

MySQL

初识MySQL

为什么学习数据库

- 1、岗位技能需求
- 2、现在的世界,得数据者得天下
- 3、存储数据的方法
- 4、程序,网站中,大量数据如何长久保存?
- 5、数据库是几乎软件体系中最核心的一个存在。

什么是数据库

数据库 (**DataBase** , 简称**DB**)

概念 : 长期存放在计算机内,有组织,可共享的大量数据的集合,是一个数据 "仓库"

作用 : 保存,并能安全管理数据(如:增删改查等),减少冗余...

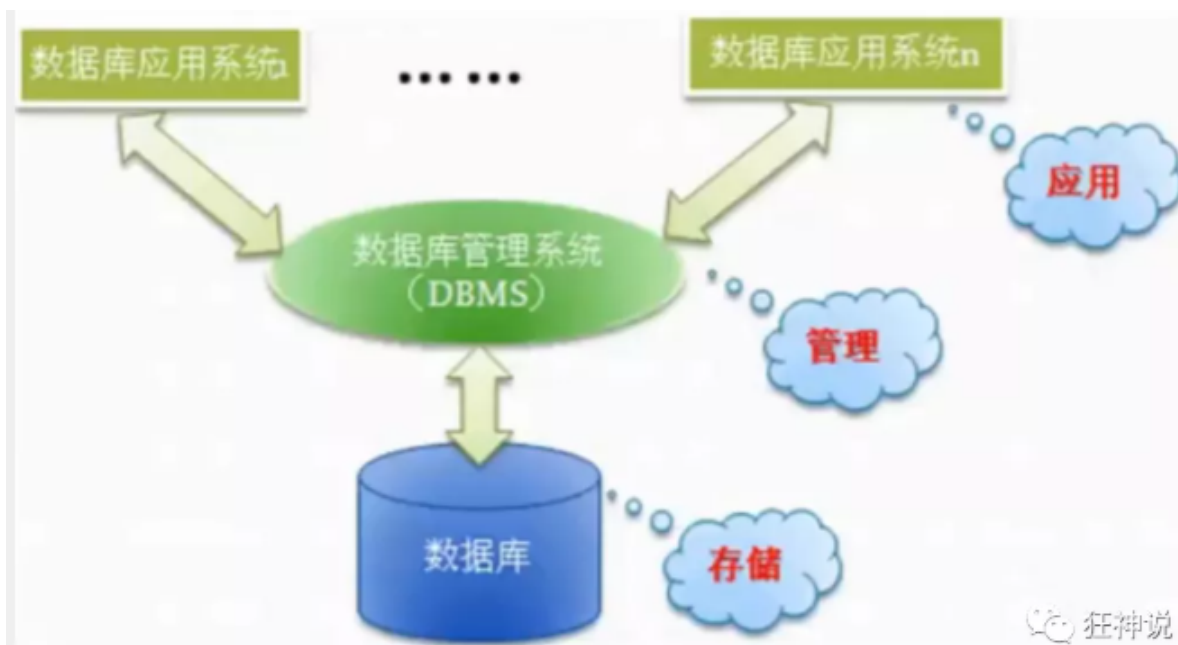
数据库总览 :

- 关系型数据库 (SQL)
 - MySQL , Oracle , SQL Server , SQLite , DB2 , ...
 - 关系型数据库通过外键关联来建立表与表之间的关系
- 非关系型数据库 (NOSQL)
 - Redis , MongoDB , ...
 - 非关系型数据库通常指数据以对象的形式存储在数据库中, 而对象之间的关系通过每个对象自身的属性来决定

什么是DBMS

数据库管理系统 (**DataBase Management System**)

数据库管理软件, 科学组织和存储数据, 高效地获取和维护数据



为什么要说这个呢?

因为我们要学习的MySQL应该算是一个数据库管理系统.

MySQL简介

概念: 是现在流行的开源的,免费的 关系型数据库

历史: 由瑞典MySQL AB 公司开发, 目前属于 Oracle 旗下产品。

特点:

- 免费, 开源数据库
- 小巧, 功能齐全
- 使用便捷
- 可运行于Windows或Linux操作系统
- 可适用于中小型甚至大型网站应用

官网: <https://www.mysql.com/>

安装MySQL

这里建议大家使用压缩版, 安装快, 方便. 不复杂.

软件下载

mysql5.7 64位下载地址:

<https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-5.7.19-winx64.zip>

电脑是64位的就下载使用64位版本的!

安装步骤

- 1、下载后得到zip压缩包.
- 2、解压到自己想要安装到的目录, 本人解压到的是D:\Environment\mysql-5.7.19
- 3、添加环境变量: 我的电脑->属性->高级->环境变量

选择PATH, 在其后面添加: 你的mysql 安装文件下面的bin文件夹

4、编辑 my.ini 文件 ,注意替换路径位置

```
[mysqld]
basedir=D:\Program Files\mysql-5.7\
datadir=D:\Program Files\mysql-5.7\data\
port=3306
skip-grant-tables
```

5、启动管理员模式下的CMD，并将路径切换至mysql下的bin目录，然后输入mysql -install (安装mysql)

6、再输入 mysql -initialize-insecure --user=mysql 初始化数据文件

7、然后再次启动mysql 然后用命令 mysql -u root -p 进入mysql管理界面（密码可为空）

8、进入界面后更改root密码

```
update mysql.user set authentication_string=password('123456') where user='root'
and Host = 'localhost';
```

9、刷新权限

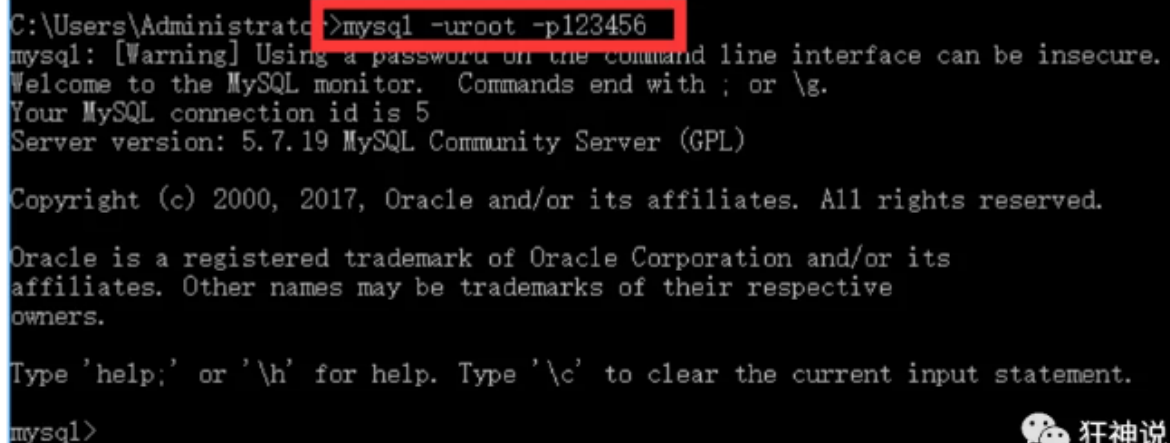
```
flush privileges;
```

10、修改 my.ini文件删除最后一句skip-grant-tables

11、重启mysql即可正常使用

```
net stop mysql
net start mysql
```

12、连接上测试出现以下结果就安装好了



```
C:\Users\Administrator>mysql -uroot -p123456
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.19 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

一步步去做，理论上没有任何问题的。

如果您以前装过,现在需要重装,一定要将环境清理干净。

好了,到这里大家都装好了,因为刚接触,所以我们先不学习命令。

这里给大家推荐一个工具：SQLyog。

即便有了可视化工具,可是基本的DOS命名大家还是要记住!

SQLyog

可手动操作,管理MySQL数据库的软件工具

特点:简洁,易用,图形化

使用SQLyog管理工具自己完成以下操作:

- 连接本地MySQL数据库
- 新建MySchool数据库
 - 字段
 - GradeID : int(11) , Primary Key (pk)
 - GradeName : varchar(50)
 - 数据库名称MySchool
 - 新建数据库表(grade)

在历史记录中可以看到相对应的数据库操作的语句

连接数据库

打开MySQL命令窗口

- 在DOS命令行窗口进入 **安装目录\mysql\bin**
- 可设置环境变量, 设置了环境变量, 可以在任意目录打开!

连接数据库语句: mysql -h 服务器主机地址 -u 用户名 -p 用户密码

注意: -p后面不能加空格,否则会被当做密码的内容,导致登录失败!

几个基本的数据库操作命令:

```
update user set password=password('123456')where user='root'; 修改密码
flush privileges; 刷新数据库
show databases; 显示所有数据库
use dbname; 打开某个数据库
show tables; 显示数据库mysql中所有的表
describe user; 显示表mysql数据库中user表的列信息
create database name; 创建数据库
use databasename; 选择数据库

exit; 退出Mysql
? 命令关键词 : 寻求帮助
-- 表示注释
```

数据库操作

名称	解释	命令
DDL (数据定义语言)	定义和管理数据对象,如数据库,数据表等	CREATE、DROP、ALTER
DML (数据操作语言)	用于操作数据库对象中所包含的数据	INSERT、UPDATE、DELETE
DQL (数据查询语言)	用于查询数据库数据	SELECT
DCL (数据控制语言)	用于管理数据库的语言,包括管理权限及数据更改	GRANT、commit、rollback

大神说

命令行操作数据库

创建数据库: create database [if not exists] 数据库名;

删除数据库：drop database [if exists] 数据库名;

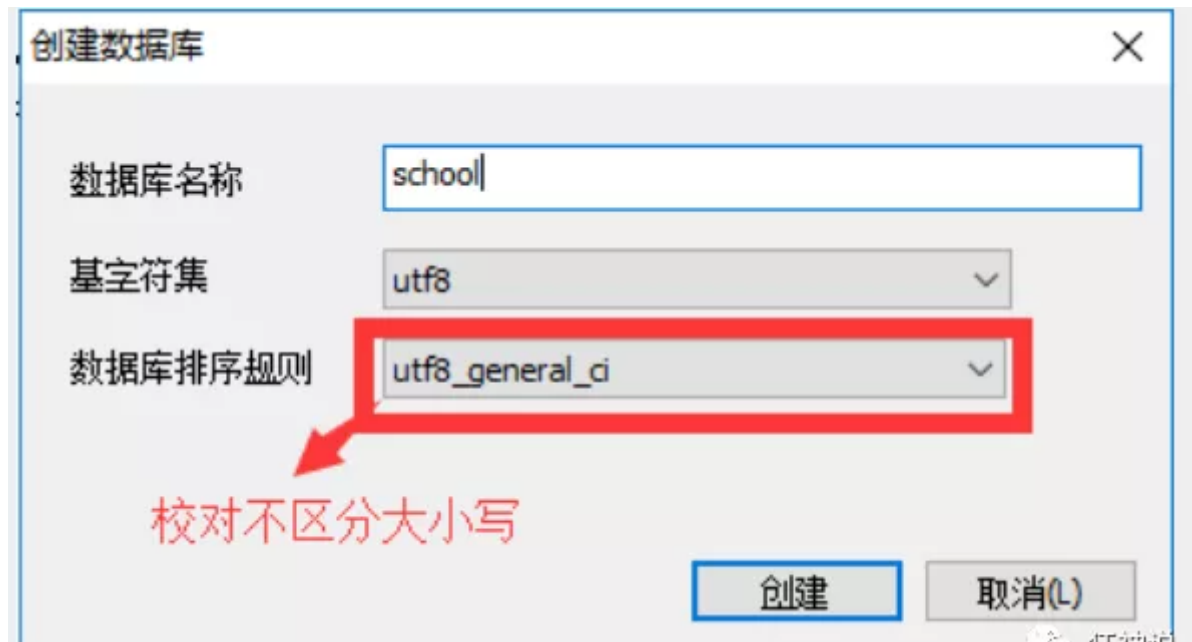
查看数据库：show databases;

使用数据库：use 数据库名;

对比工具操作数据库

学习方法：

- 对照SQLyog工具自动生成的语句学习
- 固定语法中的单词需要记忆



创建数据表

属于DDL的一种，语法：

```
create table [if not exists] `表名` (  
    '字段名1' 列类型 [属性] [索引] [注释],  
    '字段名2' 列类型 [属性] [索引] [注释],  
    #...  
    '字段名n' 列类型 [属性] [索引] [注释]  
) [表类型] [表字符集] [注释];
```

说明：反引号用于区别MySQL保留字与普通字符而引入的 (键盘esc下面的键).

数据值和列类型

列类型：规定数据库中该列存放的数据类型

数值类型

类型	说明	取值范围	存储需求
tinyint	非常小的数据	有符值: $-2^7 \sim 2^7-1$ 无符号值: $0 \sim 2^8-1$	1字节
smallint	较小的数据	有符值: $-2^{15} \sim 2^{15}-1$ 无符号值: $0 \sim 2^{16}-1$	2字节
mediumint	中等大小的数据	有符值: $-2^{23} \sim 2^{23}-1$ 无符号值: $0 \sim 2^{24}-1$	3字节
int	标准整数	有符值: $-2^{31} \sim 2^{31}-1$ 无符号值: $0 \sim 2^{32}-1$	4字节
bigint	较大的整数	有符值: $-2^{63} \sim 2^{63}-1$ 无符号值: $0 \sim 2^{64}-1$	8字节
float	单精度浮点数	$\pm 1.1754351e-38$	4字节
double	双精度浮点数	$\pm 2.2250738585072014e-308$	8字节
decimal	字符串形式的浮点数	decimal(m, d)	个字节 狂神说

字符串类型

类型	说明	最大长度
char [(M)]	固定长字符串, 检索快但费空间, $0 \leq M \leq 255$	M字符
varchar [(M)]	可变字符串 $0 \leq M \leq 65535$	变长度
tinytext	微型文本串	$2^8 - 1$ 字节
text	文本串	$2^{16} - 1$ 字节

日期和时间型数值类型

类型	说明	取值范围
DATE	YYYY-MM-DD, 日期格式	1000-01-01~ 9999-12-31
TIME	Hh:mm:ss , 时间格式	-838:59:59~838:59:59
DATETIME	YY-MM-DD hh:mm:ss	1000-01-01 00:00:00 至 9999-12-31 23:59:59
TIMESTAMP	YYYYMMDDhhmmss格式表示的时间戳	197010101000000 ~2037年的某个时刻
YEAR	YYYY格式的年份值	1901~2155

NULL值

- 理解为 "没有值" 或 "未知值"
- 不要用NULL进行算术运算, 结果仍为NULL

数据字段属性

Unsigned

- 无符号的
- 声明该数据列不允许负数.

ZEROFILL

- 0填充的
- 不足位数的用0来填充, 如int(3),5则为005

Auto_InCrement

- 自动增长的, 每添加一条数据, 自动在上一个记录数上加 1(默认)
- 通常用于设置**主键**, 且为整数类型
- 可定义起始值和步长
 - 当前表设置步长(AUTO_INCREMENT=100): 只影响当前表
 - SET @@auto_increment_increment=5; 影响所有使用自增的表(全局)

NULL 和 NOT NULL

- 默认为NULL, 即没有插入该列的数值
- 如果设置为NOT NULL, 则该列必须有值

DEFAULT

- 默认的
- 用于设置默认值
- 例如, 性别字段, 默认为"男", 否则为 "女"; 若无指定该列的值, 则默认值为"男"的值

```
-- 创建学生表(列,字段)
-- 学号int 登录密码varchar(20) 姓名,性别varchar(2),出生日期(datetime),家庭住址,email
-- 创建表之前 , 一定要先选择数据库

CREATE TABLE IF NOT EXISTS `student` (
  `id` int(4) NOT NULL AUTO_INCREMENT COMMENT '学号',
  `name` varchar(30) NOT NULL DEFAULT '匿名' COMMENT '姓名',
  `pwd` varchar(20) NOT NULL DEFAULT '123456' COMMENT '密码',
  `sex` varchar(2) NOT NULL DEFAULT '男' COMMENT '性别',
  `birthday` datetime DEFAULT NULL COMMENT '生日',
  `address` varchar(100) DEFAULT NULL COMMENT '地址',
  `email` varchar(50) DEFAULT NULL COMMENT '邮箱',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8

-- 查看数据库的定义
SHOW CREATE DATABASE school;
-- 查看数据表的定义
SHOW CREATE TABLE student;
-- 显示表结构
DESC student; -- 设置严格检查模式(不能容错了)SET sql_mode='STRICT_TRANS_TABLES';
```

数据表的类型

设置数据表的类型

```
CREATE TABLE 表名(
  -- 省略一些代码
  -- Mysql注释
  -- 1. # 单行注释
  -- 2. /*...*/ 多行注释
)ENGINE = MyISAM (or InnoDB)

-- 查看mysql所支持的引擎类型 (表类型)
SHOW ENGINES;
```

MySQL的数据表的类型: **MyISAM** , **InnoDB** , HEAP , BOB , CSV等...

常见的 MyISAM 与 InnoDB 类型:

名称	MyISAM	InnoDB
事务处理	不支持	支持
数据行锁定	不支持	支持
外键约束	不支持	支持
全文索引	支持	不支持
表空间大小	教小	较大, 约 2 倍!

经验 (适用场合) :

- 适用 MyISAM : 节约空间及相应速度
- 适用 InnoDB : 安全性, 事务处理及多用户操作数据表

数据表的存储位置

- MySQL数据表以文件方式存放在磁盘中
- - 包括表文件，数据文件，以及数据库的选项文件
 - 位置：Mysql安装目录\data\下存放数据表。目录名对应数据库名，该目录下文件名对应数据表
- 注意：
- - *.frm -- 表结构定义文件
 - *.MYD -- 数据文件 (data)
 - *.MYI -- 索引文件 (index)
 - InnoDB类型数据表只有一个 *.frm文件，以及上一级目录的ibdata1文件
 - MyISAM类型数据表对应三个文件：

名称	类型
innodb.frm	FRM 文件
myisam.frm	FRM 文件
myisam.MYD	MYD 文件
myisam.MYI	MYI 文件

设置数据表字符集

我们可为数据库,数据表,数据列设定不同的字符集，设定方法：

- 创建时通过命令来设置，如：CREATE TABLE 表名()CHARSET = utf8;
- 如无设定，则根据MySQL数据库配置文件 my.ini 中的参数设定

修改数据库

修改表 (ALTER TABLE)

修改表名：ALTER TABLE 旧表名 RENAME AS 新表名

添加字段：ALTER TABLE 表名 ADD 字段名 列属性[属性]

修改字段：

- ALTER TABLE 表名 MODIFY 字段名 列类型[属性]
- ALTER TABLE 表名 CHANGE 旧字段名 新字段名 列属性[属性]

删除字段：ALTER TABLE 表名 DROP 字段名

删除数据表

语法：DROP TABLE [IF EXISTS] 表名

- IF EXISTS为可选，判断是否存在该数据表
- 如删除不存在的数据表会抛出错误

其他

1. 可用反引号 (`) 为标识符 (库名、表名、字段名、索引、别名) 包裹，以避免与关键字重名！中文也可以作为标识符！
2. 每个库目录存在一个保存当前数据库的选项文件db.opt。

3. 注释：

单行注释 `# 注释内容`

多行注释 `/* 注释内容 */`

单行注释 `-- 注释内容` （标准SQL注释风格，要求双破折号后加一空格符（空格、TAB、换行等））

4. 模式通配符：

`_` 任意单个字符

`%` 任意多个字符，甚至包括零字符

单引号需要进行转义 `\'`

5. CMD命令行内的语句结束符可以为 `;"`，`"\G"`，`"\g"`，仅影响显示结果。其他地方还是用分号结束。
`delimiter` 可修改当前对话的语句结束符。

6. SQL对大小写不敏感 （关键字）

7. 清除已有语句：`\c`

DML语言

外键

外键概念

如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外键。由此可见，外键表示了两个关系之间的相关联系。以另一个关系的外键作主关键字的表被称为**主表**，具有此外键的表被称为主表的**从表**。

在实际操作中，将一个表的值放入第二个表来表示关联，所使用的值是第一个表的主键值(在必要时可包括复合主键值)。此时，第二个表中保存这些值的属性称为外键(**foreign key**)。

外键作用

保持数据**一致性**，**完整性**，主要目的是控制存储在外键表中的数据**约束**。使两张表形成关联，外键只能引用外表中的列的值或使用空值。

创建外键

建表时指定外键约束

```
-- 创建外键的方式一 ： 创建子表同时创建外键

-- 年级表 (id\年级名称)
CREATE TABLE `grade` (
  `gradeid` INT(10) NOT NULL AUTO_INCREMENT COMMENT '年级ID',
  `gradename` VARCHAR(50) NOT NULL COMMENT '年级名称',
  PRIMARY KEY (`gradeid`)
) ENGINE=INNODB DEFAULT CHARSET=utf8

-- 学生信息表 (学号,姓名,性别,年级,手机,地址,出生日期,邮箱,身份证号)
CREATE TABLE `student` (
  `studentno` INT(4) NOT NULL COMMENT '学号',
  `studentname` VARCHAR(20) NOT NULL DEFAULT '匿名' COMMENT '姓名',
  `sex` TINYINT(1) DEFAULT '1' COMMENT '性别',
  `gradeid` INT(10) DEFAULT NULL COMMENT '年级',
  `phoneNum` VARCHAR(50) NOT NULL COMMENT '手机',
```

```
`address` VARCHAR(255) DEFAULT NULL COMMENT '地址',
`borndate` DATETIME DEFAULT NULL COMMENT '生日',
`email` VARCHAR(50) DEFAULT NULL COMMENT '邮箱',
`idCard` VARCHAR(18) DEFAULT NULL COMMENT '身份证号',
PRIMARY KEY (`studentno`),
KEY `FK_gradeid` (`gradeid`),
CONSTRAINT `FK_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade` (`gradeid`)
) ENGINE=INNODB DEFAULT CHARSET=utf8
```

建表后修改

```
-- 创建外键方式二：创建子表完毕后,修改子表添加外键
ALTER TABLE `student`
ADD CONSTRAINT `FK_gradeid` FOREIGN KEY (`gradeid`) REFERENCES `grade`
(`gradeid`);
```

删除外键

操作：删除 grade 表，发现报错



注意：删除具有主外键关系的表时，要先删子表，后删主表

```
-- 删除外键
ALTER TABLE student DROP FOREIGN KEY FK_gradeid;
-- 发现执行完上面的,索引还在,所以还要删除索引
-- 注:这个索引是建立外键的时候默认生成的
ALTER TABLE student DROP INDEX FK_gradeid;
```

DML语言

数据库意义： 数据存储、数据管理

管理数据库数据方法：

- 通过SQLyog等管理工具管理数据库数据
- 通过**DML语句**管理数据库数据

DML语言： 数据操作语言

- 用于操作数据库对象中所包含的数据
- 包括：
 - INSERT (添加数据语句)
 - UPDATE (更新数据语句)
 - DELETE (删除数据语句)

添加数据

INSERT命令

语法：

```
INSERT INTO 表名[(字段1,字段2,字段3,...)] VALUES('值1','值2','值3')
```

注意：

- 字段或值之间用英文逗号隔开。
- '字段1,字段2...' 该部分可省略, 但添加的值务必与表结构,数据列,顺序相对应,且数量一致。
- 可同时插入多条数据, values 后用英文逗号隔开。

```
-- 使用语句如何增加语句?
-- 语法 : INSERT INTO 表名[(字段1,字段2,字段3,...)] VALUES('值1','值2','值3')
INSERT INTO grade(gradename) VALUES ('大一');

-- 主键自增,那能否省略呢?
INSERT INTO grade VALUES ('大二');

-- 查询:INSERT INTO grade VALUE ('大二')错误代码: 1136
Column count doesn't match value count at row 1

-- 结论:'字段1,字段2...'该部分可省略, 但添加的值务必与表结构,数据列,顺序相对应,且数量一致。

-- 一次插入多条数据
INSERT INTO grade(gradename) VALUES ('大三'),('大四');
```

练习题目

自己使用INSERT语句为课程表subject添加数据。使用到外键

修改数据

update命令

语法：

```
UPDATE 表名 SET column_name=value [,column_name2=value2,...] [WHERE condition];
```

注意：

- column_name 为要更改的数据列
- value 为修改后的数据，可以为变量，具体指，表达式或者嵌套的SELECT结果
- condition 为筛选条件，如不指定则修改该表的所有列数据

where条件子句

可以简单的理解为：有条件地从表中筛选数据

运算符	含义	范围	结果
=	等于	5=6	false
<> 或 !=	不等于	5!=6	true
>	大于	5>6	false
<	小于	5<6	true
>=	大于等于	5>=6	false
<=	小于等于	5<=6	true
BETWEEN	在某个范围之内	BETWEEN 5 AND 10	
AND	并且	5 > 1 AND 1 > 2	false
OR	或	5 > 1 OR 1 > 2	true

测试：

```
-- 修改年级信息
UPDATE grade SET gradename = '高中' WHERE gradeid = 1;
```

删除数据

DELETE命令

语法：

```
DELETE FROM 表名 [WHERE condition];
```

注意：condition为筛选条件，如不指定则删除该表的所有列数据

```
-- 删除最后一个数据
DELETE FROM grade WHERE gradeid = 5
```

TRUNCATE命令

作用：用于完全清空表数据，但表结构，索引，约束等不变；

语法：

```
TRUNCATE [TABLE] table_name;

-- 清空年级表
TRUNCATE grade
```

注意：区别于DELETE命令

- 相同：都能删除数据，不删除表结构，但TRUNCATE速度更快
- 不同：
 - 使用TRUNCATE TABLE 重新设置AUTO_INCREMENT计数器
 - 使用TRUNCATE TABLE不会对事务有影响（事务后面会说）

测试：

```
-- 创建一个测试表
CREATE TABLE `test` (
  `id` INT(4) NOT NULL AUTO_INCREMENT,
  `col1` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8

-- 插入几个测试数据
INSERT INTO test(col1) VALUES('row1'),('row2'),('row3');

-- 删除表数据(不带where条件的delete)
DELETE FROM test;
-- 结论：如不指定where则删除该表的所有列数据，自增当前值依然从原来基础上进行，会记录日志。

-- 删除表数据(truncate)
TRUNCATE TABLE test;
-- 结论：truncate删除数据，自增当前值会恢复到初始值重新开始；不会记录日志。

-- 同样使用DELETE清空不同引擎的数据库表数据。重启数据库服务后
-- InnoDB：自增列从初始值重新开始（因为是存储在内存中，断电即失）
-- MyISAM：自增列依然从上一个自增数据基础上开始（存在文件中，不会丢失）
```

使用DQL查询数据

DQL(Data Query Language 数据查询语言)

- 查询数据库数据，如SELECT语句
- 简单的单表查询或多表的复杂查询和嵌套查询
- 是数据库语言中最核心,最重要的语句
- 使用频率最高的语句

SELECT语法

```

SELECT [ALL | DISTINCT]
{* | table.* | [table.field1[as alias1][,table.field2[as alias2]][,...]]}
FROM table_name [as table_alias]
    [left | right | inner join table_name2]  -- 联合查询
[WHERE ...]  -- 指定结果需满足的条件
[GROUP BY ...]  -- 指定结果按照哪几个字段来分组
[HAVING]  -- 过滤分组的记录必须满足的次要条件
[ORDER BY ...]  -- 指定查询记录按一个或多个条件排序
[LIMIT {[offset],row_count | row_countOFFSET offset}];
    -- 指定查询的记录从哪条至哪条

```

注意：[] 括号代表可选的，{} 括号代表必选得

指定查询字段

```

-- 查询表中所有的数据列结果，采用 "*" \* " 符号；但是效率低，不推荐。

-- 查询所有学生信息
SELECT * FROM student;

-- 查询指定列(学号，姓名)
SELECT studentno,studentname FROM student;

```

AS 子句作为别名

作用：

- 可给数据列取一个新别名
- 可给表取一个新别名
- 可把经计算或总结的结果用另一个新名称来代替

```

-- 这里是为列取别名(当然as关键词可以省略)
SELECT studentno AS 学号,studentname AS 姓名 FROM student;

-- 使用as也可以为表取别名
SELECT studentno AS 学号,studentname AS 姓名 FROM student AS s;

-- 使用as,为查询结果取一个新名字
-- CONCAT()函数拼接字符串
SELECT CONCAT('姓名:',studentname) AS 新姓名 FROM student;

```

DISTINCT关键字的使用

作用：去掉SELECT查询返回的记录结果中重复的记录(返回所有列的值都相同)，只返回一条

```

-- # 查看哪些同学参加了考试(学号) 去除重复项
SELECT * FROM result; -- 查看考试成绩
SELECT studentno FROM result; -- 查看哪些同学参加了考试
SELECT DISTINCT studentno FROM result; -- 了解:DISTINCT 去除重复项，(默认是ALL)

```

使用表达式的列

数据库中的表达式：一般由文本值，列值，NULL，函数和操作符等组成

应用场景：

- SELECT语句返回结果列中使用
- SELECT语句中的ORDER BY , HAVING等子句中使用
- DML语句中的 where 条件语句中使用表达式

```
-- selcet查询中可以使用表达式
SELECT @@auto_increment_increment; -- 查询自增步长
SELECT VERSION(); -- 查询版本号
SELECT 100*3-1 AS 计算结果; -- 表达式

-- 学员考试成绩集体提分一分查看
SELECT studentno,StudentResult+1 AS '提分后' FROM result;
```

- 避免SQL返回结果中包含 '.', '*' 和括号等干扰开发语言程序.

where条件语句

作用：用于检索数据表中 符合条件的 记录

搜索条件可由一个或多个逻辑表达式组成，结果一般为真或假.

逻辑操作符

操作符名称	语法	描述
AND 或 &&	a AND b 或 a && b	逻辑与，同时为真结果才为真
OR 或	a OR b 或 a b	逻辑或，只要一个为真，则结果为真
NOT 或 ！	NOT a 或 ！a	逻辑非，若操作数为假，则结果为真！

测试

```
-- 满足条件的查询(where)
SELECT Studentno,StudentResult FROM result;

-- 查询考试成绩在95-100之间的
SELECT Studentno,StudentResult
FROM result
WHERE StudentResult>=95 AND StudentResult<=100;

-- AND也可以写成 &&
SELECT Studentno,StudentResult
FROM result
WHERE StudentResult>=95 && StudentResult<=100;

-- 模糊查询(对应的词:精确查询)
SELECT Studentno,StudentResult
FROM result
WHERE StudentResult BETWEEN 95 AND 100;

-- 除了1000号同学,要其他同学的成绩
SELECT studentno,studentresult
FROM result
WHERE studentno!=1000;
```



```
-- 使用NOT
SELECT studentno,studentresult
FROM result
WHERE NOT studentno=1000;
```

模糊查询：比较操作符

操作符名称	语法	描述
IS NULL	a IS NULL	若操作符为NULL，则结果为真
IS NOT NULL	a IS NOT NULL	若操作符不为NULL，则结果为真
BETWEEN	a BETWEEN b AND c	若 a 范围在 b 与 c 之间，则结果为真
LIKE	a LIKE b	SQL 模式匹配，若a匹配b，则结果为真
IN	a IN (a1 , a2 , a3 ,)	若 a 等于 a1,a2..... 中的某一个，则结果为真

注意：

- 数值数据类型的记录之间才能进行算术运算；
- 相同数据类型的数据之间才能进行比较；

测试：

```
-- 模糊查询 between and \ like \ in \ null

-- =====
-- LIKE
-- =====
-- 查询姓刘的同学的学号及姓名
-- like结合使用的通配符：%（代表0到任意个字符）_（一个字符）
SELECT studentno,studentname FROM student
WHERE studentname LIKE '刘%';

-- 查询姓刘的同学,后面只有一个字的
SELECT studentno,studentname FROM student
WHERE studentname LIKE '刘_';

-- 查询姓刘的同学,后面只有两个字的
SELECT studentno,studentname FROM student
WHERE studentname LIKE '刘__';

-- 查询姓名中含有 嘉 字的
SELECT studentno,studentname FROM student
WHERE studentname LIKE '%嘉%';

-- 查询姓名中含有特殊字符的需要使用转义符号 '\ '
-- 自定义转义符关键字：ESCAPE ':'

-- =====
-- IN
-- =====
-- 查询学号为1000,1001,1002的学生姓名
SELECT studentno,studentname FROM student
WHERE studentno IN (1000,1001,1002);

-- 查询地址在北京,南京,河南洛阳的学生
SELECT studentno,studentname,address FROM student
```

```

WHERE address IN ('北京','南京','河南洛阳');

-- =====
-- NULL 空
-- =====
-- 查询出生日期没有填写的同学
-- 不能直接写=NULL，这是代表错误的，用 is null
SELECT studentname FROM student
WHERE BornDate IS NULL;

-- 查询出生日期填写的同学
SELECT studentname FROM student
WHERE BornDate IS NOT NULL;

-- 查询没有写家庭住址的同学(空字符串不等于null)
SELECT studentname FROM student
WHERE Address='' OR Address IS NULL;

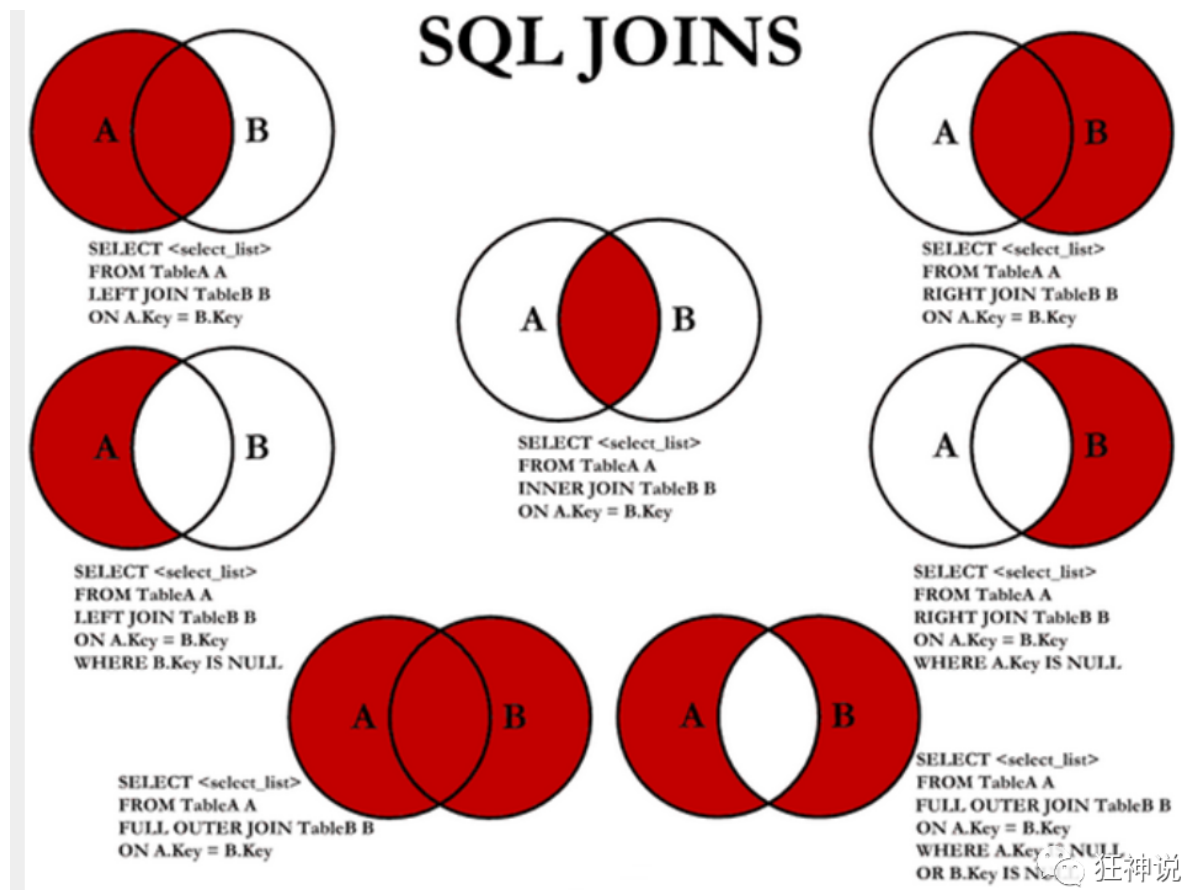
```

连接查询

JOIN 对比

操作符名称	描述
INNER JOIN	如果表中有至少一个匹配，则返回行
LEFT JOIN	即使右表中没有匹配，也从左表中返回所有的行
RIGHT JOIN	即使左表中没有匹配，也从右表中返回所有的行

七种Join:



测试

```
/*
连接查询
    如需要多张数据表的数据进行查询,则可通过连接运算符实现多个查询
内连接 inner join
    查询两个表中的结果集中的交集
外连接 outer join
    左外连接 left join
        (以左表作为基准,右边表来一一匹配,匹配不上的,返回左表的记录,右表以NULL填充)
    右外连接 right join
        (以右表作为基准,左边表来一一匹配,匹配不上的,返回右表的记录,左表以NULL填充)

等值连接和非等值连接

自连接
*/

-- 查询参加了考试的同学信息(学号,学生姓名,科目编号,分数)
SELECT * FROM student;
SELECT * FROM result;

/*思路:
(1):分析需求,确定查询的列来源于两个类,student result,连接查询
(2):确定使用哪种连接查询?(内连接)
*/
SELECT s.studentno,studentname,subjectno,StudentResult
FROM student s
INNER JOIN result r
ON r.studentno = s.studentno

-- 右连接(也可实现)
SELECT s.studentno,studentname,subjectno,StudentResult
FROM student s
RIGHT JOIN result r
ON r.studentno = s.studentno

-- 等值连接
SELECT s.studentno,studentname,subjectno,StudentResult
FROM student s , result r
WHERE r.studentno = s.studentno

-- 左连接 (查询了所有同学,不考试的也会查出来)
SELECT s.studentno,studentname,subjectno,StudentResult
FROM student s
LEFT JOIN result r
ON r.studentno = s.studentno

-- 查一下缺考的同学(左连接应用场景)
SELECT s.studentno,studentname,subjectno,StudentResult
FROM student s
LEFT JOIN result r
ON r.studentno = s.studentno
WHERE StudentResult IS NULL

-- 思考题:查询参加了考试的同学信息(学号,学生姓名,科目名,分数)
SELECT s.studentno,studentname,subjectname,StudentResult
```

```

FROM student s
INNER JOIN result r
ON r.studentno = s.studentno
INNER JOIN `subject` sub
ON sub.subjectno = r.subjectno

```

自连接

```

/*
自连接
    数据表与自身进行连接

需求: 从一个包含栏目ID, 栏目名称和父栏目ID的表中
    查询父栏目名称和其他子栏目名称
*/

-- 创建一个表
CREATE TABLE `category` (
`categoryid` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主题id',
`pid` INT(10) NOT NULL COMMENT '父id',
`categoryName` VARCHAR(50) NOT NULL COMMENT '主题名字',
PRIMARY KEY (`categoryid`)
) ENGINE=INNODB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8

-- 插入数据
INSERT INTO `category` (`categoryid`, `pid`, `categoryName`)
VALUES('2','1','信息技术'),
('3','1','软件开发'),
('4','3','数据库'),
('5','1','美术设计'),
('6','3','web开发'),
('7','5','ps技术'),
('8','2','办公信息');

-- 编写SQL语句, 将栏目的父子关系呈现出来 (父栏目名称, 子栏目名称)
-- 核心思想: 把一张表看成两张一模一样的表, 然后将这两张表连接查询(自连接)
SELECT a.categoryName AS '父栏目', b.categoryName AS '子栏目'
FROM category AS a, category AS b
WHERE a.`categoryid`=b.`pid`

-- 思考题: 查询参加了考试的同学信息(学号, 学生姓名, 科目名, 分数)
SELECT s.studentno, studentname, subjectname, StudentResult
FROM student s
INNER JOIN result r
ON r.studentno = s.studentno
INNER JOIN `subject` sub
ON sub.subjectno = r.subjectno

-- 查询学员及所属的年级(学号, 学生姓名, 年级名)
SELECT studentno AS 学号, studentname AS 学生姓名, gradename AS 年级名称
FROM student s
INNER JOIN grade g
ON s.`GradeId` = g.`GradeID`

-- 查询科目及所属的年级(科目名称, 年级名称)
SELECT subjectname AS 科目名称, gradename AS 年级名称
FROM SUBJECT sub

```

```

INNER JOIN grade g
ON sub.gradeid = g.gradeid

-- 查询 数据库结构-1 的所有考试结果(学号 学生姓名 科目名称 成绩)
SELECT s.studentno,studentname,subjectname,StudentResult
FROM student s
INNER JOIN result r
ON r.studentno = s.studentno
INNER JOIN `subject` sub
ON r.subjectno = sub.subjectno
WHERE subjectname='数据库结构-1'

```

排序和分页

测试

```

/*===== 排序 =====
语法 : ORDER BY
        ORDER BY 语句用于根据指定的列对结果集进行排序。
        ORDER BY 语句默认按照ASC升序对记录进行排序。
        如果您希望按照降序对记录进行排序, 可以使用 DESC 关键字。

*/

-- 查询 数据库结构-1 的所有考试结果(学号 学生姓名 科目名称 成绩)
-- 按成绩降序排序
SELECT s.studentno,studentname,subjectname,StudentResult
FROM student s
INNER JOIN result r
ON r.studentno = s.studentno
INNER JOIN `subject` sub
ON r.subjectno = sub.subjectno
WHERE subjectname='数据库结构-1'
ORDER BY StudentResult DESC

/*===== 分页 =====
语法 : SELECT * FROM table LIMIT [offset,] rows | rows OFFSET offset
好处 : (用户体验,网络传输,查询压力)

推导:
    第一页 : limit 0,5
    第二页 : limit 5,5
    第三页 : limit 10,5
    .....
    第N页 : limit (pageNo-1)*pageSize,pageSize
    [pageNo:页码,pageSize:单页面显示条数]

*/

-- 每页显示5条数据
SELECT s.studentno,studentname,subjectname,StudentResult
FROM student s
INNER JOIN result r
ON r.studentno = s.studentno
INNER JOIN `subject` sub
ON r.subjectno = sub.subjectno
WHERE subjectname='数据库结构-1'

```

```
ORDER BY StudentResult DESC , studentno
LIMIT 0,5

-- 查询 JAVA第一学年 课程成绩前10名并且分数大于80的学生信息(学号,姓名,课程名,分数)
SELECT s.studentno,studentname,subjectname,StudentResult
FROM student s
INNER JOIN result r
ON r.studentno = s.studentno
INNER JOIN `subject` sub
ON r.subjectno = sub.subjectno
WHERE subjectname='JAVA第一学年'
ORDER BY StudentResult DESC
LIMIT 0,10
```

子查询

```
/*===== 子查询 =====
什么是子查询?
    在查询语句中的WHERE条件子句中,又嵌套了另一个查询语句
    嵌套查询可由多个子查询组成,求解的方式是由里及外;
    子查询返回的结果一般都是集合,故而建议使用IN关键字;
*/

-- 查询 数据库结构-1 的所有考试结果(学号,科目编号,成绩),并且成绩降序排列
-- 方法一:使用连接查询
SELECT studentno,r.subjectno,StudentResult
FROM result r
INNER JOIN `subject` sub
ON r.`SubjectNo`=sub.`SubjectNo`
WHERE subjectname = '数据库结构-1'
ORDER BY studentresult DESC;

-- 方法二:使用子查询(执行顺序:由里及外)
SELECT studentno,subjectno,StudentResult
FROM result
WHERE subjectno=(
    SELECT subjectno FROM `subject`
    WHERE subjectname = '数据库结构-1'
)
ORDER BY studentresult DESC;

-- 查询课程为 高等数学-2 且分数不小于80分的学生的学号和姓名
-- 方法一:使用连接查询
SELECT s.studentno,studentname
FROM student s
INNER JOIN result r
ON s.`StudentNo` = r.`StudentNo`
INNER JOIN `subject` sub
ON sub.`SubjectNo` = r.`SubjectNo`
WHERE subjectname = '高等数学-2' AND StudentResult>=80

-- 方法二:使用连接查询+子查询
-- 分数不小于80分的学生的学号和姓名
SELECT r.studentno,studentname FROM student s
INNER JOIN result r ON s.`StudentNo`=r.`StudentNo`
WHERE StudentResult>=80
```

```
-- 在上面SQL基础上,添加需求:课程为 高等数学-2
SELECT r.studentno,studentname FROM student s
INNER JOIN result r ON s.`StudentNo`=r.`StudentNo`
WHERE StudentResult>=80 AND subjectno=(
    SELECT subjectno FROM `subject`
    WHERE subjectname = '高等数学-2'
)

-- 方法三:使用子查询
-- 分步写简单sql语句,然后将其嵌套起来
SELECT studentno,studentname FROM student WHERE studentno IN(
    SELECT studentno FROM result WHERE StudentResult>=80 AND subjectno=(
        SELECT subjectno FROM `subject` WHERE subjectname = '高等数学-2'
    )
)

/*
练习题目:
    查 C语言-1 的前5名学生的成绩信息(学号,姓名,分数)
    使用子查询,查询郭靖同学所在的年级名称
*/
```

MySQL函数

常用函数

数据函数

```
SELECT ABS(-8); /*绝对值*/
SELECT CEILING(9.4); /*向上取整*/
SELECT FLOOR(9.4); /*向下取整*/
SELECT RAND(); /*随机数,返回一个0-1之间的随机数*/
SELECT SIGN(0); /*符号函数: 负数返回-1,正数返回1,0返回0*/
```

字符串函数

```
SELECT CHAR_LENGTH('狂神说坚持就能成功'); /*返回字符串包含的字符数*/
SELECT CONCAT('我','爱','程序'); /*合并字符串,参数可以有多个*/
SELECT INSERT('我爱编程helloworld',1,2,'超级热爱'); /*替换字符串,从某个位置开始替换某个长度*/
SELECT LOWER('KuangShen'); /*小写*/
SELECT UPPER('KuangShen'); /*大写*/
SELECT LEFT('hello,world',5); /*从左边截取*/
SELECT RIGHT('hello,world',5); /*从右边截取*/
SELECT REPLACE('狂神说坚持就能成功','坚持','努力'); /*替换字符串*/
SELECT SUBSTR('狂神说坚持就能成功',4,6); /*截取字符串,开始和长度*/
SELECT REVERSE('狂神说坚持就能成功'); /*反转

-- 查询姓周的同学,改成邹
SELECT REPLACE(studentname,'周','邹') AS 新名字
FROM student WHERE studentname LIKE '周%';
```

日期和时间函数

```

SELECT CURRENT_DATE();    /*获取当前日期*/
SELECT CURDATE();         /*获取当前日期*/
SELECT NOW();             /*获取当前日期和时间*/
SELECT LOCALTIME();       /*获取当前日期和时间*/
SELECT SYSDATE();         /*获取当前日期和时间*/

-- 获取年月日,时分秒
SELECT YEAR(NOW());
SELECT MONTH(NOW());
SELECT DAY(NOW());
SELECT HOUR(NOW());
SELECT MINUTE(NOW());
SELECT SECOND(NOW());

```

系统信息函数

```

SELECT VERSION();    /*版本*/
SELECT USER();       /*用户*/

```

聚合函数

函数名称	描述
COUNT()	返回满足Select条件的记录总和数，如 select count(*) 【不建议使用 *，效率低】
SUM()	返回数字字段或表达式列作统计，返回一列的总和。
AVG()	通常为数值字段或表达列作统计，返回一列的平均值
MAX()	可以为数值字段，字符字段或表达式列作统计，返回最大的值。
MIN()	可以为数值字段，字符字段或表达式列作统计，返回最小的值。

```

-- 聚合函数
/*COUNT:非空的*/
SELECT COUNT(studentname) FROM student;
SELECT COUNT(*) FROM student;
SELECT COUNT(1) FROM student;    /*推荐*/

-- 从含义上讲，count(1) 与 count(*) 都表示对全部数据行的查询。
-- count(字段) 会统计该字段在表中出现的次数，忽略字段为null 的情况。即不统计字段为null 的记录。
-- count(*) 包括了所有的列，相当于行数，在统计结果的时候，包含字段为null 的记录；
-- count(1) 用1代表代码行，在统计结果的时候，包含字段为null 的记录 。
/*
    很多人认为count(1)执行的效率会比count(*)高，原因是count(*)会存在全表扫描，而count(1)可以针对一个字段进行查询。其实不然，count(1)和count(*)都会对全表进行扫描，统计所有记录的条数，包括那些为null的记录，因此，它们的效率可以说是相差无几。而count(字段)则与前两者不同，它会统计该字段不为null的记录条数。

```

下面它们之间的一些对比：

1) 在表没有主键时，count(1)比count(*)快

2) 有主键时, 主键作为计算条件, **count(主键)**效率最高;

3) 若表格只有一个字段, 则**count(*)**效率较高。

*/

```
SELECT SUM(StudentResult) AS 总和 FROM result;
```

```
SELECT AVG(StudentResult) AS 平均分 FROM result;
```

```
SELECT MAX(StudentResult) AS 最高分 FROM result;
```

```
SELECT MIN(StudentResult) AS 最低分 FROM result;
```

题目:

-- 查询不同课程的平均分,最高分,最低分

-- 前提:根据不同的课程进行分组

```
SELECT subjectname,AVG(studentresult) AS 平均分,MAX(StudentResult) AS 最高分,MIN(StudentResult) AS 最低分
FROM result AS r
INNER JOIN `subject` AS s
ON r.subjectno = s.subjectno
GROUP BY r.subjectno
HAVING 平均分>80;
```

/*

where写在group by前面.

要是放在分组后面的筛选

要使用HAVING..

因为having是从前面筛选的字段再筛选, 而where是从数据表中的>字段直接进行的筛选的

*/

MD5 加密

一、MD5简介

MD5即Message-Digest Algorithm 5 (信息-摘要算法5), 用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一(又译摘要算法、哈希算法), 主流编程语言普遍已有MD5实现。将数据(如汉字)运算为另一固定长度值, 是杂凑算法的基础原理, MD5的前身有MD2、MD3和MD4。

二、实现数据加密

新建一个表 testmd5

```
CREATE TABLE `testmd5` (
  `id` INT(4) NOT NULL,
  `name` VARCHAR(20) NOT NULL,
  `pwd` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8
```

插入一些数据

```
INSERT INTO testmd5 VALUES(1,'kuangshen','123456'),(2,'qinjiang','456789')
```

如果我们要对pwd这一列数据进行加密, 语法是:

```
update testmd5 set pwd = md5(pwd);
```

如果单独对某个用户(如kuangshen)的密码加密:

```
INSERT INTO testmd5 VALUES(3,'kuangshen2','123456')
update testmd5 set pwd = md5(pwd) where name = 'kuangshen2';
```

插入新的数据自动加密

```
INSERT INTO testmd5 VALUES(4,'kuangshen3',md5('123456'));
```

查询登录用户信息 (md5对比使用, 查看用户输入加密后的密码进行比对)

```
SELECT * FROM testmd5 WHERE `name`='kuangshen' AND pwd=MD5('123456');
```

小结

```
-- ===== 内置函数 =====
-- 数值函数
abs(x)          -- 绝对值 abs(-10.9) = 10
format(x, d)    -- 格式化千分位数值 format(1234567.456, 2) = 1,234,567.46
ceil(x)         -- 向上取整 ceil(10.1) = 11
floor(x)        -- 向下取整 floor (10.1) = 10
round(x)        -- 四舍五入去整
mod(m, n)       -- m%n m mod n 求余 10%3=1
pi()            -- 获得圆周率
pow(m, n)       -- m^n
sqrt(x)         -- 算术平方根
rand()          -- 随机数
truncate(x, d)  -- 截取d位小数

-- 时间日期函数
now(), current_timestamp(); -- 当前日期时间
current_date();           -- 当前日期
current_time();           -- 当前时间
date('yyyy-mm-dd hh:ii:ss'); -- 获取日期部分
time('yyyy-mm-dd hh:ii:ss'); -- 获取时间部分
date_format('yyyy-mm-dd hh:ii:ss', '%d %y %a %d %m %b %j'); -- 格式化时间
unix_timestamp();         -- 获得unix时间戳
from_unixtime();          -- 从时间戳获得时间

-- 字符串函数
length(string)           -- string长度, 字节
char_length(string)      -- string的字符个数
substring(str, position [,length]) -- 从str的position开始,取length个字符
replace(str,search_str,replace_str) -- 在str中用replace_str替换search_str
instr(string,substring)  -- 返回substring首次在string中出现的位置
concat(string [...])     -- 连接字符串
charset(str)             -- 返回字符串字符集
lcase(string)            -- 转换成小写
left(string, length)     -- 从string2中的左边起取length个字符
load_file(file_name)     -- 从文件读取内容
locate(substring, string [,start_position]) -- 同instr,但可指定开始位置
lpad(string, length, pad) -- 重复用pad加在string开头,直到字符串长度为length
ltrim(string)            -- 去除前端空格
```

```
repeat(string, count)    -- 重复count次
rpad(string, length, pad)  -- 在str后用pad补充,直到长度为length
rtrim(string)            -- 去除后端空格
strcmp(string1 ,string2)  -- 逐字符比较两字符串大小

-- 聚合函数
count()
sum();
max();
min();
avg();
group_concat()

-- 其他常用函数
md5();
default();
```

事务

什么是事务

- 事务就是将一组SQL语句放在同一批次内去执行
- 如果一个SQL语句出错,则该批次内的所有SQL都将被取消执行
- MySQL事务处理只支持InnoDB和BDB数据表类型

事务的ACID原则 百度 ACID

原子性(Atomic)

- 整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（ROLLBACK）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性(Consist)

- 一个事务可以封装状态改变（除非它是一个只读的）。事务必须始终保持系统处于一致的状态，不管在任何给定的时间并发事务有多少。也就是说：如果事务是并发多个，系统也必须如同串行事务一样操作。其主要特征是保护性和不变性(Preserving an Invariant)，以转账案例为例，假设有五个账户，每个账户余额是100元，那么五个账户总额是500元，如果在这个5个账户之间同时发生多个转账，无论并发多少个，比如在A与B账户之间转账5元，在C与D账户之间转账10元，在B与E之间转账15元，五个账户总额也应该还是500元，这就是保护性和不变性。

隔离性(Isolated)

- 隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请求，使得在同一时间仅有一个请求用于同一数据。

持久性(Durable)

- 在事务完成以后，该事务对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

基本语法

```
-- 使用set语句来改变自动提交模式
```

```

SET autocommit = 0;    /*关闭*/
SET autocommit = 1;    /*开启*/

-- 注意：
--- 1. MySQL中默认是自动提交
--- 2. 使用事务时应先关闭自动提交

-- 开始一个事务,标记事务的起始点
START TRANSACTION

-- 提交一个事务给数据库
COMMIT

-- 将事务回滚,数据回到本次事务的初始状态
ROLLBACK

-- 还原MySQL数据库的自动提交
SET autocommit =1;

-- 保存点
SAVEPOINT 保存点名称 -- 设置一个事务保存点
ROLLBACK TO SAVEPOINT 保存点名称 -- 回滚到保存点
RELEASE SAVEPOINT 保存点名称 -- 删除保存点

```

测试

```

/*
课堂测试题目

A在线买一款价格为500元商品,网上银行转账.
A的银行卡余额为2000,然后给商家B支付500.
商家B一开始的银行卡余额为10000

创建数据库shop和创建表account并插入2条数据
*/

CREATE DATABASE `shop` CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `shop`;

CREATE TABLE `account` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(32) NOT NULL,
  `cash` DECIMAL(9,2) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=INNODB DEFAULT CHARSET=utf8

INSERT INTO account (`name`,`cash`)
VALUES('A',2000.00),('B',10000.00)

-- 转账实现
SET autocommit = 0; -- 关闭自动提交
START TRANSACTION; -- 开始一个事务,标记事务的起始点
UPDATE account SET cash=cash-500 WHERE `name`='A';
UPDATE account SET cash=cash+500 WHERE `name`='B';
COMMIT; -- 提交事务
# rollback;
SET autocommit = 1; -- 恢复自动提交

```

索引

索引的作用

- 提高查询速度
- 确保数据的唯一性
- 可以加速表和表之间的连接,实现表与表之间的参照完整性
- 使用分组和排序子句进行数据检索时,可以显著减少分组和排序的时间
- 全文检索字段进行搜索优化.

分类

- 主键索引 (Primary Key)
- 唯一索引 (Unique)
- 常规索引 (Index)
- 全文索引 (FullText)

主键索引

主键: 某一个属性组能唯一标识一条记录

特点:

- 最常见的索引类型
- 确保数据记录的唯一性
- 确定特定数据记录在数据库中的位置

唯一索引

作用: 避免同一个表中某数据列中的值重复

与主键索引的区别

- 主键索引只能有一个
- 唯一索引可能有多个

```
CREATE TABLE `Grade` (  
  `GradeID` INT(11) AUTO_INCREMENT PRIMARYKEY,  
  `GradeName` VARCHAR(32) NOT NULL UNIQUE  
  -- 或 UNIQUE KEY `GradeID` (`GradeID`)  
)
```

常规索引

作用: 快速定位特定数据

注意:

- index 和 key 关键字都可以设置常规索引
- 应加在查询找条件的字段
- 不宜添加太多常规索引,影响数据的插入,删除和修改操作

```
CREATE TABLE `result` (
    -- 省略一些代码
    INDEX/KEY `ind` (`studentNo`, `subjectNo`) -- 创建表时添加
)
-- 创建后添加
ALTER TABLE `result` ADD INDEX `ind` (`studentNo`, `subjectNo`);
```

全文索引

百度搜索：全文索引

作用：快速定位特定数据

注意：

- 只能用于MyISAM类型的数据表
- 只能用于CHAR, VARCHAR, TEXT数据列类型
- 适合大型数据集

```
/*
#方法一：创建表时
    CREATE TABLE 表名 (
        字段名1 数据类型 [完整性约束条件...],
        字段名2 数据类型 [完整性约束条件...],
        [UNIQUE | FULLTEXT | SPATIAL ] INDEX | KEY
        [索引名] (字段名[(长度)] [ASC | DESC])
    );

#方法二：CREATE在已存在的表上创建索引
    CREATE [UNIQUE | FULLTEXT | SPATIAL ] INDEX 索引名
        ON 表名 (字段名[(长度)] [ASC | DESC]) ;

#方法三：ALTER TABLE在已存在的表上创建索引
    ALTER TABLE 表名 ADD [UNIQUE | FULLTEXT | SPATIAL ] INDEX
        索引名 (字段名[(长度)] [ASC | DESC]) ;

#删除索引：DROP INDEX 索引名 ON 表名字；
#删除主键索引：ALTER TABLE 表名 DROP PRIMARY KEY；

#显示索引信息：SHOW INDEX FROM student;
*/

/*增加全文索引*/
ALTER TABLE `school`.`student` ADD FULLTEXT INDEX `studentname` (`StudentName`);

/*EXPLAIN ：分析SQL语句执行性能*/
EXPLAIN SELECT * FROM student WHERE studentno='1000';

/*使用全文索引*/
-- 全文搜索通过 MATCH() 函数完成。
-- 搜索字符串作为 against() 的参数被给定。搜索以忽略字母大小写的方式执行。对于表中的每个记录行，MATCH() 返回一个相关性值。即，在搜索字符串与记录行在 MATCH() 列表中指定的列的文本之间的相似性尺度。
```

```
EXPLAIN SELECT *FROM student WHERE MATCH(studentname) AGAINST('love');
```

```
/*
```

开始之前，先说一下全文索引的版本、存储引擎、数据类型的支持情况

MySQL 5.6 以前的版本，只有 MyISAM 存储引擎支持全文索引；

MySQL 5.6 及以后的版本，MyISAM 和 InnoDB 存储引擎均支持全文索引；

只有字段的数据类型为 char、varchar、text 及其系列才可以建全文索引。

测试或使用全文索引时，要先看一下自己的 MySQL 版本、存储引擎和数据类型是否支持全文索引。

```
*/
```

拓展：测试索引

建表app_user:

```
CREATE TABLE `app_user` (  
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,  
  `name` varchar(50) DEFAULT '' COMMENT '用户昵称',  
  `email` varchar(50) NOT NULL COMMENT '用户邮箱',  
  `phone` varchar(20) DEFAULT '' COMMENT '手机号',  
  `gender` tinyint(4) unsigned DEFAULT '0' COMMENT '性别（0:男；1: 女）',  
  `password` varchar(100) NOT NULL COMMENT '密码',  
  `age` tinyint(4) DEFAULT '0' COMMENT '年龄',  
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP,  
  `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='app用户表'
```

批量插入数据：100w

```
DROP FUNCTION IF EXISTS mock_data;  
DELIMITER $$  
CREATE FUNCTION mock_data()  
RETURNS INT  
BEGIN  
  DECLARE num INT DEFAULT 1000000;  
  DECLARE i INT DEFAULT 0;  
  WHILE i < num DO  
    INSERT INTO app_user(`name`, `email`, `phone`, `gender`, `password`, `age`)  
    VALUES(CONCAT('用户', i), '24736743@qq.com', CONCAT('18', FLOOR(RAND()*  
(999999999-100000000)+100000000)), FLOOR(RAND()*2), UUID(), FLOOR(RAND()*100));  
    SET i = i + 1;  
  END WHILE;  
  RETURN i;  
END;  
SELECT mock_data();
```

索引效率测试

无索引

```
SELECT * FROM app_user WHERE name = '用户9999'; -- 查看耗时  
SELECT * FROM app_user WHERE name = '用户9999';  
SELECT * FROM app_user WHERE name = '用户9999';
```

```
mysql> EXPLAIN SELECT * FROM app_user WHERE name = '用户9999'\G
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: app_user
  partitions: NULL
      type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 992759
   filtered: 10.00
      Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

创建索引

```
CREATE INDEX idx_app_user_name ON app_user(name);
```

测试普通索引

```
mysql> EXPLAIN SELECT * FROM app_user WHERE name = '用户9999'\G
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: app_user
  partitions: NULL
      type: ref
possible_keys: idx_app_user_name
         key: idx_app_user_name
      key_len: 203
         ref: const
        rows: 1
   filtered: 100.00
      Extra: NULL
1 row in set, 1 warning (0.00 sec)

mysql> SELECT * FROM app_user WHERE name = '用户9999';
1 row in set (0.00 sec)

mysql> SELECT * FROM app_user WHERE name = '用户9999';
1 row in set (0.00 sec)

mysql> SELECT * FROM app_user WHERE name = '用户9999';
1 row in set (0.00 sec)
```

索引准则

- 索引不是越多越好
- 不要对经常变动的数据加索引
- 小数据量的表建议不要加索引
- 索引一般应加在查找条件的字段

索引的数据结构

-- 我们可以在创建上述索引的时候，为其指定索引类型，分两类

hash类型的索引：查询单条快，范围查询慢

btree类型的索引：**b+**树，层数越多，数据量指数级增长（我们就用它，因为**innodb**默认支持它）

-- 不同的存储引擎支持的索引类型也不一样

InnoDB 支持事务，支持行级别锁定，支持 **B-tree**、**Full-text** 等索引，不支持 **Hash** 索引；

MyISAM 不支持事务，支持表级别锁定，支持 **B-tree**、**Full-text** 等索引，不支持 **Hash** 索引；

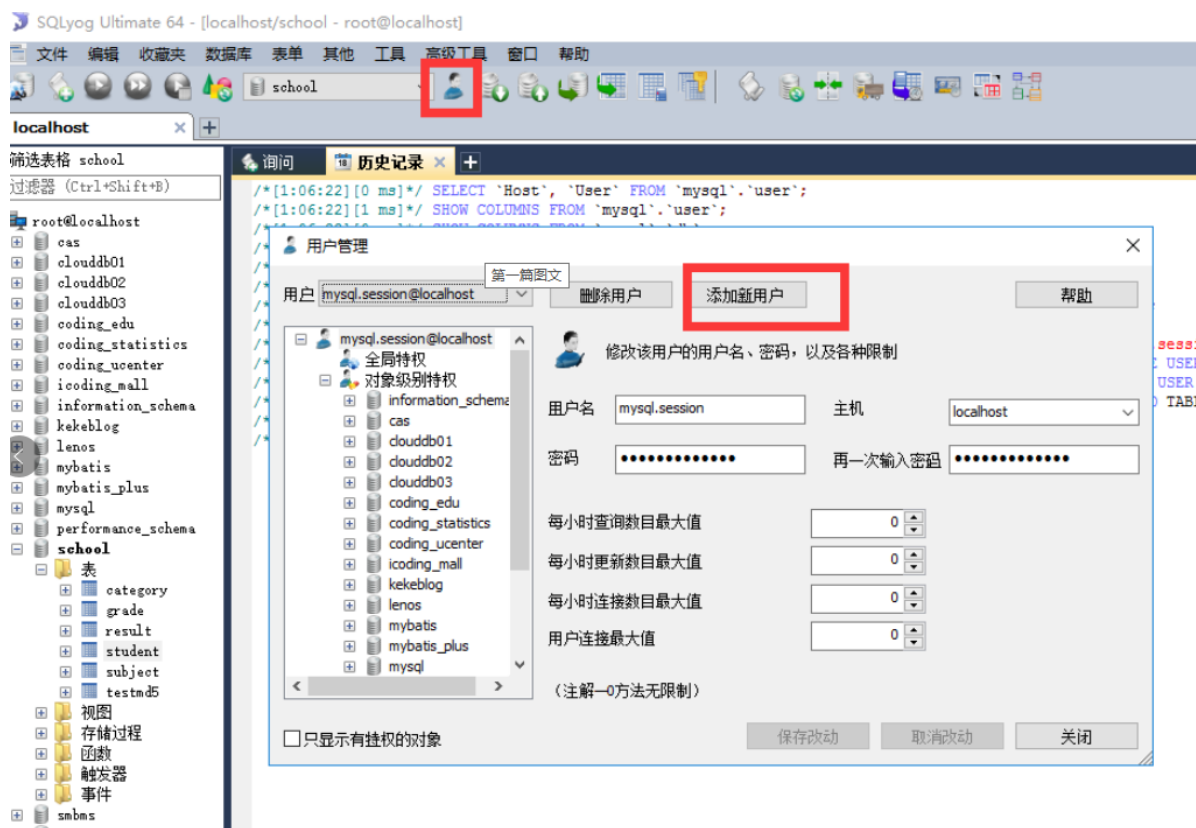
Memory 不支持事务，支持表级别锁定，支持 **B-tree**、**Hash** 等索引，不支持 **Full-text** 索引；

NDB 支持事务，支持行级别锁定，支持 **Hash** 索引，不支持 **B-tree**、**Full-text** 等索引；

Archive 不支持事务，支持表级别锁定，不支持 **B-tree**、**Hash**、**Full-text** 等索引；

权限及如何设计数据库

使用SQLyog 创建用户，并授予权限演示



基本命令

/* 用户和权限管理 */ -----

用户信息表: **mysql.user**

-- 刷新权限

FLUSH PRIVILEGES

-- 增加用户 **CREATE USER kuangshen IDENTIFIED BY '123456'**

CREATE USER 用户名 **IDENTIFIED BY** [PASSWORD] 密码(字符串)

- 必须拥有**mysql**数据库的全局**CREATE USER**权限，或拥有**INSERT**权限。
- 只能创建用户，不能赋予权限。
- 用户名，注意引号：如 **'user_name'@'192.168.1.1'**
- 密码也需引号，纯数字密码也要加引号

- 要在纯文本中指定密码，需忽略PASSWORD关键词。要把密码指定为由PASSWORD()函数返回的混编值，需包含关键字PASSWORD

```
-- 重命名用户 RENAME USER kuangshen TO kuangshen2
RENAME USER old_user TO new_user

-- 设置密码
SET PASSWORD = PASSWORD('密码')      -- 为当前用户设置密码
SET PASSWORD FOR 用户名 = PASSWORD('密码')    -- 为指定用户设置密码

-- 删除用户 DROP USER kuangshen2
DROP USER 用户名

-- 分配权限/添加用户
GRANT 权限列表 ON 表名 TO 用户名 [IDENTIFIED BY [PASSWORD] 'password']
- all privileges 表示所有权限
- *.* 表示所有库的所有表
- 库名.表名 表示某库下面的某表

-- 查看权限 SHOW GRANTS FOR root@localhost;
SHOW GRANTS FOR 用户名
-- 查看当前用户权限
SHOW GRANTS; 或 SHOW GRANTS FOR CURRENT_USER; 或 SHOW GRANTS FOR
CURRENT_USER();

-- 撤销权限
REVOKE 权限列表 ON 表名 FROM 用户名
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 用户名    -- 撤销所有权限
```

权限解释

```
-- 权限列表
ALL [PRIVILEGES]      -- 设置除GRANT OPTION之外的所有简单权限
ALTER      -- 允许使用ALTER TABLE
ALTER ROUTINE      -- 更改或取消已存储的子程序
CREATE      -- 允许使用CREATE TABLE
CREATE ROUTINE      -- 创建已存储的子程序
CREATE TEMPORARY TABLES      -- 允许使用CREATE TEMPORARY TABLE
CREATE USER      -- 允许使用CREATE USER, DROP USER, RENAME USER和REVOKE ALL
PRIVILEGES。
CREATE VIEW      -- 允许使用CREATE VIEW
DELETE      -- 允许使用DELETE
DROP      -- 允许使用DROP TABLE
EXECUTE      -- 允许用户运行已存储的子程序
FILE      -- 允许使用SELECT...INTO OUTFILE和LOAD DATA INFILE
INDEX      -- 允许使用CREATE INDEX和DROP INDEX
INSERT      -- 允许使用INSERT
LOCK TABLES      -- 允许对您拥有SELECT权限的表使用LOCK TABLES
PROCESS      -- 允许使用SHOW FULL PROCESSLIST
REFERENCES      -- 未被实施
RELOAD      -- 允许使用FLUSH
REPLICATION CLIENT      -- 允许用户询问从属服务器或主服务器的地址
REPLICATION SLAVE      -- 用于复制型从属服务器（从主服务器中读取二进制日志事件）
SELECT      -- 允许使用SELECT
SHOW DATABASES      -- 显示所有数据库
SHOW VIEW      -- 允许使用SHOW CREATE VIEW
SHUTDOWN      -- 允许使用mysqladmin shutdown
```

```
SUPER      -- 允许使用CHANGE MASTER, KILL, PURGE MASTER LOGS和SET GLOBAL语句,
mysqladmin debug命令; 允许您连接(一次), 即使已达到max_connections。
UPDATE      -- 允许使用UPDATE
USAGE      -- “无权限”的同义词
GRANT OPTION -- 允许授予权限
```

```
/* 表维护 */
```

```
-- 分析和存储表的关键字分布
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE 表名 ...
-- 检查一个或多个表是否有错误
CHECK TABLE tbl_name [, tbl_name] ... [option] ...
option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
-- 整理数据文件的碎片
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

MySQL备份

数据库备份必要性

- 保证重要数据不丢失
- 数据转移

MySQL数据库备份方法

- mysqldump备份工具
- 数据库管理工具,如SQLyog
- 直接拷贝数据库文件和相关配置文件

mysqldump客户端

作用:

- 转储数据库
- 搜集数据库进行备份
- 将数据转移到另一个SQL服务器,不一定是MySQL服务器

```
mysqldump -h 主机名 -u 用户名 -p [options] 数据库名
[ table1 table2 table3 ] > path/filename.sql
```

示例

备份myschool数据库如:

```
> mysqldump -u root -p myschool > d:/myschool.sql
```

EnterPassword: *****

预存文件目录, 须有该
目录读写权限

狂神说

```
-- 导出
1. 导出一张表 -- mysqldump -uroot -p123456 school student >D:/a.sql
mysqldump -u用户名 -p密码 库名 表名 > 文件名(D:/a.sql)
2. 导出多张表 -- mysqldump -uroot -p123456 school student result >D:/a.sql
mysqldump -u用户名 -p密码 库名 表1 表2 表3 > 文件名(D:/a.sql)
3. 导出所有表 -- mysqldump -uroot -p123456 school >D:/a.sql
mysqldump -u用户名 -p密码 库名 > 文件名(D:/a.sql)
```

```
4. 导出一个库 -- mysqldump -uroot -p123456 -B school >D:/a.sql  
mysqldump -u用户名 -p密码 -B 库名 > 文件名(D:/a.sql)
```

可以-w携带备份条件

-- 导入

1. 在登录mysql的情况下: -- source D:/a.sql
source 备份文件
2. 在不登录的情况下
mysql -u用户名 -p密码 库名 < 备份文件

规范化数据库设计

为什么需要数据库设计

当数据库比较复杂时我们需要设计数据库

糟糕的数据库设计：

- 数据冗余,存储空间浪费
- 数据更新和插入的异常
- 程序性能差

良好的数据库设计：

- 节省数据的存储空间
- 能够保证数据的完整性
- 方便进行数据库应用系统的开发

软件项目开发周期中数据库设计：

- 需求分析阶段: 分析客户的业务和数据处理需求
- 概要设计阶段: 设计数据库的E-R模型图, 确认需求信息的正确和完整.

设计数据库步骤

- 收集信息
 - 与该系统有关人员进行交流, 座谈, 充分了解用户需求, 理解数据库需要完成的任务.
- 标识实体[Entity]
- - 标识数据库要管理的关键对象或实体, 实体一般是名词
- 标识每个实体需要存储的详细信息[Attribute]
- 标识实体之间的关系[Relationship]

三大范式

问题：为什么需要数据规范化？

不合规的表设计会导致的问题：

- 信息重复
- 更新异常
- 插入异常
 - 无法正确表示信息
- 删除异常

- ◦ 丢失有效信息

三大范式

第一范式 (1st NF)

第一范式的目标是确保每列的原子性,如果每列都是不可再分的最小数据单元,则满足第一范式

第二范式(2nd NF)

第二范式 (2NF) 是在第一范式 (1NF) 的基础上建立起来的, 即满足第二范式 (2NF) 必须先满足第一范式 (1NF) 。

第二范式要求每个表只描述一件事情

第三范式(3rd NF)

如果一个关系满足第二范式,并且除了主键以外的其他列都不传递依赖于主键列,则满足第三范式.

第三范式需要确保数据表中的每一列数据都和主键直接相关, 而不能间接相关。

规范化和性能的关系

为满足某种商业目标, 数据库性能比规范化数据库更重要

在数据规范化的同时, 要综合考虑数据库的性能

通过在给定的表中添加额外的字段,以大量减少需要从中搜索信息所需的时间

通过在给定的表中插入计算列,以方便查询

