MySQL基础

一、 了解数据库

1.1. 相关概念

- **数据库（database，简称DB）**：保存有组织的数据的容器（通常是一个文件或一组文件）。

  > **误用导致混淆**　人们通常用数据库这个术语来代表他们使用的数据库软件。这是不正确的，确切地说,数据库软件应当称为DBMS（数据库管理系统）。数据库是通过DBMS创建和操纵的容器。数据库可以是保存在硬设备上的文件，但也可以不是。

- **数据库管理系统（Database Management System，简称DBMS）**：是为管理数据库而设计的电脑软件系统，一般具有存储、截取、安全保障、备份等基础功能。

1.2. 关系型数据库

关系型数据库（Relational database）：是创建在关系模型上的数据库，以行列的形式来存储数据。方便用户的理解，在关系型数据库中一系列的行和列称为表，一组表组成数据库。（单库的表容量是固定：可以进行分库分表的操作），可以将关系型数据库理解为二维数据表格模型，而一个关系型数据库是由二维表及其之间的关系组成的数据的组织。

1.2.1 常用的关系型数据库管理系统

- MySQL

  > 最受欢迎的开源的SQL数据库管理系统
  >
  > 2003年Mysql5.0：支持SQL特性，事务，视图、存储过程、触发器等功能。
  >
  > 2010年MySQL5.5，InnoDB存储引擎变为MySQL的默认存储引擎。

  > 优势：
  >
  > - MySQL是开放源代码的，可以免费使用（甚至可以修改源码）
  > - MySQL服务器是一个快速的、易于使用的数据库服务器
  > - MySQL可以在不同的操作系统中使用

- MariaDB

> 是由MySQL的创始人主导开发的。担心Oracle将MySQL闭源。目前大型的互联网公司纷纷抛弃MySQL
>
> 转入到MariaDB。

- PostgreSQL

> 完整的支持了SQL标准，开源，可以在不同的操作系统中运行。

- Oracle数据库

> 最先将关系型数据库转到桌面计算机上。客户/服务器结构的概念。

> Oracle数据库的优势：
>
> - 兼容性（采用SQL标准）
> - 可移植性（window,linux,unix,dos）
> - 可连接性（支持各种网络传输协议：TCP/IP,、DECnet,LU6.2)
> - 高生产率（提供了多种开发工具，可以方便用户快速的开发）
> - 开放性（oracle良好的兼容性、可以移植性、可连接性和高生产率使用oracle具有良好的开放性)

- SQL Server

> 微软旗下，和.net，在国内广泛用于电力，保险等行业。2017版之前的SQL Server只支持windows操作
>
> 系统。2017年后SQL Server可以运行在windows,linux,docker等平台。

- SQLite

> 广泛应用与嵌入式开发中。

- Sybase

> PowerDesginer数据库建模工具。

1.2.2 SQL语言

SQL（Structured Query Language：结构化查询语言）。

SQL是数据库查询和设计语言，用于存取数据、查询、更新、管理关系数据库。与其他程序设计语言的差别是，SQL由很少的关键字组成，每个SQL语言通过一个或多个关键字构成。

> SQL基于关系代数和元组关系演算，包括一个数据定义语言和数据操纵语言。SQL的范围包括数据插入、查询、更新和删除，数据库模式创建和修改，以及数据访问控制。尽管SQL经常被描述为，而且很大程度上是一种声明式编程（4GL），但是其也含有过程式编程的元素。

> SQL的优点：
>
> - 几乎所有的RDBMS都支持SQL。
> - SQL简单易学。
> - 使用方式灵活：SQL2种使用方式，可以直接以命令方式交互使用；也可以嵌入到其他程序设计语
>
> 言中使用(jdbc)
> - 非过程化：“做什么",不需要使用sql告诉计算机"怎么做"

> **注意：**
>
> SQL语句不区分大小写，对SQL中的关键字进行大写，而对表名、列名、数据库名称使用小写。可
>
> 以提高代码的阅读星和可维护性。

SQL包含四个部分：

- **DDL**(Data Defifinition Language)：**数据定义语言**

```
CREATE,DROP,ALTER
```

- **DML**(Data Manipulate Language)：**数据操纵语言**

```
INSERT,UPDATE,DELETE
```

- **DQL**(Data Query Lanaguage)：**数据查询语言**

```
SELECT
```

- **DCL**(Data Control Language)：**数据控制语言**

```
COMMIT,ROLLBACK
```

1.2.3 相关概念

- **表**

> **表（table）** 某种特定类型数据的结构化清单

> **模式（schema）** 关于数据库和表的布局及特性的信息

- **列和数据类型****

> **列（Colum）** 表中的一个字段。所有的表都是由一个或多个列组成的。

> **数据类型（datatype）** 所允许的数据的类型。每个表列都有相应的数据类型，它限制（或允许）该列中存储的数据。

- **行**

> 行（row） 表中的一个记录

- **主键**

> 主键（primary key） 一列（或一组列），其值能够唯一区分表中每个行

- **外键**

> 外键（foreign key）　父数据表（Parent Entity）的主键（primary key）会放在另一个数据表，当做属性以创建彼此的关系，而这个属性就是外键。

1.3. 非关系型数据库

NOSQL(Not Only SQL)是对不同于传统的**关系型数据库**的**数据库管理系统**的统称。

随着web2.0的兴起，传统的关系型数据库在处理海量数据时，会显得力不从心，从而产生了NOSQL，主流NOSQL，都才用KEY-VALUE的形式。

常见的非关系型数据库：Redis ,HBase,MongoDb,CouchDB

二、 MySQL基础

2.1. MySQL的安装与配置

2.1.1 下载MySQL

在官网，现在最新版的MySQL：https://dev.mysql.com/downloads/mysql/

将下载好的压缩文件，解压至【安装目录】, D:/DataBase/mysql。

如果提示缺少dll文件，去 https://www.microsoft.com/zh-cn/download/confirmation.aspx? id=48145 下载组件并安装。

2.1.2 初始化

在初始化时，需要使用已管理员身份启动的cmd,在 windows/system32/cmd.exe ，右键以管理员身份运行

```
#使用dos命令，进入到mysql的bin目录中
cd d:#进入
cd d:/DataBase/mysql/bin
#使用mysqld命令进行初始化
##进行初始化，同时创建随机的密码，并显示在控制塔中
mysqld --basedir=d:/DataBase/mysql --datadir=d:/DataBase/mysql/mysql_data --initialize --console
#随机生成的密码iiowpr<2felX
```

2.1.3 配置文件

将配置文件存储在 d:/DataBase/mysql/mysql_data/my.ini

```
[client]
default-character-set=utf8mb4
port            = 3306

[mysqld]
port            = 3306
basedir         = D:/DataBase/mysql
datadir         = D:/DataBase/mysql/mysql_data

character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci

default-time_zone='+8:00'
default-storage-engine=INNODB
default_authentication_plugin=mysql_native_password

max_allowed_packet = 256M
max_connections=20

[mysqldump]
quick
max_allowed_packet = 256M

[mysql]
default-character-set=utf8mb4
auto-rehash
```

2.1.4 安装服务

```
#在install后面可以添加安装的服务名称，默认使用MySQL作为服务名称
mysqld --install MySQL --defaults-file=D:/DataBase/mysql/mysql_data/my.ini
```

2.1.5 启动MySQL服务

- 使用命令方式启动

```
net start mysql
```

- 使用服务方式启动

  在运行中输入services.msc，在【服务】中右键MySQL，启动

2.1.6 登录MySQL

```
#客户端登录命令
mysql -h 主机地址 -u 用户名 -p 用户密码
mysql -uroot -p;
```

2.1.7 修改密码

通过命令修改root用户的密码

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '123456';
exit;
#退出
mysql -uroot -p123456;
```

2.1.8 删除服务

- 删除注册表（不推荐）
- 使用命令删除服务

```
SC DELETE mysql -- 服务的名称
```

2.2. 认识MySQL

2.2.1 MySQL介绍

MySQL是一个关系型的数据库管理系统，由MySQL AB公司开发，目前属于Oracle公司。

MySQL体积较小、速度快、成本较低、开放源代码。同时支持跨平台。MySQL集群搭建及分库分表。

> MySQL特性：
>
> - MySQL使用C和C++编写，保证了源代码的可移植性。
> - 跨平台：支持主流的操作系统（Windows,Linux,Mac os ,HP-UX)
> - 对编程语言的支持，对多种编程语言提供了API(C,C++,JAVA,PYTHON,PHP,Perl,Ruby)
> - 支持多线程，充分利用CPU资源。（服务器单核（A,B）
> - 优化的SQL查询算法，能提高查询效率。
> - 提供了TCP/IP,ODBC和JDBC等多种数据库连接途径
> - 支持多种存储引擎
> - 提供了用于管理，检查，优化数据库操作的管理工具。

2.2.2 MySQL服务端常用程序

- mysqld

  > mysqld是MySQL的后台程序（进程),只有该程序运行后，客户端才可以连接访问数据库。

- mysqld_safe

  > 也是服务器启动脚本。在Unix中使用mysqld_safe

- mysql.server

  > 服务器启动脚本。是通过调用mysqld_safe来启动mysql服务器的

- mysqld_multi

  > 服务器启动脚本，可以启动或停止系统中安装的多个mysql服务器

- mysql_install_db

  > 用于默认权限创建Mysql授权表，只能在系统首次安装mysql时执行，并且只执行一次。

2.2.3 MySQL客户端常用程序

- mysql

  > 通过交互式SQL语句输入来执行的命令行工具。（Index)

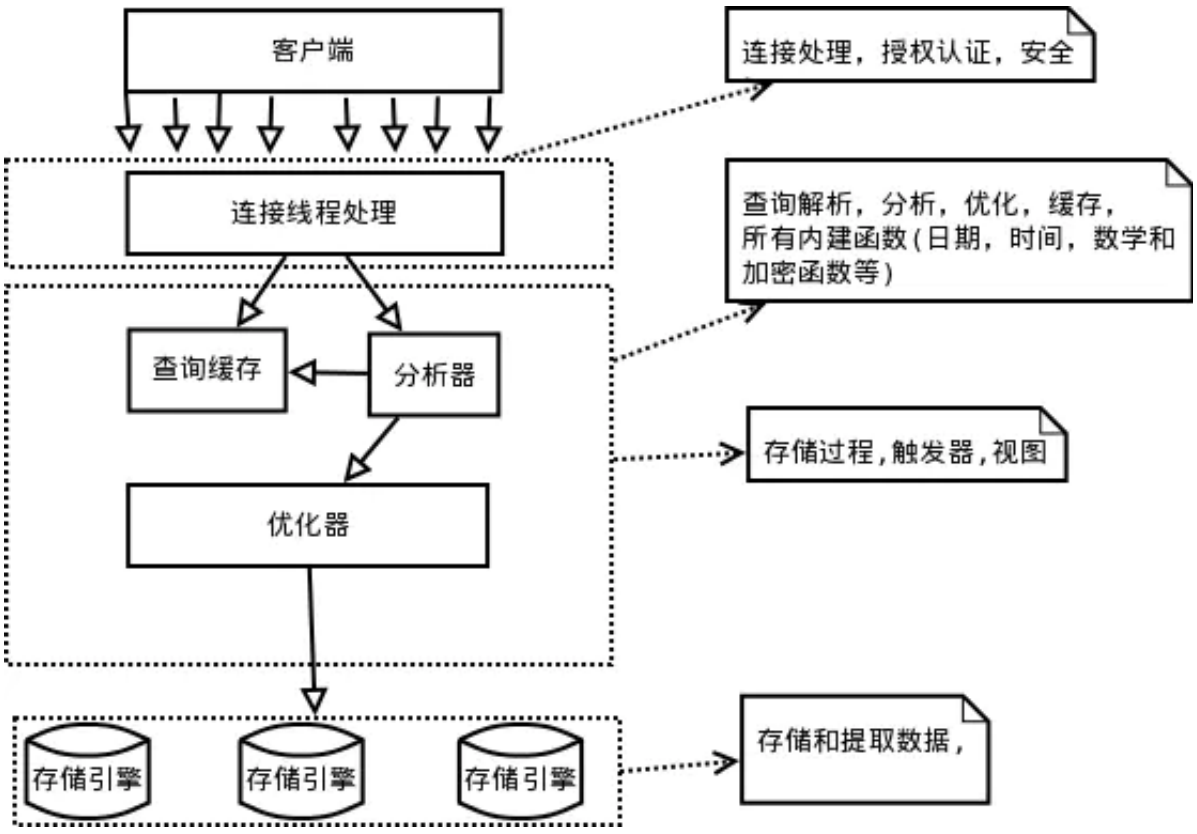
- mysqldump

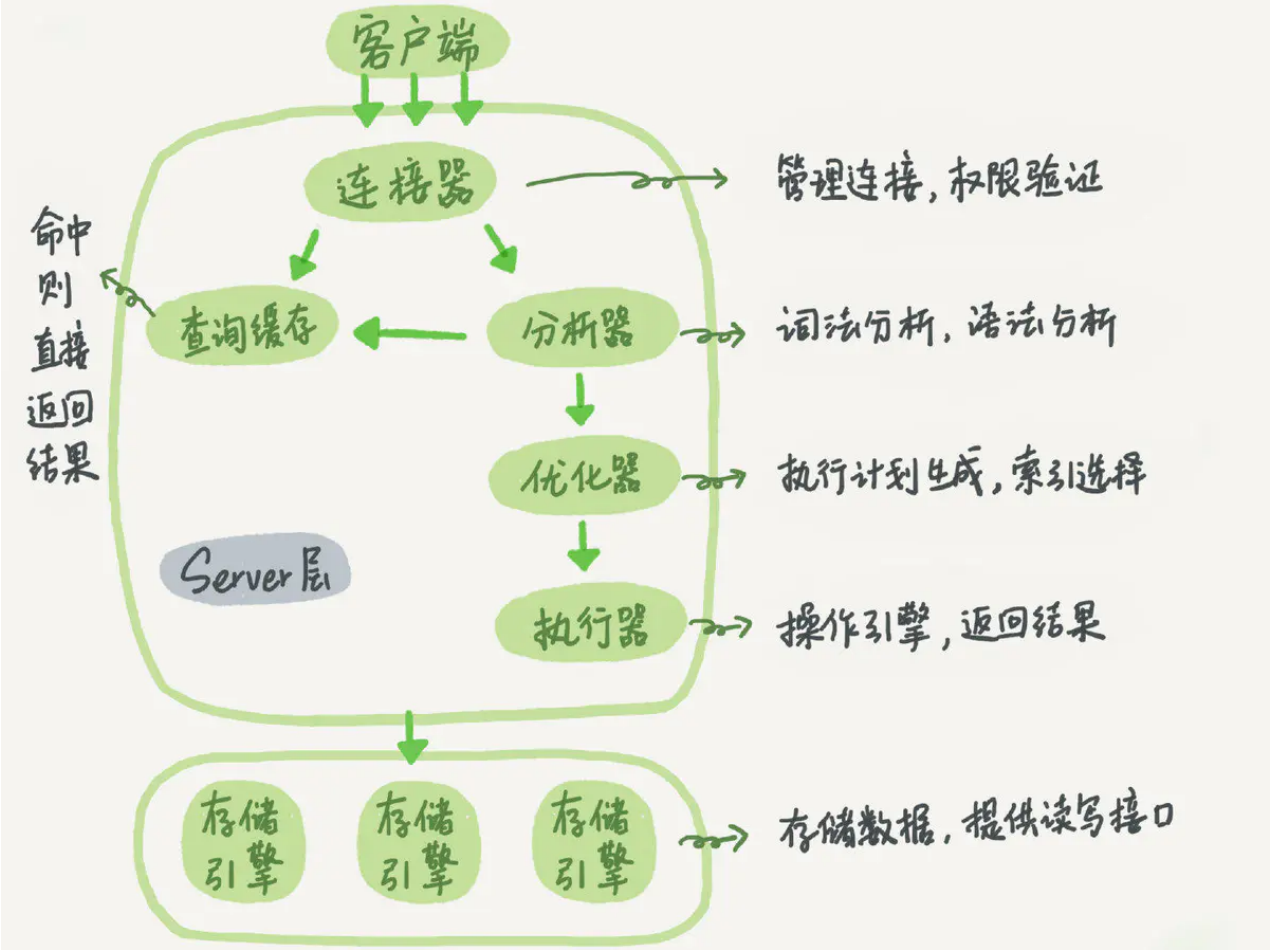
  > 用户备份数据的。（将MYSQL数据库转存至一个文件中.sql,.csv)

- mysqlshow
- error
- mysqlcheck
- mysqlhotcopy(MyISAM)

三、 MySQL架构

逻辑架构：





- 第一层为客户端的连接认证，C/S都有此架构
- 第二层为服务器层，包含MySQL的大多数核心服务功能
- 第三层包含了存储引擎，服务器通过API与其通信，API规避了不同存储引擎的差异，不同存储引擎也不会互相通信，另外存储引擎不会去解析SQL(InnoDB是例外，它会解析外键定义，因为服务器本身没有实现该功能)

四、 存储引擎

4.1. InnoDB

MySQ5.5及更高版本，默认存储引擎使用InnoDB，它提供了事务安全表（兼容ACID），支持外键引用的完整性约束。支持事务的提交，回滚和紧急数据恢复。它支持行级锁定。可以将数据存储在集群索引中，从而减少了基于主键查询的I/O次数

4.2 MyISAM

管理非事务性表，提高了存储和检索的效率，支持全文搜索。

4.3 MEMORY*

将数据存储在RAM中，数据的存储、查询更快

4.4 MERGE

将多个类似的MYISAM表分组为一个表，可以处理非事务性表。

4.5 EXAMPLE

开发人员学习如何变成存储过程，不能存储和查询数据

4.6 ARCHIVE

用于存储海量数据，单不支持索引

4.7 CSV

以,来分割数据并存储

4.8 BLACKHOLE

只接受数据，不存储数据

4.9 FEDERATED

将数据存储到远程数据库中

mysql数据库中分为行和列。数据在计算机上存储是以页为单位存储的。

五、 数据类型

数据类型是定义列中可以**存储什么数据**以及该数据**实际怎样存储**的基本规则

5.1 串数据类型

字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。

数据类型	大小	说明
CHAR	0~255 bytes	定长字符串,长度必须在创建时指定
VARCHAR	0~65535 bytes	变长字符串
TEXT	0~64K	长文本数据
TINYTEXT	0~255 bytes	短文本字符串
MEDIUMTEXT	0~16K	中等长度文本数据
LONGTEXT	0~4GB	极大文本数据
BLOB	0~64KB	二进制形式的长文本数据
TINYBLOB	0~255 bytes	不超过 255 个字符的二进制字符串
MEDIUMBLOB	0~16MB	二进制形式的中等长度文本数据
LONGBLOB	0~4GB	二进制形式的极大文本数据
ENUM		接受最多64K个串组成的一个预定义集合的某个串
SET		接受最多64个串组成的一个预定义集合的零个或多个串

```
/*
    TEXT: 存放富文本编辑器中的数据
    BLOB:用于存放二进制文本数据
*/
#1. 枚举
CREATE TABLE test_enum(
    n1 ENUM('a','b','c')
);
INSERT INTO test_enum(n1) values('a');
INSERT INTO test_enum(n1) values('b');
INSERT INTO test_enum(n1) values('c');
INSERT INTO test_enum(n1) values('d'); -- 超出枚举范围的会报错

#2. 集合类型 SET关键字声明
CREATE TABLE test_set(
    n1 SET('a','b','c')
```

```
);
insert into test_set(n1) values('a');
insert into test_set(n1) values('a,b');
insert into test_set(n1) values('a,b,c');
insert into test_set(n1) values('a,b,c,d'); -- 超出范围会报错
```

5.2 数值数据类型

5.2.1 整数类型（精确值）

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值

```
#默认情况下，整型时有符号
#以前版本超出范围只会保留临界值，8.0版本超出范围会报错
#在创建表时，可以通过unsigned来指定该整型为无符号的
CREATE TABLE test_int(
    n1 INT,-- id的范围-2 147 483 648，2 147 483 647
    n2 INT UNSIGNED -- 用unsigned关键字来指名该字段时无符号的
);

#整型数据的长度（大小）是由类型决定，整型后加括号，用于指定数据的显示宽度，一般与ZEROFILL关键字 一起使用，作用是在左侧自动填充0已达到自定宽度,一旦使用ZEROFILL，有符号就会变成无符号

CREATE TABLE test_int2(
```

```
    n1 INT(8) ZEROFILL,
    n2 INT UNSIGNED
);
```

5.2.2 定点类型（精确值）

类型	大小	范围（有符号）	范围（无符号）	用途
DECIMAL/NUMERIC	对DECIMAL(M,D)，如果M>D，为M+2 否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

5.2.3 浮点类型（近似值）

类型	大小	范围（有符号）	范围（无符号）	用途
FLOAT	4 bytes	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值

```
/*浮点型:
    float(M,D)
    double(M,D)
```

```
4   定点型
5       decimal(M,D)
6       numeric(M,D)
7       M:代表小数位+整数位的总长度 D:小数位的长度
8       默认情况下:  float,double,numeric,decimal会根据插入的数据来自动调整精度
    (float(10,3),10.000)
9       decimal默认MD为(10,0) 和整形一样，超出范围时，报错
10
11      定点型的精度比浮点型的精度高，设计到货币金额的字段一般都用定点型（decimal,numeric）
12  */
13  CREATE TABLE test_float( n1 float(10,3), n2 numeric(10,2) );
14  insert into test_float(n1,n2) values(11.1,12.2);
15  insert into test_float(n1,n2) values(11.11111,12.2);
16  insert into test_float(n1,n2) values(11.1,222.2222222);
17  insert into test_float(n1,n2)
    values(11.1,22222222221111111111111111111111111111111.0);
```

5.2.4 其他数值类型

数据类型	说明
BIT	位字段，1~64位
REAL	4字节的浮点值
BOOLEAN	布尔标志，或者为0或者为1，主要用于开/关标志

5.3 日期时间数据类型

数据类型	说明
DATE	表示1000-01-01~9999-12-31的日期，格式为YYYY-MM-DD
TIME	格式为HH:MM:SS
DATETIME	DATE和TIME的组合
TIMEDTAMP	功能和DATETIME相同（但范围较小）
YEAR	用2位数字表示，范围是70（1970年）~69（2069年），用4位数字表示，范围是1901年~2155年

> timestamp时间戳，微信公总号、支付宝开发过程中，参数是时间戳。timestamp能更好的体现时区，
>
> 如果项目对时区比较敏感，选择日期类型时推荐使用timestamp

```
CREATE TABLE test_date(
    n1 DATETIME
);
INSERT INTO test_date values('2020-01-01');
```

六、 SQL语句

6.1 基础语句

6.1.1 操作数据库

```sql
#查看mysql版本信息
SELECT VERSION();-- 系统函数，常量
#查看mysql中有哪些数据库
SHOW DATABASES;
#创建数据库
CREATE DATABASE t2;-- t2数据库名称
#切换数据库（默认登录后没有使用任何数据库，需要操作t2数据库
USE t2; -- t2需要切换的数据库
#删除数据库
DROP DATABASE t2;-- t2就是需要删除的数据库的名称
```

6.1.2 操作表

```sql
#创建表
/*语法:
    CREATE TABLE 表名(
    字段名 类型 【约束】,
    字段名 类型 【约束】, ...
    字段名 类型 【约束】
    );
*/
#查看有哪些表
SHOW TABLES;
#查看表结构
DESC test_int;-- test_int是表名
#修改表
ALTER TABLE test_int CHANGE COLUMN n2 n3 varchar(10); -- 将n2字段重命名为n3
#删除表
DROP TABLE test_int;
```

6.1.3 操作数据

```sql
#C 插入
-- INSERT INTO 表名(字段列表) VALUES(值列表)
INSERT INTO test_int2(n1,n2) VALUES(1,2);
#R 查询
SELECT * FROM test_int2;
#U 更新
-- UPDATE 表名 set 字段1=值1，字段2=值2 【WHERE 筛选】
UPDATE test_int2 set n2=100;
#D 删除
--  DELETE FROM 表名
DELETE FROM 表名
DELETE FROM test_int2;
```

6.1.4 导入导出

```sql
#项目开发环境和生产环境
##数据库的导出
mysqldump -uroot -p123456 java1908z > d:\java1908z.sql
##将生产环境中的数据库导入到开发环境中来
#1.创建数据库
CREATE DATABASE java1908z;
#2.切换数据库
USE java1908z;
#3.导入数据库
SOURCE D:\java1908z.sql;
```

6.2 DDL(Data Defifinition Language)：数据定义语言

6.2.1 关键字

- **CREATE**

  CREATE在数据库中创建一个对象，凡是数据库、数据表、数据库索引、存储程序、用户函数、触发程序或是用户自定义类型等对象，都可以使用**CREATE**指令来创建。

```
CREATE DATABASE d1;-- 创建数据库--
CREATE TABLE  t1;-- 创建数据表--
CREATE INDEX i1;-- 创建数据表索引--
CREATE PROCEDURE p1;-- 创建存储程序--
CREATE FUNCTION f1;-- 创建用户函数--
CREATE  TRIGGER  tr1;-- 创建触发程序--
```

- **ALTER**

  ALERT以不同方式修改现有对象的结构，相较于**CREATE**需要完整的数据对象参数，**ALERT**则是可以按照要修改的幅度来决定使用的参数。

```
ALTER TABLE doc_exa ADD column_b VARCHAR(20) NULL;-- 在表doc_exa中加入列,名称为cilumn_b,类型为varchar(20),允许为NULL--
ALTER TABLE doc_exb DROP COLUMN column_b ; -- 在表doc_exb中移出column_b列--
alter table scm.scm_d_pp_detail_1h  change  column PP_ID_2H PP_ID_1H varchar(50)  -- 列改名
alter table test modify address char(10) -- 修改表列类型--
```

- **DROP**

  DROP则是删除数据库对象的指令，并且只需要指定删除的数据库对象名称即可，在DDL语法中是最简单的。

```
DROP TABLE myTable;--删除myTable表--
DROP VIEW myView;--删除myView视图--
```

6.2.2 数据库的管理

```
#1. 创建数据库
#语法1: CREATE DATABASE 库名;
CREATE DATABASE t1;
#语法2: CREATE DATABASE IF NOT EXISTS 库名 -- 当库名不存在时，创建他，如果存在，不执行
CREATE DATABASE IF NOT EXISTS t1;
#2. 修改数据库
##一般不去修改数据库名称
```

```
##修改数据的编码格式
ALTER DATABASE t1 CHARACTER SET utf8;-- 修改t1的字符编码格式
#3. 删除数据库
#语法1: DROP DATABASE 库名;
DROP DATABASE t2;
#语法2:  DROP DATABASE IF EXISTS 库名 -- 当库名存在时删除，否则不执行
DROP DATABASE IF EXISTS t2;
```

6.2.3 数据表的管理

```
/*创建表
    CREATE TABLE 表名(
        列名 类型【(长度) 约束】, ...
        列名 类型【(长度) 约束】,
        【表约束】,
        【表约束】 )
  修改表:
    ALTER TABLE 表名 ADD|DROP|MODIFY|CHANGE COLUMN 列名【列的类型(长度) 约束】
  删除表:
    DROP TABLE 表名
 */
#1.创建表
CREATE TABLE b_company(
    id INT NOT NULL AUTO_INCREMENT COMMENT '主键ID',
    name VARCHAR(20) NOT NULL COMMENT '公司名称',
    record_date DATE COMMENT '注册时间',
    PRIMARY KEY(id)
);
#2.修改表名称
ALTER TABLE b_company RENAME TO bcompany;
RENAME TABLE b_company TO bcompany;
#3.修改name字段的名称 company_name
ALTER TABLE bcompany CHANGE COLUMN name company_name VARCHAR(30) COMMENT '公司注册名称';
#4.修改company_name字段的类型和约束
ALTER TABLE bcompany MODIFY COLUMN company_name VARCHAR(20) NOT NULL;
#5.添加列
ALTER TABLE bcompany ADD COLUMN fr_name VARCHAR(20) NOT NULL;
#6.删除列
ALTER TABLE bcompany DROP COLUMN fr_name;
#7.删除表
DROP TABLE bcompany;
```

```sql
DROP TABLE IF EXISTS bcompany;-- 加入bcompany存在，将其删除，否则不执行
#8.截断表（清空表，与DELETE的差别，truncate表时，自增主键重新开始计算，而删除时，不会从1开始
TRUNCATE TABLE bcompany;
#9.复制表结构
CREATE TABLE b_company LIKE bcompany;
#10.复制表结构并赋值数据
CREATE TABLE b_comapny2 SELECT * FROM bcompany;
#11.如果赋值部分表结构，不复制数据
CREATE TABLE b_company3 SELECT id,company_name FROM bcompany WHERE 0;
#12.复制部分表结构，并复制数据
CREATE TABLE b_company4 SELECT id,company_name FROM bcompany;
```

6.2.4 约束的管理

数据库管理系统不仅提供了数据的保存，数据的完整性，合法性进行限制，这种限制就叫约束。

常见约束

- **PRIMARY KEY 主键约束，**每张表中只能有一个主键约束。表中唯一一列可以确定一行的列称为主

  键。主键不能为空，表中可以没有主键。mysql中如果没有显示的定义主键，mysql InnoDB存储

  引擎会自动生成一个隐藏的自增主键。
- **NOT NULL 非空约束**，标识该列的数据不允许为空
- **DEFAULT 默认约束**
- **UNIQUE 唯一约束**，唯一约束的列可以为空
- **CHECK 检查约束**（检查你输入的数据是否满足我自定义的约束条件）
- **FOREIGN KEY（外键约束）**：外键约束是用于限制2张表之间的关系的，保证了外键字段中的数据

  来源必须源于另一张表。**主表中的非主键字段指向另一个表的主键字段。主表中的该非主键字段是一个外键。**

约束的添加时机

- 创建表时添加约束
- 修改表时添加约束

约束分类

- 列级约束：支持默认、非空、主键，唯一约束、检查约束，**不支持外键约束**

- 表级约束：CONSTRAINT 约束名 约束类型(字段名)，**一般不添加非空约束，默认约束**

```
#创建表时添加约束 使用列级约束
#用于存储王者荣耀（吃鸡）用户信息
CREATE TABLE b_user(
    id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    age INT DEFAULT 18,-- 默认值
    sex CHAR(1) CHECK(sex='男' OR sex='女'),
    mobile CHAR(11) UNIQUE,
    ipaddr INT
);
#在创建表时添加表级约束
CREATE TABLE b_user2(
    id INT NOT NULL AUTO_INCREMENT,-- 自增策略
    age INT DEFAULT 18, sex CHAR(1),
    mobile CHAR(11),
    ipaddr INT,
    -- 表级约束
    PRIMARY KEY(id),-- 增加了一个名为p的主键约束，但是主键约束默认的名称PRIMARY
    CONSTRAINT uq UNIQUE(mobile),
    CONSTRAINT fk FOREIGN KEY(ipaddr) REFERENCES s_ipaddr(id)-- 添加外键约束
);
INSERT INTO b_user(sex,mobile,ipaddr) values('男','1111',1);
INSERT INTO b_user(sex,mobile,ipaddr) values('未知','1111',2);
#服务器列表
CREATE TABLE s_ipaddr(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20)
);
```

```
#修改表时添加约束
CREATE TABLE b_user3(
    id INT,
    age INT,-- 默认值
    sex CHAR(1),
    mobile CHAR(11),
    ipaddr INT
);
#添加主键约束
ALTER TABLE b_user3 MODIFY COLUMN id INT NOT NULL PRIMARY KEY AUTO_INCREMENT;-- 修改
的列级约束
```

```
ALTER TABLE b_user3 ADD PRIMARY KEY(id);-- 表级约束
#添加默认约束
ALTER TABLE b_user3 MODIFY COLUMN age INT DEFAULT 18;
#添加检查约束【8.0不能修改检查约束】
ALTER TABLE b_user3 MODIFY COLUMN sex CHAR(1) CHECK(sex='男' or sex='女');
#添加唯一约束
ALTER TABLE b_user3 MODIFY COLUMN mobile CHAR(11) UNIQUE;-- 添加列级唯一约束
ALTER TABLE b_user3 ADD UNIQUE(mobile);-- 添加表级唯一约束
#添加外加约束
ALTER TABLE b_user3 ADD FOREIGN KEY(ipaddr) REFERENCES s_ipaddr(id);
```

```
#删除约束
/*可以使用列级约束的
    ALTER TABLE 表名 MODIFY COLUMN 字段名 字段类型
  可以使用表级约束的，并且有约束名称（PRIMARY KEY），
    ALTER TABLE 表名 DROP 约束类型 约束名
*/
#删除外键
ALTER TABLE b_user2 DROP FOREIGN KEY fk;
#删除主键
ALTER TABLE b_user3 DROP PRIMARY KEY;
#删除唯一约束
ALTER TABLE b_user3 MODIFY COLUMN mobile VARCHAR(11);
##DROP INDEX 唯一键的名称
ALTER TABLE b_user3 DROP INDEX mobile;
##查询b_users表中的index
SHOW INDEX FROM b_user3;
```

6.3 DML(Data Manipulation Language)：数据操纵语言

6.3.1 关键字

- **INSERT**

```
#单行插入
INSERT INTO 表名【(字段1，字段2，字段3)】 VALUES (值1,值2，值3)
##注意 值列表的个数和字段列表的个数完全一致，并且类型一致
INSERT INTO b_user(name) VALUES('张三');
INSERT INTO b_user VALUES(1,'李四','男','2020-01-01');
```

```
-- 如果表名后没有字段列表，VALUES后的值列表中的个数和表字段个数一致，并且值列表的顺序和字段列表的 顺序一致。一般如果主键列自增，不显式的给自增列赋值
-- 在值列表中字符、日期字段都应该使用单引号括起来
-- 如果表中有NOT NULL字段，该字段必须赋值
-- 如果表中没有NOT NULL，可以赋值，也可以不赋值（NULL）
#多行插入
INSERT INTO 表名【(字段1，字段2，字段3..)】 VALUES (值1，值2，值3...),(值1，值2，值3...),(值1，值2，值3...)
INSERT INTO b_user(name,sex,birthday) VALUES('王宝强','男','2020-01-01'),('王','男','2020-01-01'),('王1','男','2020-01-01');
#批量插入
##将一个查询结果全部插入到数据库中 INSERT INTO 表名【(字段1，字段2，字段3...)】(SELECT c1,c2,c3 FROM 表名 【WHERE 筛选条件】)
CREATE TABLE temp_user LIKE b_user;
INSERT INTO temp_user(name,sex,birthday) (SELECT name,sex,birthday FROM b_user WHERE user_id>2);

##注意:字段列表应该和SELECT后的显示字段列表个数，类型一致
```

了解

```
INSERT INTO 表名 SET 字段名=值，字段名=值
INSERT INTO temp_user SET name='kobe',sex='男',birthday='1976-01-01';
```

- **UPDATE**

```
/*语法:
    UPDATE 表名 SET 字段名=值,字段名=值 注意：最后一个没有逗号 WHERE 筛选条件
*/
#将temp_user表中的所有性别都改为男
UPDATE temp_user SET sex='女';
UPDATE temp_user SET sex='男' WHERE user_id=1 OR user_id=4;
```

- **DELETE**

```
/*语法:
    DELETE FROM 表名 WHERE 筛选条件 注意: 如果没有筛选条件, 等于清空整张表
*/
#删除ID=3的用户信息
DELETE FROM temp_user WHERE user_id=3;
##企业开发过程中谨慎的去使用delete语句
#1.先用查询语句
SELECT * FROM temp_user WHERE user_id=2;
#2.将SELECT *改为DELETE
DELETE FROM temp_user WHERE user_id=2;
##TRUNCATE (截断表)和DELETE的差别
#1. TRUNCATE在截断表时, 自增列重新从1开始自增, DELETE不影响自增列
#2. DELETE支持事务回滚, 而TRUNCATE不支持事务回滚
```

6.4 DQL(Data Query Lanaguage)：数据查询语言

6.4.1 关键字

```
SELECT
```

6.4.2 基础查询

```
/*
#基础语法
    SELECT
        查询字段列表（最后一个字段不加逗号）
    FROM
        表名
#执行顺序
    先执行 FROM 再执行SELECT
注意:
    查询字段列表中: 字段, 函数, 表达式, 常量 , 标量子查询
    查询结果是一张虚拟的二维表, 不能对查询结果进行更新和删除。
*/
#查询常量
SELECT 1;
#查询函数
SELECT VERSION();
#表达式
```

```sql
SELECT 100+200;
SELECT 1>2;-- mysql中没有boolean类型，0代表false,1代表true
#查询单个字段
SELECT name FROM temp_user;
#多字段查询
SELECT * FROM temp_user;
SELECT name,sex FROM temp_user;-- 在企业开发过程中推荐使用
#多个字段+常量+函数+表达式
SELECT name,sex,2,VERSION(),100+200,user_id*2 FROM temp_user;
#IFNULL(参数1，参数2)，参数1放字段名，参数2放参数1字段为空时，你需要替换的值
SELECT name,IFNULL(birthday,'未知') FROM temp_user;
#别名设置 AS 关键字设置，别名可以用单引号括起来，也可以不加单引号
SELECT name,IFNULL(birthday,'未知') AS 'birthday'FROM temp_user;
#别名设置，省去AS关键字，别名可以用单引号括起来，也可以不加单引号
SELECT name,IFNULL(birthday,'未知') birthday FROM temp_user;
#去重 DISTINCT
SELECT DISTINCT name FROM temp_user;
#字符拼接(在oracle和sqlserver中字符拼接直接用+可以拼接)
##语法:CONCAT(字符1，字符2)，将字符1和字符2拼接
SELECT CONCAT('篮球',name) AS name FROM temp_user;
```

6.4.3 条件查询

```sql
/*
语法结构:
    SELECT
        查询字段列表
    FROM
        表名
    WHERE
        筛选条件
筛选条件:
    1.条件表达式: >,>=,<,<=,=,!=,<>,IS NULL,IS NOT NULL
    2.逻辑表达式: AND,OR,NOT
    3.模糊查询: LIKE,BETWEEN AND,IN
条件查询的执行顺序:
    1.FROM
    2.WHERE
    3.SELECT
*/
#条件表达式
##查询年龄>18的运动员
```

```
SELECT * FROM temp_user WHERE age>18;
##查询年龄不等于18
SELECT * FROM temp_user WHERE age!=18;
SELECT * FROM temp_user WHERE age<>18;
#注意：对于NULL值的条件表达式，只能使用IS NULL或IS NOT NULL
#查询生日不为空的
SELECT * FROM temp_user WHERE birthday IS NOT NULL;
SELECT * FROM temp_user WHERE birthday IS NULL;
##如果筛选条件中包含2个或以上个条件表达式，我们应该在条件表达式中间使用逻辑表达式
#查询年龄>18的并且生日不能为空
SELECT * FROM temp_user WHERE age>18 AND birthday IS NOT NULL;
#查询年龄<=18 或生日为空的
SELECT * FROM temp_user WHERE age<=18 OR birthday is null;
SELECT * FROM temp_user WHERE NOT(age>18 AND birthday IS NOT NULL);
#模糊查询
##模糊查询的语法:SELECT 查询字段列表 FROM 表名 WHERE 字段名 LIKE '通配符+字符'
###2种通配符
#### %代表任意（0或多个）个任意字符
SELECT * FROM temp_user WHERE name LIKE 'kobe%';-- 查询已kobe开头的用户
SELECT * FROM temp_user WHERE name like '%kobe';-- 以kobe结尾的用户
SELECT * FROM temp_user WHERE name like '%kobe%';-- 包含kobe的用户
#### _代表1个任意字符，如果要表示_字符，使用\_
SELECT * FROM temp_user WHERE name like 'kobe_';-- 是以kobe开头，但是应该是5个字符
kobe_ SELECT * FROM temp_user WHERE name like '_kobe';-- 以kobe结尾，前面只能有一个任意字
符
SELECT * FROM temp_user WHERE name like '_kobe_';-- 包含kobe，前后只能有一个字符
##BETWEEN AND 闭合区间查询
###语法： BETWEEN 值1 AND 值2 == 字段>=值1 AND 字段<=值2
SELECT * FROM temp_user WHERE user_id BETWEEN 1 AND 2;
SELECT * FROM temp_user WHERE user_id>=1 AND user_id <=2;
#IN子句可以使用or来替换
##查询user_id 在1,3,4中的数据
SELECT * FROM temp_user where user_id=1 OR user_id=3 OR user_id=4;
SELECT * FROM temp_user WHERE user_id in(1,3,4);
##注意：在in的集合中不允许出现通配符
SELECT * FROM temp_user WHERE user_id in(1,3,%); -- 错误写法
```

注意

> 在实际开发中筛选条件中的条件表达式，逻辑表达式，模糊查询经常会混合使用，在混合使用时，使用()来提升优先级

在WHERE子句中使用()

```
#查询年龄>18并且name是已kobe开头的,或者年龄<18 并且name是包含kobe的，或者年龄=18 并且name=kobe
SELECT * FROM temp_user WHERE (age>18 AND name like 'kobe%') OR (age<18 AND name like '%kobe%') OR (age=18 AND name='kobe');
```

6.4.4 排序

对查询结果进行顺序调整

```
/*
基本语法:
    SELECT
        查询字段列表
    FROM
        表名
     【
    WHERE
        筛选条件
    】
    ORDER BY
        排序字段列表
注意:
    1.默认排序使用ASC升序排列（不加ASC|DESC),DESC降序（数字（1,2,3），字符（根据各个国家的 排序情        况））
    2.多个字段，按出现的先后顺序排列
    3.在排序字段列表中支持：单个字段，多个字段，表达式，函数，别名
    4.ORDER BY 放在查询语句的最后（除了LIMIT）
执行顺序
    1.FROM
    2.WHERE
    3.SELECT
    4.ORDER BY
*/
CREATE TABLE n1(
    n1 INT PRIMARY KEY,
    n2 INT,
    n3 INT,
```

```
    n4 INT,
    n5 VARCHAR(20)
);
INSERT INTO n1 values(1,2,3,4,'a');
INSERT INTO n1 values(5,2,3,4,'a');
INSERT INTO n1 values(2,1,3,4,'b');
INSERT INTO n1 values(3,5,3,4,'c');
INSERT INTO n1 values(4,3,2,4,'d');
# 按n1进行升序排列
SELECT * FROM n1 ORDER BY n1+1 ASC;
# 按n1进行降序排列
SELECT n1 as tn1,n2,n3,n4,n5 FROM n1 ORDER BY n1 DESC;
# 按n2,3升序排列
SELECT * FROM n1 ORDER BY n2,n3;
```

6.4.5 单行函数

```
/*
语法:
    SELECT 单行函数（参数列表）
    FROM 表名
    WHERE 单行函数
*/
```

6.4.5.1 字符串函数

函数	描述
CONCAT(str1,str2,...)	字符串连接
LENGTH(str)	返回str的长度
UPPER(str)	返回str的大写
LOWER(str)	返回str的小写
SUBSTR(str,indexNum)/SUBSTRING(str,index)	截取str字符串，从indexNum位开始（索引以1开始）
substr(str,pos,length)/substring(str,pos,length)	截取str字符串，从pos开始，截取length长 度
instr(str,substr)	返回substr第一次在str中出现的索引位置，找不见返回0
trim(str)	去除str首尾的空格
LPAD(str,length,padStr)	使用padStr在str左侧填充，使str的长度为length
RPAD(str,length,padStr)	使用padStr在右侧填充，使str的长度为length
REPLACE(str,from_str,to_str)	返回字符串str，字符串的所有匹配项from_str替换为字符串to_str。当搜索from_str时，REPLACE()执行区分大小写匹配。

```sql
SELECT CONCAT(firstname,lastname) AS name FROM S_USER;
SELECT LENGTH('HELLOWORLD');
SELECT UPPER('helloWorLD');
SELECT LOWER('HELLOWORLD');
SELECT SUBSTR('HELLOWORLD',2);
SELECT SUBSTRING('HELLOWORLD',2);
SELECT SUBSTR('HELLOWORLD',2,2);
SELECT SUBSTRING('HELLOWORLD',2,2);
SELECT INSTR('HELLOWORLD','EL');-- indexOf("EL"),返回-1
SELECT TRIM(' H ');
SELECT TRIM('O' FROM 'OHOHO');-- 去除以'O'开头或结尾的
SELECT LPAD('HE',4,'|');
SELECT RPAD('HE',4,'|');
SELECT REPLACE('www.mysql.com','w','WW');
```

6.4.5.2 数字函数

函数	描述
MOD()	取余数
ABS()	返回绝对值
CEIL/CEILING	向上取整
FLOOR	向下取整
ROUND(X)	四舍五入取整数
ROUND(X,D)	D代表四舍五入后的小数位数
TRUNCATE(X,D)	按指定的小数位D截断X

```
SELECT 5%2;
SELECT MOD(5,2);-- 取余正负看被除数
```

6.4.5.3 日期函数

函数	描述
NOW()	返回当前的日期时间
CURDATE()	返回当前的日期
CURTIME()	返回当前的时间
YEAR(DATE/str)	返回当前日期的年份
MONTH(date)	返回日期所在的月份
DAY(date)/DAYOFMONTH	返回日期所在的天
DAYNAME(DATE)	返回星期
LAST_DAY(DATE)	返回当前日期所在月的最后一天
DATE_FORMAT(DATE,format_str)	date按照format_str的形式格式化
STR_TO_DATE(str,format)	str按照format的格式转为日期
DATE_ADD	时间增加
DATEDIFF	日期减少

```
SELECT YEAR(NOW());
SELECT YEAR('2020-05-10');
SELECT MONTH(NOW());
SELECT DAY(NOW());
SELECT DAYNAME(NOW());
SELECT DATE_FORMAT(NOW(),'%Y年%m月%d日'); -- %Y 2020 %y 20
SELECT STR_TO_DATE('2020年05月10日','%Y年%m月%d日');
```

6.4.5.4 其他函数

函数	描述
VERSION()	查看数据库版本
DATABASE()	查看当前数据库名称
USER()	查看当前用户

```
SHOW DATABASES;-- 查看有哪些数据库
SELECT DATABASE();-- 查看当前数据库名称
```

6.4.5.5 流程控制函数

- IF三目运算

  > IF(逻辑表达式,值1代表逻辑表达式成立，值2)

  ```
  -- age字段，不要求显示年龄，要求显示是否为未成年人
  SELECT name,IF(age>18,'成年人','未成年人') FROM b_user;
  ```

- CASE函数　switch case

  ```
  -- 等值判断
  CASE 字段名
      WHEN 值 THEN 显示值
      WHEN 值 THEN 显示值
      ..
      ELSE 值
      END
  SELECT name,
      CASE age
      WHEN 18 THEN '青年'
      WHEN 55 THEN '中年'
      WHEN 65 THEN '老年'
      ELSE '少年'
      END
  FROM b_user;
  ```

- CASE 函数　if-else

```
CASE
    WHEN 条件表达式 THEN 结果1
    WHEN 条件表达式 THEN 结果2
    ....
    ELSE 结果n
    END
```

```
-- 根据年龄，来显示少年，青年，中年，老年
-- 少年<18
-- 青年>=18 and <=44
-- 中年>=45 and <=59
-- 老年>=60
SELECT name,
    CASE
    WHEN age<18 THEN '少年'
    WHEN age>=18 and age<=44 THEN '青年'
    WHEN age>=45 and age<=59 THEN '中年'
    ELSE '老年'
    END AS age1,age
FROM b_user;
```

6.4.6 分组函数

也称为统计函数、聚集函数

```
/*
分组函数:
    1. SUM 求和
    2. MAX 求最大值
    3. MIN 求最小值
    4. AVG 求平均值
    5. COUNT 统计行数
注意:
    1. 数据类型
        SUM,AVG:一般处理数值型字段 MAX,MIN,COUNT:可以处理任意类型的字段
    2. NULL处理
        都会忽略NULL值
    3. distinct
```

```
14          都可以和distinct搭配使用,sum(a1)--sum(distinct a1)
15      4. 统计行数
16          COUNT(列名): 当列值为null时，忽略该行
17          COUNT(*): 只要该行有任意一列不为null，该行就会被统计
18          COUNT(1):相当于给查询结果多加了一个常量列,
19
20          InnoDB下: COUNT(*)和COUNT(1)执行效率接近【在sqlserver中count(1)的效率要高于
    count(*)的效率】
21          MYISAM下: count(*)效率最高 通常在mysql中使用count(*)来统计行数。
22      5. 与分组函数一同出现在查询字段列表中的字段，要求必须出现在GROUP BY 的字段列表中
23  */
24  -- 查询所有用户的年龄总和
25  SELECT SUM(age) FROM b_user;
26  -- 求平均年龄
27  SELECT AVG(age) FROM b_user;
28  SELECT SUM(age)/COUNT(age) FROM B_USER;
29  -- 求最大年龄
30  SELECT MAX(age) FROM b_user;
31  -- 最小年龄
32  SELECT MIN(age) FROM b_user;
33  -- 总人数
34  SELECT count(*) FROM b_user;
```

6.4.7 分组查询

一般用于统计分析中

```
1   /*
2   语法:
3       SELECT
4           分组函数，列（要求出现在分组字段列表中）
5       FROM 表名
6       【WHERE 筛选条件】
7       GROUP BY 分组字段列表
8       【HAVING 筛选条件 】
9       【ORDER BY 排序字段列表 】
10  注意:
11      (***重要***)
12      1.查询字段列表中的字段必须出现在分组字段列表中，如果没有出现在分组字段列表中，必须使用聚合函数
13      2.WHERE 是在分组前进行筛选，处理的是原始数据
14      3.HAVING 是在分组和进行筛选，处理的是分组后的数据结果
15      4.GROUP BY 后面可以是字段、函数,别名
16  执行顺序:
```

```
    1. FROM
    2. WHERE
    3. GROUP BY
    4. HAVING
    5. SELECT
    6. ORDER BY
*/
-- 按照性别，统计总总年龄
SELECT sex,SUM(age) FROM b_user GROUP BY sex;
-- 按照性别，统计总年龄，单不统计年龄<=13
CREATE TABLE b_student(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    realname VARCHAR(20),
    sex CHAR(1),
    classId int,
    score NUMERIC(12,2)
);
INSERT INTO b_student(realname,sex,classId,score) values('张三','男',1,98.5);
-- 插入数据
-- 查询男生和女生的最高成绩、评价成绩
SELECT sex,MAX(score),AVG(score) FROM b_student GROUP BY sex;
-- 及格的男生女生的最高成绩和平均成绩
SELECT sex,MAX(score),AVG(score) FROM b_student WHERE SCORE>=60 GROUP BY sex;
-- 查询男生和女生平均成绩高于80的信息
SELECT sex,MAX(score),AVG(score) FROM b_student GROUP BY sex HAVING AVG(score)>=80;
```

6.4.8 连接查询

MySQL连接查询分类

1. 按推出时间划分
    - SQL92标准：仅支持内连接
    - SQL99标准（推荐）：支持除了全外连接的所有连接类型
2. 按功能划分
    - 内连接
        - 等值连接
        - 非等值连接
        - 自连接
    - 外连接
        - 左外连接
        - 右外连接
        - 全外连接（MySQL不支持FULL OUTER JOIN）
    - 交叉连接（没有关联关系等价于笛卡尔连接）

内连接

```
-- SQL92内连接
SELECT 查询字段列表
FROM 表1, 表2
WHERE 表1和表2关联关系

-- SQL99内连接
SELECT 查询字段列表
FROM 表1 别名1 INNER JOIN 表2 别名2 ON 表1和表2的关联关系

-- 查询订单信息，并关联显示用户姓名
-- 等值连接
-- SQL92
SELECT o.*,u.name
FROM b_order o,b_user u
WHERE o.user_id = u.user_id
-- SQL99
SELECT o.*,u.name
FROM b_order o INNER JOIN b_user u ON o.user_id=u.user_id;
-- 非等值内连接
-- SQL99
SELECT o.*,u.name
FROM b_order o INNER JOIN b_user u ON o.user_id between u.user_id and u.age;
-- 自连接
-- SQL99
# 自连接一般用于一张表中有上下级关系的表，一般会有一个指向上级的字段，用法和等值连接一致，我们将同一张表起步同的别名进行等值连接
SELECT *
FROM china province inner join china city ON province.id = city.pid
```

外连接

```
#左外连接
SELECT <SELECT_LIST> FROM TABLE A LEFT JOIN TABLE B ON A.KEY=B.KEY
-- 注意:
-- 1. 确定主表
-- 2. 在左外连接中，LEFT 左边的就是主表
-- 3. 显示所有的主表记录，并关联显示从表中的数据，如果从表中没有和主表可以关联的数据，使用NULL 进行匹配

##查询订单信息，并关联信息用户姓名
```

```
SELECT o.*,u.name
FROM b_order o LEFT JOIN b_user u ON o.user_id = u.user_id;
-- b_order表中3条数据，并关联用户信息时，订单编号为00003的没有用户姓名(NULL)
SELECT o.*,u.name
FROM b_order o LEFT JOIN b_user u ON o.user_id = u.user_id
WHERE u.user_id is NULL;

# 右外连接
SELECT <SELECT_LIST>
FROM TABLE A RIGHT JOIN B ON A.KEY=B.KEY
-- 注意:
-- 1. 确定主表
-- 2. 在右外连接中，RIGHT 右边的就是主表
-- 3. 显示所有的主表记录，并关联显示从表中的数据，如果从表中没有和主表可以关联的数据，使用NULL 进行匹配

##查询订单信息，并关联信息用户姓名
SELECT o.*,u.name
FROM b_user u RIGHT JOIN b_order o ON u.user_id = o.user_id;

SELECT o.*,u.name
FROM b_user u RIGHT JOIN b_order o ON u.user_id = o.user_id
WHERE u.user_id is NULL;

-- 注意:在实际开发中，我们需要用数据量较小的表作为主表（小表驱动大表），这样可以减少从表和主表匹配数据的时间，从而提升查询效率
```

CROSS连接

```
SELECCT <SELECT_LIST>
FROM TABLE A CROSS JOIN B
-- 等价于笛卡尔积
SELECT <SELECT_LIST>
FROM TABLEA CROSS JOIN B
WHERE A.KEY=B.KEY
```

6.4.9 联合查询

联合查询是指将多个查询结果合并成一个结果集（二维表），通常出现在统计分析中。

```
/*
联合查询语法:
    查询语句1
    UNION
    查询语句2
    ...
    查询语句n
注意:
    1.所有查询语句返回结果的列数必须相等
    2.每列的数据类型必须一致，【查询语句1中字段列表的类型必须和查询语句2中的字段列表类型对应且 一致】
*/

-- 查询所有用户的信息并显示总年龄
SELECT user_id,name,sex,birthday,age FROM b_user
UNION
SELECT 0,'合计','','',SUM(age) FROM b_user;
```

6.4.10 子查询

所谓子查询是指嵌套在另一个SQL语句内部中的查询语句

子查询的分类：

- 按结果集的行数
  - 标量子查询（单行子查询，结果集只有一行一列）
  - 列子查询（多行子查询，结果集有多行一列）
  - 行子查询（结果集有多行多列）
  - 表子查询（结果集有多行多列）
- 按出现的位置
  - SELECT之后：只能出现标量子查询
  - FROM之后：表子查询，**查询结果必须起别名**
  - WHERE | HAVING 之后：支持标量子查询、列子查询、行子查询
  - EXISTS之后：支持表子查询

```sql
#查询订单信息，并显示用户姓名
SELECT a.*,(SELECT name FROM b_user WHERE user_id=a.user_id) FROM b_order a
#查询所有用户信息
SELECT * FROM b_user;
SELECT * FROM (SELECT * FROM b_user) a;
#查询李四购买的订单信息
SELECT * FROM b_order WHERE user_id=(SELECT user_id FROM b_user WHERE name='李四');
```

关键字	描述
IN / NOT IN	条件等于列表中的任意一个值在/不在列表中。IN子句后只能加标量子查询或列子查询
ANY / SOME	和子查询返回的某个值比较（可以使用=,>,<,>=,<=,!=）
ALL	和子查询返回的所有值比较
EXISTS / NOT EXISTS	看子查询是否成立，如果子查询有返回结果，显示查询内容，否则返回空

```sql
#IN子句
-- 查询平台购买过商品的用户（查询用户表，只要用户的user_id在b_order表中，满足条件）
SELECT * FROM b_user WHERE user_id IN (SELECT user_id FROM b_order);
-- 查询未在平台购买过商品的用户
SELECT * FROM b_user WHERE user_id NOT IN(SELECT user_id FROM b_order);

# ANY|SOME
SELECT s1 FROM t1 WHERE s1 > ANY|SOME (SELECT s1 FROM t2);
-- t1中有5条记录，每一行的s1去和(select s1 from t2)每一行s1去比较，只要有t1中的s1大于t2中 的任意一个s1，那么当前行满足查询条件，

# ALL
SELECT s1 FROM t1 WHERE s1 > ALL(select s1 FROM t2);
-- t1中有5条记录，每一行的s1去和(select s1 from t2)每一行s1去比较，必须t1中的s1大于t2中的 所有的s1，那么当前行满足查询条件，

# EXISTS
SELECT * FROM b_user WHERE EXISTS (SELECT * FROM b_order WHERE order_id>10);
-- (select * from b_order where order_id>10)有返回结果，执行select* from b_user;
-- select * from b_order where order_id>10没有返回结果，执行select* from b_user返回空
```

6.4.11 分页查询

如果数据量过大(100亿)，如果一次性显示10亿条数据，（100亿条数据本身从数据库中读取时慢**【分库分表】**，将100亿条新闻展示在网页的过程也是很慢的）将100亿条新闻拆分开，每次给用户显示10条。

- 手工分页

  百度新闻、微商城、淘宝这些根据滚动条的位置来刷新数据

- 滚动条分页

```
/*
基础语法:
    SELECT 查询字段列表
    FROM 表名
    WHERE 筛选条件
    GROUP BY 分组列表
    HAVING 筛选条件
    ORDER BY 排序列表
    LIMIT offset,size offset:代表查询的起始索引，从0开始 size:你需要显示的条数
注意:
    如果offset是从0开始，可以省略
*/
#查询前2条数据 S
SELECT * FROM b_user LIMIT 0,2;
-- 如果offset为0
SELECT * FROM b_user LIMIT 2;
```

注意：在SQLServer中使用top关键字进行分页。例如 top 7，代表查询前7条数据。

6.5 DCL(Data Control Language)：数据控制语言

6.5.1 关键字

- **commit**
- **rollback**

6.5.2 事务的四大特性（ACID）

> **事务**：事务是指数据库中的**一组逻辑操作（包含有1条或多条相关的SQL语句）**，这组逻辑操作中所有的SQL语句要么全部执行成功，要么全部执行失败。

- 原子性（Atomicity）

  指事务是最小单位，不可分割，事务中的所有操作要么全部成功，要么全部失败。

- 一致性（Consistency）

  事务必须使数据库从一个状态到另一个状态

- 隔离性（Isolation）

  一个事务在执行过程中不受其他事务的干扰

- 持久性（Durability）

  事务一旦提交，数据就会被持久化的数据库中

6.5.3 数据库的并发问题

对于同时运行的多个事务，当这些事务访问数据库中相同的数据时，如果没有采取其他必要的隔离机制，就会导致以下问题：

- 写问题
- 读问题

  > - **脏读**：有两个事务T1,T2，T1读取了T2更新但**没有被提交**的数据。如果T2回滚后，T1读取的内容是无效的。
  > - **不可重复读**：有两个事务T1,T2，T1读取一个字段，然后T2**更新**该字段。之后T1再去读取同一字段，T1事务两次读取同一字段读到的数据是不一样的。
  > - **幻读**：有两个事务T1和T2，T1从一个表中读取了一个字段，然后T2在该表中**插入**了一些数据。之后T1再次读取同一个表，就会出现不同行数。

数据库事务的隔离性：

数据库必须具有隔离并发运行各个事务的能力，使他们不会相互影响，尽量避免各种并发问题。

隔离级别：

一个事务与其他事务隔离的程度称为隔离级别。数据库中规定了4种隔离级别，不同的隔离界别对于不同的干扰程度，**隔离级别越高，数据一致性越好，但是并发越差。**

6.5.4 四种隔离级别

隔离级别	描述
READ UNCOMMITED（读未提交数据）	允许事务读取其他事务未提交的数据，可能出现脏读、不可重复读、幻读
READ COMMITED（读已提交数据）	允许事务读取其他事务已经提交的数据，解决了脏读问题但是可能出现不可重复读、幻读的问题
REPEATABLE READ（可重复读）	确保一个事务可以多次从一个字段中读到相同的值，在事务执行期间内禁止其他事务对该字段的更新。解决了脏读和不可重复读的问题，但是可能出现幻读的问题
SERIALIZABLE（串行化）	确保一个事务可以从一个表中读取相同的行。在事务执行期间禁止其他事务对该表进行插入、更新、删除操作。解决了所有事务并发时读的问题，但是性能十分低下。

- MySQL支持4种隔离级别，默认使用的是REPEATABLE READ（可重复读）
- Oracle支持2种隔离级别，READ COMMITTED ,SERIALIZABLE，默认使用READ COMMITTED
- Sql Sever支持4种隔离级别，默认使用READ COMMITTED（读已提交）

6.5.5 设置隔离级别

每次启动一个MySQL程序时，都会获取一个单独的数据库连接，每个连接都有一个全局变量用于记录数据库当前的隔离级别 @@transation_isolation

- 查看当前连接数据库隔离级别

```
SELECT @@transaction_isolation;
```

- 设置当前连接数据库隔离级别

```
-- 设置当前连接的隔离级别
SET SESSION transaction isolation level READ COMMITTED;
```

- 设置数据库全局的隔离级别

```
SET GLOBAL transaction isolation level READ COMMITTED;-- 不推荐修改
```

6.5.6 隐式事务

默认情况下MySQL、Oracle、Sql Sever**开启了自动提交的事务**

```
#查看事务
show variables like 'autocommit';
```

6.5.7 显示事务

显示事务是**需要手动开启事务，并且提交事务的**。在使用显示事务之前必须将隐式事务修改为显示事务。

```
#将隐式事务改为显式事务
SET autocommit=0;
# 开启事务
START TRANSACTION;
INSERT INTO t1(s1) VALUES(1);
INSERT INTO t1(s1) VALUES(2);
# 事务提交
COMMIT;
# 事务回滚
ROLLBACK;-- 将INSERT,UPDATE,DELETE到数据库中的数据，恢复为原样。对已经提交的数据没有办法回滚。
```

七、 索引

索引是帮助数据库高效获取数据的数据结构

7.1 索引分类

7.1.1 按数据结构

- B树索引：MySQL大多数存储引擎支持B树索引（InnDB使用B+树）
- 哈希索引：memory引擎支持哈希索引

> Sql Server的索引使用的是**平衡二叉树**

7.1.2 按存储类型

- 非聚集索引：
  - 索引文件和数据文件时分离的；
  - 就是以非主键创建的索引（也叫二级索引）
  - 索引的逻辑顺序与磁盘上行的物理存储顺序不同；
  - 非聚集索引在叶子节点存储的是主键和索引列；
- 聚集索引：
  - 表数据文件本身就是按B+Tree 组织的一个索引结构，树的叶节点data 域保存了完整的数据记录；
  - 就是以主键创建的索引；
  - 聚集索引表记录的排列顺序和索引的排列顺序一致；
  - 聚集索引在叶子节点存储的是表中的数据；

7.1.3 按应用层次

- 普通索引：索引定义的列可以出现空值或重复值。
- 唯一索引：索引上的列可以出现空值，但是不能出现重复值。
- 主键索引：在创建主键时会自动创建主键索引（聚集索引），主键列，非空、唯一。
- 复合索引：在多个字段上创建的索引。
- 全文索引：在某一个字段中查找包含的值，允许为空，允许重复。
- 空间索引：myiam支持空间索引。

7.2 创建索引

7.2.1 普通索引

- 直接创建

```
/*
语法:
    CREATE INDEX 索引名称 ON 表名(字段名称(长度))
注意:
    如果索引所在的字段为char,varchar类型，长度可以小于实际长度，也可以省略。如果字段为text或 blob字段，必须指定长度
*/
-- 在年龄上定义普通索引
CREATE INDEX index_age ON b_user(age);
```

- 修改表时创建

```
/*
语法:
    ALTER TABLE 表名 ADD INDEX 索引名称(字段名称)
*/
-- 修改表时创建索引
ALTER TABLE b_user ADD INDEX b_user(birthday);
```

- 创建表时创建索引（开发中一般不使用该方式）

```
CREATE TABLE index_test(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username varchar(20),
    pwd varchar(20),
    INDEX index_username(username)
);
```

7.2.1 唯一索引

- 直接创建

```
/*
    CREATE UNIQUE INDEX 索引名称 ON 表名(字段名)
*/
CREATE UNIQUE INDEX uq_pwd ON index_test(pwd);
```

- 修改表时创建

```
/*
    ALTER TABLE 表名 ADD UNIQUE 索引名称(字段名称)
*/
ALTER TABLE index_test ADD UNIQUE uq2_pwd(pwd);
```

- 创建表时创建索引

```
CREATE TABLE index_test1(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username varchar(20),
    pwd varchar(20),
    UNIQUE uq_pwd(pwd)
);
```

7.2.1 主键索引

在创建主键时会自动创建主键索引，主键索引不是主键。如果在一个表中没有主键，MySQL会自动创建一个隐藏的的自增列。

7.2.1 复合索引

```
/*
    CREATE INDEX 索引名称 ON 表名(字段(长度),字段(长度)...)
*/
CREATE INDEX index_uname_pwd on index_test(username,pwd);
```

7.2.1 全文索引

- 直接创建

```
CREATE FULLTEXT INDEX 索引名称 ON 表名(字段(长度))
```

- 修改表时创建

```
ALTER TABLE 表名 ADD FULLTEXT 索引名称(字段(长度))
```

- 创建表时创建

```
CREATE TABLE index_test2(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(2000),
    pwd VARCHAR(20),
    FULLTEXT index_username(username(20))
);
```

7.3 删除索引

```
DROP INDEX 索引名称 ON 表名
```

7.4 如何添加索引（索引使用规则）

- 在不含NULL值的列上添加索引
- 使用短索引：

  长度varchar(2000)，发现该字典钱10-20个字符内，多数是唯一的，这时你在添加索引时，不需要将整列添加索引。短索引不仅可以提高查询效率，同时节省磁盘空间并且减少I/O操作。
- 索引排序：

  一般将需要排序的字段作为索引，在开发时尽量不要使用包含多个列的排序，如果需要多列排序，建议在多个排序列上使用复合索引。
- like操作：

  不推荐使用like，如果使用like，不使用like '%aa%'，而使用like 'aa%'
- 不要在列上运算（条件中）

  在条件中使用运算，会导致索引失效，从而进行全表扫描。
- 不要在索引上使用NOT IN和<>操作
- 字符串不加单引号容易索引失效
- 少用or，在某些引擎或者版本下or并不会匹配对应的单值索引，尽量使用union

7.5 索引的缺点

- 虽然索引可以提高查询效率，但是会降低更新效率
- 索引也会占用磁盘空间，如果在一张表上创建了多种组合索引，索引文件会增长很快

7.6 InnnDB索引（B+树）

7.6.1 B+树基本特点

- 非叶子节点的子树指针与关键字个数相同。
- 非叶子节点的子树指针 P[i]，指向关键字属于 **[k[i],K[i+1])** 的子树（**注意：区间是前闭后开**）。
- **为所有叶子节点增加一个链指针**。
- **所有关键字都在叶子节点出现**。

这些基本特点是为了满足以下的特性。

7.6.2 B+树的特性

- 所有的关键字 **都出现在叶子节点的链表中**，且链表中的关键字是有序的。
- **搜索只在叶子节点命中**。
- 非叶子节点相当于是 **叶子节点的索引层**，叶子节点是 **存储关键字数据的数据层**。

7.6.3 相对 B 树，B+树做索引的优势

- B+树的磁盘读写代价更低。**B+树的内部没有指向关键字具体信息的指针**，所以其内部节点相对 B 树更小，如果把所有关键字存放在同一块盘中，那么盘中所能容纳的关键字数量也越多，一次性读入内存的需要查找的关键字也就越多，**相应的，IO 读写次数就降低了**。
- **树的查询效率更加稳定**。B+树所有数据都存在于叶子节点，所有关键字查询的路径长度相同，每次数据的查询效率相当。而 B 树可能在非叶子节点就停止查找了，所以查询效率不够稳定。
- **B+树只需要去遍历叶子节点就可以实现整棵树的遍历**。

八、 视图

视图是一张虚拟的表，本身并不存储数据，是通过普通的SQL查询来动态生出数据的。

在企业开发中不允许对视图进行INSERT、UPDATE、DELEAE

8.1 视图的使用场景

- 多个程序中使用了相同的SQL语句，可以用相同的查询作为视图
- 如果在查询中使用了复杂的SQL语句

8.2 视图的优点

- 简化了程序中的复杂的查询

```
String sql ="select * from ( select * from ( select * from c where t1='a') b where t2='c') a";
String sql1="select * from 视图A";
```

- 有助于限制对特定用户的数据访问
- 试图提供了额外的安全层（可以在视图中定义是否有查询权限）
- 解耦合

8.3 视图的缺点

- 性能：视图的查询效率较慢，特别是视图中调用视图（视图是基于视图创建）
- 表依赖关系耦合度较高：当我们修改基础表和预期相关联的表的表结构时，我们必须要更新视图

8.4 视图的创建

```
/*
创建语法:
    CREATE VIEW 视图名称 AS 查询语句

使用语法:
    SELECT 查询字段列表
    FROM 视图名称
    WHERE
    GROUP BY
    HAVING
    ORDER BY
    LIMIT

修改视图:
    1. ALTER VIEW 视图名称 AS 查询语句
    2. CREATE OR REPLACE VIEW 视图名称 AS 查询语句

查看视图结构:
    DESC 视图名称

删除视图:
    DROP VIEW 视图名称列表
*/
```

案例：在电商项目中有大量的根据用户id来查询对应的订单总金额和其所购买的商品的功能。

```sql
-- 视图的创建
CREATE VIEW view_user_order AS
    SELECT a.*,c.name
    FROM b_order a
    LEFT JOIN b_order_detail b on a.order_id=b.order_id
    LEFT JOIN b_goods c on b.goods_id=c.goods_id;

-- 使用
SELECT * FROM view_user_order;

-- 视图修改
ALTER VIEW view_user_order AS
    SELECT a.*,c.name AS goodsname
    FROM b_order a
    LEFT JOIN b_order_detail b on a.order_id=b.order_id
    LEFT JOIN b_goods c on b.goods_id=c.goods_id;

-- 修改视图2
CREATE OR REPLACE VIEW view_user_order AS
    SELECT a.*,c.name AS goods_name
    FROM b_order a
    LEFT JOIN b_order_detail b on a.order_id=b.order_id
    LEFT JOIN b_goods c on b.goods_id=c.goods_id;
```

九、 触发器

> 触发器是与表有关的数据库对象，**在满足定义条件时触发并执行触发器中定义语句集合**

9.1 触发器的特性

- 有begin end结构体（包含多条SQL语句）
- 指定触发的条件：INSERT、UPDATE、DELETE
- 指定触发的时间：BEFORE、AFTER
- 指定触发的频率：FOR EACH ROW
- 触发器定义在表上

9.2 触发器的创建

- 单条业务逻辑的触发器创建

```
/*单条业务逻辑触发器语法:
    CREATE TRIGGER 触发器名称 BEFORE|AFTER INSERT|UPDATE|DELETE ON 表名
    FOR EACH ROW
    业务逻辑
*/

#当b_user表中插入数据后，b_log表中也插入一条数据
CREATE TRIGGER tigger_inser AFTER INSERT ON b_user
FOR EACH ROW
INSERT INTO b_log(comments) VALUES('插入数据');
```

- 多条业务逻辑触发器创建

```
/*
    DELIMITER $
    CREATE TRIGGER 触发器名称 BEFORE|AFTER INSERT|UPDATE|DELETE ON 表名
    FOR EACH ROW
    BEGIN
    INSERT....;
    UPDATE....;
    END;$
*/
#注意 DELIMITER $

#在b_user表中插入数据前，b_log表中插入2条数据
DELIMITER $
CREATE TRIGGER trigger_insert_before BEFORE INSERT ON b_user
FOR EACH ROW
BEGIN
INSERT INTO b_log(comments,name) values('insert1',NEW.name);
INSERT INTO b_log(comments) values('insert2',NEW.name);
END;$
```

总结:

- DELIMITER $作用类似于在jQuery中的$符的让渡

- 在INSERT型触发器中，NEW用来表示将要（BEFORE）或已经（AFTER）插入的新数据；
- 在UPDATE型触发器中，OLD用来表示将要或已经被修改的原数据，NEW用来表示将要或已经修改为的新数据；
- 在DELETE型触发器中，OLD用来表示将要或已经被删除的原数据；

9.3 触发器的删除

```
DROP TRIGGER 触发器名称
```

十、 存储过程

10.1 变量

10.1.1 系统变量

由MySQL DBMS提供的，变量名称固定，可以查看和修改值，分为**全局变量**和**会话变量**。

> **全局变量**：在MYSQL启动的时候由服务器自动将它们初始化为默认值，这些默认值可以通过更改my.ini这个文件来更改
>
> **会话变量**：在每次建立一个新的连接的时候，由MYSQL来初始化。MySQL会将当前所有全局变量的值复制一份。来做为会话变量。
>
> 也就是说，如果在建立会话以后，没有手动更改过会话变量与全局变量的值，那所有这些变量的值都是一样的。
>
> 全局变量与会话变量的区别就在于，对全局变量的修改会影响到整个服务器，但是对会话变量的修改，只会影响到当前的会话（也就是当前的数据库连接）。
>
> 全局变量用**global**来形式，而会话变量用**session**,通常session是可以省略的。

- 查看系统变量

```
#查看全局变量
SHOW GLOBAL variables;
SHOW GLOBAL variables like '%dir%';-- 模糊查询环境变量

#查看会话变量
SHOW SESSION variables;
SHOW variables;

SELECT @@datadir;
SELECT @@session_track_transaction_info;
```

- 修改系统变量

```
SET global autocommit=0;-- 全局的自动提交的事务改为手动提交
SET SESSION autocommit=0;-- 修改会话变量

SET @@session.autocommit=1;
SET @@global.autocommit=0;
```

总结：

1. **全局变量**在修改后，在不同的会话中都会立即生效。但是在重新启动MySQL服务后全局变量会恢复为默认值。如果想让全局变量依旧有效，需要去修改.ini文件（MySQL配置文件）。
2. **会话变量**在修改后只对当前会话有效。一般在开发过程中修改会话变量。如：字符编码格式等可以在.ini 文件中进行设置。

10.1.2 自定义变量

MySQL允许用户自定义变量，可分为**用户变量**和**局部变量**。

- 用户变量：

> **作用域**：当前会话有效

```
#设置方式1，先去声明并初始化用户变量，赋值操作既可以使用=进行赋值，也可以使用:=进行变量赋值
SET @变量名=值;
SET @变量名:=值;
SELECT @变量名:=值;
# 设置用户变量
SET @a='helloWorld';
SELECT @a;
SET @b:='sofwin';
select @b;
SELECT @c:='helloworld sofwin';
SELECT @a,@b,@c;
#设置方式2
SELECT 字段 into @变量名 FROM 表名 SELECT count(*) into @d FROM b_user; select @d;
```

- 局部变量：

> **作用域**：在begin end的结构体中。必须是begin end结构体的第一句

```
#声明方式,必须在begin后面从第一行开始
DECLARE 变量名 类型;
DECLARE 变量名 类型 DEFAULT 值;

-- 例子
BEGIN
DECLARE a1 INT;
DECLARE b1 VARCHAR(20);
END;
# 局部变量的赋值
SET 变量名:=值;
SELECT @变量名:=值;
SELECT 字段 into 变量名 From 表名;

# 例子
DELIMITER $
CREATE TRIGGER test BEFORE INSERT ON b_user
FOR EACH ROW
BEGIN
DECLARE a1 INT DEFAULT 11;
DECLARE a2 INT DEFAULT 12;
```

```
22    DECLARE a3 INT;
23    SET a3:=a1+a2;
24    SELECT a3;
25    END;$
```

10.2 存储过程的特性

存储过程是一组已经预先编译好的SQL语句集，理解为批处理语句（增加流程控制语句），一般在复杂的业务逻辑中才会使用存储过程。

存储过程思想上很简单，就是数据库 SQL 语言层面的代码封装与重用。

优点：

- 提高了代码的可重用性
- 简化了数据库操作，将业务逻辑的细节隐藏在存储过程中
- 减少了编译次数，减少了网络I/O次数，从而提高了操作效率

10.3 存储过程的创建

```
1   /*
2   存储过程的创建语法:
3       DELIMITER $
4       CREATE PROCEDURE 存储过程的名称（参数列表）
5       BEGIN
6       局部变量的定义
7       多条SQL语句
8       流程控制语句
9       END;$
10  */
11  -- 如果存储过程中只有一条SQL语句，可以省略begin    end
```

参数列表：

参数模式	形参名称	参数类型
IN	username	mysql数据库中的数据类型（数值型、字符型、日期型）
OUT	pwd	mysql数据库中的数据类型（数值型、字符型、日期型）
INPUT	xxx	mysql数据库中的数据类型（数值型、字符型、日期型）

- **IN**：声明该参数是一个输入型参数（类似于java中的形参）
- **OUT**：声明该参数为一个输出型参数（类似于java的返回值），在一个存储过程中可以定义多个out类型的参数。
- **INPUT**：声明该参数既可以为输入型参数，也可以为输出型参数

10.4 存储过程的调用

```
CALL 存储过程名称(实参列表)
-- 实参列表中包含有输出类型的参数
-- CALL pro1(@a,@b,@c);
-- Object a = Util.a();
```

10.5 存储过程的演示

- 无参数的存储过程

```
# 用于向b_user表中插入2条数据
DELIMITER $
CREATE PROCEDURE pro_insert()
BEGIN
INSERT INTO b_user(user_name,sex) VALUES('存储过程1','男');
INSERT INTO b_user(user_name,sex) VALUES('存储过程2','女');
END;$

CALL pro_insert();
```

- 创建带有IN模式参数的存储过程

```
# 用于向b_user表中插入2条数据，性别由客户输入
DELIMITER $ CREATE PROCEDURE pro_insert2(IN sex CHAR(1))
BEGIN
INSERT INTO b_user(name,sex) VALUES('存储过程22',sex);
INSERT INTO b_user(name,sex) VALUES('存储过程33',sex);
END;$

CALL pro_insert2('男');
```

- 创建带有多个IN参数的存储过程

```
# 用于向b_user表中插入2条数据，用户名和密码由客户输入
DELIMITER $ CREATE PROCEDURE pro_insert3(IN name varchar(10),IN sex
varchar(20))
BEGIN
INSERT INTO b_user(name,sex) VALUES(name,sex);
INSERT INTO b_user(name,sex) VALUES(name,sex);
END;$

CALL pro_insert3('uname','男');
```

- 创建带有IN、OUT参数的存储过程

```
# 判断用户登录，如果用户名和密码输入正确登录成功，否则登录失败
# 根据输入的用户名和密码作为条件去b_user表中查询，如果查询总行数==1，则认为登录成功，让
result 的值=登录成功，否则登录失败
DELIMITER $ CREATE PROCEDURE pro_login(IN name varchar(20),IN pwd
varchar(20),OUT result varchar(20))
BEGIN
DECLARE total INT DEFAULT 0; -- 用于存放查询总行数
SELECT COUNT(*) INTO total FROM b_user a WHERE a.name=name and a.pwd=pwd;-- 将查
询结果赋值给total局部变量
SET result:=IF(total=1,,'登录成功','登录失败');
END;$

```

```
# 存储过程如何执行
-- 解决办法使用自定义变量
SET @result:='';
CALL pro_login('李四','123',@result);
SELECT @result;
```

- 删除存储过程

```
DROP PROCEDURE 存储过程名称
```

- 查看存储过程

```
SHOW CREATE PROCEDURE 存储过程名称;
```

- 修改存储过程

```
DROP -- 删除
CREATE -- 创建
```

10.6 流程控制语句

10.6.1 选择结构

- IF函数
  - 功能：三目运算
  - 语法：IF(逻辑表达式，表达式1，表达式2)：当逻辑表达式成立时执行表达式1，否则执行表达式2
- IF结构
  - 功能：实现多路选择
  - 语法：

```
IF 逻辑表达式
THEN 语句1;
ELSEIF 逻辑表达式2
THEN 语句2;
...
ELSE 语句N;
END IF;
```

- 注意：只能用在begin　end结果体中

- CASE结构
  - 等值选择

```
CASE 字段|变量|表达式
WHEN 值 THEN 值|语句
WHEN 值 THEN 值|语句
...
ELSE 值|语句
END
```

  - 不等值选择

```
CASE
WHEN 逻辑表达式 THEN 语句1
WHEN 逻辑表达式 THEN 语句2
.....
ELSE 语句N
END
```

10.6.2 循环结构

- WHILE

```
/*
    WHILE 逻辑表达式 DO
    循环体
    END WHILE;
```

```
*/
#需求：创建存储过程，输入一个值，返回1到该值的和（3）
# 分析：一个输入参数，一个返回值，在结构体中，从1循环到输入的值，并求和
DELIMITER //
CREATE PROCEDURE pro_sum(IN input INT,OUT total INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE sum_ INT DEFAULT 0;
WHILE i<=input DO
SET sum_:=sum_+i;
SET i:=i+1;
END WHILE;
SET total:=sum_;
END;//

SET @result:=0;
call pro_sum(10,@result);
select @result;
```

- LOOP

```
/*
loopname:LOOP

    IF 逻辑表达式 THEN
    LEAVE loopname;-- 跳出当前指定的loopname循环，类似于java中的break
    END IF;
END LOOP;
#loopname定义的循环名称，为了跳出循环时指定跳出的循环
*/
DELIMITER //
CREATE PROCEDURE pro_sum_loop(IN input INT,OUT total INT)
BEGIN
DECLARE i int DEFAULT 1;
DECLARE sum_ int DEFAULT 0;
a:LOOP
SET sum_:=sum_+i;
SET i:=i+1;
IF i>input
THEN LEAVE a;
END IF;
END LOOP;
```

```
SET total:=sum_;
END;//

SET @result:=0;
call pro_sum_loop(3,@result);
select @result;
```

- REPEAT

```
/*
    REPEAT
    循环体
    UNTIL 逻辑表达式 -- 当满足逻辑表达式时，跳出循环
    END REPEAT;
*/
#求1到输入值的和
DELIMITER //
CREATE PROCEDURE pro_sum_repeat(IN input INT,OUT total INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE sum_ INT DEFAULT 0;
REPEAT
SET sum_:=sum_+1;
SET i:=i+1;
UNTIL i>input
END REPEAT;
SET total:=sum_;
END;//

SET @result:=0;
call pro_sum_loop(3,@result);
select @result;
```

十一、 函数

函数也是一组预先编译好的SQL语句集，基本和存储过程相似

函数和**存储过程**的区别：

- 存储过程可以有0个或多个返回值，适用于INSERT、UPDATE、DELETE操作

- 函数只能有1个返回值，适用于在处理数据之后，返回一个已知的结果

11.1 创建函数

```
#函数创建的语法
CREATE FUNCTION 函数名称(参数列表) RETURNS 返回类型 BINLOG参数
BEGIN
函数体
END
```

参数列表：参数名称 参数类型

BINLOG参数：

- NO SQL：函数体中没有SQL语句，也不会修改数据。
- READS SQL DATA：函数体中存在SQL语句，但是整个数据是只读的，不会修改数据。
- MODIFIES SQL DATA：函数体中存在SQL语句，并且会修改数据。
- CONTAINS：函数体中包含有SQL语句。

函数体：函数体中必须包含return语句，将return放在函数体的最后一行执行。

```
# 写一个函数用于求2数之和
DELIMITER //
CREATE FUNCTION sum_(input1 INT,input2 INT) RETURNS INT NO SQL
BEGIN
RETURN input1+input2;
END;//
```

11.2 使用函数

```
SELECT 函数名(参数列表);
SELECT sum_(10,20);
```

11.3 查看函数

```
SHOW CREATE FUNCTION 函数名;
```

11.4 删除函数

```
DROP FUNCTION 函数名;
```

十二、 数据库设计

范式是符合某种设计要求的总结，关系型数据库中有6种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴德斯科范式（BCNF）、第四范式（4NF）、第五范式（5NF）

一般在企业开发中，只需要满足第三范式。

12.1 数据库设计三大范式

1. 第一范式（1NF）

   每个字段不可分割，数据库表中的每一列都是原子项。

id	name	address
1	张三	山西省太原市小店区XXXX小区

   要求统计小店区的用户：返回看该表，address列不具有原子性，可以分割

id	name	province	city	county	addr
1	张三	山西省	太原市	小店区	XXXX小区

   注意：在企业开发中要求所有的表都必须满足第一范式。

2. 第二范式（2NF）

所有的非主键列完全依赖于主键列，第二范式是建立在第一范式的基础上，要满足第二范式必须满足第一范式。当表中使用联合主键时，才可能出现不满足第二范式的情况。

学号	课程号	成绩	本课学分

本课学分只依赖于课程编号，不依赖于学号，所以本课学分不满足第二范式

解决：

课程表：

课程号	本课学分

成绩表：

学号	课程号	成绩

3. 第三范式（3NF）

   满足第二范式的基础上消除传递依赖，一个表中的非主键字段不能依赖于该表中的其他非主键字段

学号	姓名	性别	年龄	班主任编号	班主任姓名	班主任年龄

学号是主键，姓名、性别、年龄，班主任编号都直接依赖于学号，班主任姓名和班主任年龄没有直接依

赖于学号，依赖的是班主任编号（非主键字段）

解决：

学生表：

学号	姓名	性别	年龄	班主任编号

老师表：

班主任编号	班主任姓名	班主任年龄

总结：在实际开发中，所有表的设计都必须满足第一范式要求，可以不满足第二范式和第三范式。

性能的优先级高于规范的优先级（当性能与规范冲突时以性能为主），特别是电商项目中，为了满足性能要求去违背设计规范。

比如：软件工程中挖掘用户需求。20W的项目，项目需要1年。国企项目（不差钱）缺管理思路

12.2 E-R图

- 矩形（实体或表）
- 椭圆（表中的属性）
- 菱形（实体和实体之间的关系）

数据库中实体和实体之间的关系有一对一，一对多的关系，多对多的关系

其中一对一的关系和一对多的关系可以使用外键设计来完成。

- 一对一

  用户登录表、用户信息表（淘宝中）

  用户登录表：

id	手机号	密码	信息表id（外键）

  用户信息表：

id	真实姓名	地址	身份证号

- 一对多

  用户表，部门表（以用户为参照，一个用户只能有一个部门，如果参照为部门，一个部门可以有多个用

  户）

id	手机号	密码	部门id（外键）

部门表：

id	名称	编号	负责人

- 多对多，需要中间表去实现该关系

  角色表、权限表

  角色表：

id	name

权限表：

id	name	addr

角色权限表：

id（自增主键）	roleId(角色ID)	menuId(权限ID)
1	1	1
2	1	2
3	2	1
4	2	2

12.3 PowerDesigner建立模型，生成数据库

电商项目的表基本在200张表，使用工具画ER图，确定了实体及实体属性，并且也确定实体间的关系，使用DDL语句去进行数据库设计。

> 真实在开发过程中，先确定实体及实体属性（ER图），建立物理模型（能够直观的感受到实体和实体间的关系，并且能够清晰的看到实体中的属性。

可以从概念模型-----物理模型。

1. 模型建立
2. 当前DBMS的配置、
3. 生成SQL文件