方便管理

数据库相关概念

- 1、DB：数据库，保存一组有组织的数据的容器
- 2、DBMS：数据库管理系统，又称为数据库软件（产品），用于管理DB中的数据
- 3、SQL：结构化查询语言，用于和DBMS通信的语言

数据库存储数据的特点

- 1、将数据放到表中，表再放到库中
- 2、一个数据库中可以有多个表，每个表都有一个的名字，用来标识自己。表名具有唯一性。
- 3、表具有一些特性，这些特性定义了数据在表中如何存储，类似java中“类”的设计。
- 4、表由列组成，我们也称为字段。所有表都是由一个或多个列组成的，每一列类似java中的“属性”
- 5、表中的数据是按行存储的，每一行类似于java中的“对象”。

MySQL产品的介绍和安装

MySQL服务的启动和停止

方式一：计算机—右击管理—服务
方式二：通过管理员身份运行
`net start 服务名`（启动服务）
`net stop 服务名`（停止服务）

MySQL服务的登录和退出

方式一：通过mysql自带的客户端
只限于root用户

方式二：通过windows自带的客户端
登录：
`mysql 【-h主机名 -P端口号】 -u用户名 -p密码`

退出：
`exit`或`ctrl+C`

MySQL的常见命令

1. 查看当前所有的数据库
`show databases;`
2. 打开指定的库
`use 库名`
3. 查看当前库的所有表
`show tables;`
4. 查看其它库的所有表
`show tables from 库名;`
5. 创建表
`create table 表名(`

`列名 列类型,`

```
列名 列类型,  
    。 。 。  
);  
6.查看表结构  
desc 表名;
```

7.查看服务器的版本

方式一：登录到mysql服务端

```
select version();
```

方式二：没有登录到mysql服务端

```
mysql --version
```

或

```
mysql --V
```

MySQL的语法规范

- 1.不区分大小写,但建议关键字大写,表名、列名小写
- 2.每条命令最好用分号结尾
- 3.每条命令根据需要,可以进行缩进 或换行
- 4.注释
单行注释: #注释文字
单行注释: -- 注释文字
多行注释: /* 注释文字 */

SQL的语言分类

DQL (Data Query Language) : 数据查询语言
select
DML(Data Manipulate Language):数据操作语言
insert 、update、delete
DDL (Data Define Language) : 数据定义语言
create、drop、alter
TCL (Transaction Control Language) : 事务控制语言
commit、rollback

SQL的常见命令

```
show databases; 查看所有的数据库  
use 库名; 打开指定 的库  
show tables ; 显示库中的所有表  
show tables from 库名;显示指定库中的所有表  
create table 表名(  
    字段名 字段类型,  
    字段名 字段类型  
); 创建表  
  
desc 表名; 查看指定表的结构  
select * from 表名;显示表中的所有数据
```

DQL语言的学习

进阶1：基础查询

语法：

```
SELECT 要查询的东西  
【FROM 表名】；
```

类似于Java中：`System.out.println(要打印的东西)`；

特点：

- ①通过select查询完的结果，是一个虚拟的表格，不是真实存在
- ②要查询的东西可以是常量值、可以是表达式、可以是字段、可以是函数

进阶2：条件查询

条件查询：根据条件过滤原始表的数据，查询到想要的结果

语法：

```
select  
    要查询的字段|表达式|常量值|函数  
from  
    表  
where  
    条件；
```

分类：

一、条件表达式

示例：`salary>10000`

条件运算符：

`> < >= <= = != <>`

二、逻辑表达式

示例：`salary>10000 && salary<20000`

逻辑运算符：

and (&&)：两个条件如果同时成立，结果为true，否则为false

or (||)：两个条件只要有一个成立，结果为true，否则为false

not(!)：如果条件成立，则not后为false，否则为true

三、模糊查询

示例：`last_name like 'a%'`

进阶3：排序查询

语法：

```
select  
    要查询的东西  
from  
    表
```

where

条件

order by 排序的字段|表达式|函数|别名 【asc|desc】

进阶4：常见函数

一、单行函数

1、字符函数

concat拼接

substr截取子串

upper转换成大写

lower转换成小写

trim去前后指定的空格和字符

ltrim去左边空格

rtrim去右边空格

replace替换

lpad左填充

rpadd右填充

instr返回子串第一次出现的索引

length 获取字节个数

2、数学函数

round 四舍五入

rand 随机数

floor向下取整

ceil向上取整

mod取余

truncate截断

3、日期函数

now当前系统日期+时间

curdate当前系统日期

curtime当前系统时间

str_to_date 将字符转换成日期

date_format将日期转换成字符

4、流程控制函数

if 处理双分支

case语句 处理多分支

情况1：处理等值判断

情况2：处理条件判断

5、其他函数

version版本

database当前库

user当前连接用户

二、分组函数

sum 求和

max 最大值

min 最小值

avg 平均值
count 计数

特点:

- 1、以上五个分组函数都忽略null值，除了count(*)
- 2、sum和avg一般用于处理数值型
max、min、count可以处理任何数据类型
- 3、都可以搭配distinct使用，用于统计去重后的结果
- 4、count的参数可以支持：
字段、*、常量值，一般放1

建议使用 count(*)

进阶5：分组查询

语法:

```
select 查询的字段, 分组函数  
from 表  
group by 分组的字段
```

特点:

- 1、可以按单个字段分组
- 2、和分组函数一同查询的字段最好是分组后的字段
- 3、分组筛选

	针对的表	位置	关键字
分组前筛选:	原始表	group by的前面	where
分组后筛选:	分组后的结果集	group by的后面	having

- 4、可以按多个字段分组，字段之间用逗号隔开
- 5、可以支持排序
- 6、having后可以支持别名

进阶6：多表连接查询

笛卡尔乘积：如果连接条件省略或无效则会出现
解决办法：添加上连接条件

一、传统模式下的连接：等值连接——非等值连接

1. 等值连接的结果 = 多个表的交集
2. n表连接，至少需要n-1个连接条件
3. 多个表不分主次，没有顺序要求
4. 一般为表起别名，提高阅读性和性能

二、sql99语法：通过join关键字实现连接

含义：1999年推出的sql语法

支持：

等值连接、非等值连接 （内连接）

外连接

交叉连接

语法：

```
select 字段, ...  
from 表1  
【inner|left outer|right outer|cross】join 表2 on 连接条件  
【inner|left outer|right outer|cross】join 表3 on 连接条件  
【where 筛选条件】  
【group by 分组字段】  
【having 分组后的筛选条件】  
【order by 排序的字段或表达式】
```

好处：语句上，连接条件和筛选条件实现了分离，简洁明了！

三、自连接

案例：查询员工名和直接上级的名称

sql99

```
SELECT e.last_name,m.last_name  
FROM employees e  
JOIN employees m ON e.`manager_id`=m.`employee_id`;
```

sql92

```
SELECT e.last_name,m.last_name  
FROM employees e,employees m  
WHERE e.`manager_id`=m.`employee_id`;
```

进阶7：子查询

含义：

一条查询语句中又嵌套了另一条完整的select语句，其中被嵌套的select语句，称为子查询或内查询
在外面的查询语句，称为主查询或外查询

特点：

- 1、子查询都放在小括号内
- 2、子查询可以放在from后面、select后面、where后面、having后面，但一般放在条件的右侧
- 3、子查询优先于主查询执行，主查询使用了子查询的执行结果
- 4、子查询根据查询结果的行数不同分为以下两类：
 - ① 单行子查询
结果集只有一行

一般搭配单行操作符使用：> < = <> >= <=

非法使用子查询的情况：

- a、子查询的结果为一组值
- b、子查询的结果为空

② 多行子查询

结果集有多行

一般搭配多行操作符使用：any、all、in、not in

in： 属于子查询结果中的任意一个就行

any和all往往可以用其他查询代替

进阶8：分页查询

应用场景：

实际的web项目中需要根据用户的需求提交对应的分页查询的sql语句

语法：

```
select 字段|表达式,...  
from 表  
【where 条件】  
【group by 分组字段】  
【having 条件】  
【order by 排序的字段】  
limit 【起始的条目索引，】条目数；
```

特点：

- 1.起始条目索引从0开始
- 2.limit子句放在查询语句的最后
- 3.公式: `select * from 表 limit (page-1)*sizePerPage,sizePerPage`
假如：
每页显示条目数sizePerPage
要显示的页数 page

进阶9：联合查询

引入：union 联合、合并

语法：

```
select 字段|常量|表达式|函数 【from 表】 【where 条件】 union 【all】  
select 字段|常量|表达式|函数 【from 表】 【where 条件】 union 【all】  
select 字段|常量|表达式|函数 【from 表】 【where 条件】 union 【all】  
.....  
select 字段|常量|表达式|函数 【from 表】 【where 条件】
```


特点：

- 1、多条查询语句的查询的列数必须是一致的
- 2、多条查询语句的查询的列的类型几乎相同
- 3、union代表去重，union all代表不去重

DML语言

插入

语法：insert into 表名(字段名 , ...) values(值1 , ...);

特点：

- 1、字段类型和值类型一致或兼容，而且一一对应
- 2、可以为空的字段，可以不用插入值，或用null填充
- 3、不可以为空的字段，必须插入值
- 4、字段个数和值的个数必须一致
- 5、字段可以省略，但默认所有字段，并且顺序和表中的存储顺序一致

修改

修改单表语法：

```
update 表名 set 字段=新值, 字段=新值  
【where 条件】
```

修改多表语法：

```
update 表1 别名1, 表2 别名2  
set 字段=新值, 字段=新值  
where 连接条件  
and 筛选条件
```

删除

方式1：delete语句

单表的删除：★ delete from 表名 【where 筛选条件】

多表的删除：delete 别名1 , 别名2 from 表1 别名1 , 表2 别名2 where 连接条件 and 筛选条件;

方式2：truncate语句

```
truncate table 表名
```

两种方式的区别【面试题】

#1.truncate不能加where条件，而delete可以加where条件

#2.truncate的效率高一丢丢

#3.truncate 删除带自增长的列的表后，如果再插入数据，数据从1开始

#delete 删除带自增长列的表后，如果再插入数据，数据从上一次的断点处开始

#4.truncate删除不能回滚，delete删除可以回滚

DDL语句

库和表的管理

库的管理：

一、创建库

```
create database 库名
```

二、删除库

```
drop database 库名
```

表的管理：#1.创建表

```
CREATE TABLE IF NOT EXISTS stuinfo(  
    stuId INT,  
    stuName VARCHAR(20),  
    gender CHAR,  
    bornDate DATETIME  
  
);
```

```
DESC studentinfo;
```

#2.修改表 alter

语法：ALTER TABLE 表名 ADD|MODIFY|DROP|CHANGE COLUMN 字段名 【字段类型】；

#①修改字段名

```
ALTER TABLE studentinfo CHANGE COLUMN sex gender CHAR;
```

#②修改表名

```
ALTER TABLE stuinfo RENAME [TO] studentinfo;
```

#③修改字段类型和列级约束

```
ALTER TABLE studentinfo MODIFY COLUMN borndate DATE ;
```

#④添加字段

```
ALTER TABLE studentinfo ADD COLUMN email VARCHAR(20) first;
```

#⑤删除字段

```
ALTER TABLE studentinfo DROP COLUMN email;
```

#3. 删除表

```
DROP TABLE [IF EXISTS] studentinfo;
```

常见类型

整型：

小数：

浮点型

定点型

字符型：

日期型：

Blob类型：

常见约束

NOT NULL

DEFAULT

UNIQUE

CHECK

PRIMARY KEY

FOREIGN KEY

数据库事务

含义

通过一组逻辑操作单元（一组DML—sql语句），将数据从一种状态切换到另外一种状态

特点

（ACID）

原子性：要么都执行，要么都回滚

一致性：保证数据的状态操作前和操作后保持一致

隔离性：多个事务同时操作相同数据库的同一个数据时，一个事务的执行不受另外一个事务的干扰

持久性：一个事务一旦提交，则数据将持久化到本地，除非其他事务对其进行修改

相关步骤：

1、开启事务

2、编写事务的一组逻辑操作单元（多条sql语句）

3、提交事务或回滚事务

事务的分类：

隐式事务，没有明显的开启和结束事务的标志

比如

insert、update、delete语句本身就是一个事务

显式事务，具有明显的开启和结束事务的标志

1、开启事务

取消自动提交事务的功能

2、编写事务的一组逻辑操作单元（多条sql语句）

insert

update

delete

3、提交事务或回滚事务

使用到的关键字

```
set autocommit=0;  
start transaction;  
commit;  
rollback;
```

savepoint 断点

commit to 断点

rollback to 断点

事务的隔离级别：

事务并发问题如何发生？

当多个事务同时操作同一个数据库的相同数据时

事务的并发问题有哪些？

脏读：一个事务读取到了另外一个事务未提交的数据

不可重复读：同一个事务中，多次读取到的数据不一致

幻读：一个事务读取数据时，另外一个事务进行更新，导致第一个事务读取到了没有更新的数据

如何避免事务的并发问题？

通过设置事务的隔离级别

1、READ UNCOMMITTED

2、READ COMMITTED 可以避免脏读

3、REPEATABLE READ 可以避免脏读、不可重复读和一部分幻读

4、SERIALIZABLE可以避免脏读、不可重复读和幻读

设置隔离级别：

```
set session|global transaction isolation level 隔离级别名;
```

查看隔离级别：

```
select @@tx_isolation;
```

视图

含义：理解成一张虚拟的表

视图和表的区别：

	使用方式	占用物理空间
视图	完全相同	不占用，仅仅保存的是sql逻辑
表	完全相同	占用

视图的好处：

- 1、sql语句提高重用性，效率高
- 2、和表实现了分离，提高了安全性

视图的创建

语法：

```
CREATE VIEW 视图名
```

```
AS
```

```
查询语句;
```

视图的增删改查

- 1、查看视图的数据 ★

```
SELECT * FROM my_v4;
```

```
SELECT * FROM my_v1 WHERE last_name='Partners';
```

- 2、插入视图的数据

```
INSERT INTO my_v4(last_name,department_id) VALUES('虚竹',90);
```

- 3、修改视图的数据

```
UPDATE my_v4 SET last_name ='梦姑' WHERE last_name='虚竹';
```

- 4、删除视图的数据

```
DELETE FROM my_v4;
```

某些视图不能更新

包含以下关键字的sql语句：分组函数、distinct、group by、having、union或者union all
常量视图
Select中包含子查询
join
from一个不能更新的视图
where子句的子查询引用了from子句中的表

视图逻辑的更新

```
#方式一：
CREATE OR REPLACE VIEW test_v7
AS
SELECT last_name FROM employees
WHERE employee_id>100;

#方式二：
ALTER VIEW test_v7
AS
SELECT employee_id FROM employees;

SELECT * FROM test_v7;
```

视图的删除

```
DROP VIEW test_v1,test_v2,test_v3;
```

视图结构的查看

```
DESC test_v7;
SHOW CREATE VIEW test_v7;
```

存储过程

含义：一组经过预先编译的sql语句的集合 好处：

- 1、提高了sql语句的重用性，减少了开发程序员的压力
- 2、提高了效率
- 3、减少了传输次数

分类：

- 1、无返回无参
 - 2、仅仅带in类型，无返回有参
 - 3、仅仅带out类型，有返回无参
 - 4、既带in又带out，有返回有参
 - 5、带inout，有返回有参
- 注意：in、out、inout都可以在一个存储过程中带多个

创建存储过程

语法：

```
create procedure 存储过程名(in|out|inout 参数名 参数类型,...)
begin
    存储过程体

end
```

类似于方法：

```
修饰符 返回类型 方法名(参数类型 参数名,...){

    方法体;

}
```

注意

1、需要设置新的结束标记

delimiter 新的结束标记

示例：

delimiter \$

CREATE PROCEDURE 存储过程名(**IN|OUT|INOUT** 参数名 参数类型,...)

BEGIN

sql语句1;

sql语句2;

END \$

2、存储过程体中可以有多条sql语句，如果仅仅一条sql语句，则可以省略**begin end**

3、参数前面的符号的意思

in: 该参数只能作为输入（该参数不能做返回值）

out: 该参数只能作为输出（该参数只能做返回值）

inout: 既能做输入又能做输出

调用存储过程

```
call 存储过程名(实参列表)
```

函数

创建函数

学过的函数：LENGTH、SUBSTR、CONCAT等 语法：

```
CREATE FUNCTION 函数名(参数名 参数类型,...) RETURNS 返回类型
BEGIN
    函数体

END
```

调用函数

```
SELECT 函数名 (实参列表)
```

函数和存储过程的区别

	关键字	调用语法	返回值	应用场景
函数	FUNCTION	SELECT 函数()	只能是一个	一般用于查询结果为一个值并返回时，当有返回值而且仅仅一个
存储过程	PROCEDURE	CALL 存储过程()	可以有0个或多个	一般用于更新

流程控制结构

系统变量

一、全局变量

作用域：针对于所有会话（连接）有效，但不能跨重启

```
查看所有全局变量
SHOW GLOBAL VARIABLES;
查看满足条件的部分系统变量
SHOW GLOBAL VARIABLES LIKE '%char%';
查看指定的系统变量的值
SELECT @@global.autocommit;
为某个系统变量赋值
SET @@global.autocommit=0;
SET GLOBAL autocommit=0;
```

二、会话变量

作用域：针对于当前会话（连接）有效

```
查看所有会话变量
SHOW SESSION VARIABLES;
查看满足条件的部分会话变量
SHOW SESSION VARIABLES LIKE '%char%';
查看指定的会话变量的值
SELECT @@autocommit;
SELECT @@session.tx_isolation;
为某个会话变量赋值
```



```
SET @@session.tx_isolation='read-uncommitted';  
SET SESSION tx_isolation='read-committed';
```

自定义变量

一、用户变量

声明并初始化：

```
SET @变量名=值;  
SET @变量名:=值;  
SELECT @变量名:=值;
```

赋值：

方式一：一般用于赋简单的值

```
SET 变量名=值;  
SET 变量名:=值;  
SELECT 变量名:=值;
```

方式二：一般用于赋表 中的字段值

```
SELECT 字段名或表达式 INTO 变量  
FROM 表;
```

使用：

```
select @变量名;
```

二、局部变量

声明：

```
declare 变量名 类型 【default 值】;
```

赋值：

方式一：一般用于赋简单的值

```
SET 变量名=值;  
SET 变量名:=值;  
SELECT 变量名:=值;
```

方式二：一般用于赋表 中的字段值

```
SELECT 字段名或表达式 INTO 变量  
FROM 表;
```

使用：

```
select 变量名
```



```
【标签：】WHILE 循环条件 DO  
    循环体  
END WHILE 【标签】；
```

特点：

只能放在BEGIN END里面

如果要搭配leave跳转语句，需要使用标签，否则可以不用标签

leave类似于java中的break语句，跳出所在循环！！