

原文地址: <https://shockerli.net/post/1000-line-mysql-note/> , JavaGuide 对本文进行了简答排版, 新增了目录。作者: 格物

非常不错的总结, 强烈建议保存下来, 需要的时候看一看。

- [基本操作](#)
- [数据库操作](#)
- [表的操作](#)
- [数据操作](#)
- [字符集编码](#)
- [数据类型\(列类型\)](#)
- [列属性\(列约束\)](#)
- [建表规范](#)
- [SELECT](#)
- [UNION](#)
- [子查询](#)
- [连接查询\(join\)](#)
- [TRUNCATE](#)
- [备份与还原](#)
- [视图](#)
- [事务\(transaction\)](#)
- [锁表](#)
- [触发器](#)
- [SQL编程](#)
- [存储过程](#)
- [用户和权限管理](#)
- [表维护](#)
- [杂项](#)

基本操作

```
/* Windows服务 */
-- 启动MySQL
    net start mysql
-- 创建Windows服务
    sc create mysql binPath= mysqld_bin_path(注意: 等号与值之间有空格)
/* 连接与断开服务器 */
mysql -h 地址 -P 端口 -u 用户名 -p 密码

SHOW VARIABLES -- 显示系统变量信息
```

SHOW PROCESSLIST -- 显示哪些线程正在运行

数据库操作

```

/* 数据库操作 */ -----
-- 查看当前数据库
    SELECT DATABASE();
-- 显示当前时间、用户名、数据库版本
    SELECT now(), user(), version();
-- 创建库
    CREATE DATABASE[ IF NOT EXISTS] 数据库名 数据库选项
    数据库选项:
        CHARACTER SET charset_name
        COLLATE collation_name
-- 查看已有库
    SHOW DATABASES[ LIKE 'PATTERN']
-- 查看当前库信息
    SHOW CREATE DATABASE 数据库名
-- 修改库的选项信息
    ALTER DATABASE 库名 选项信息
-- 删除库
    DROP DATABASE[ IF EXISTS] 数据库名
    同时删除该数据库相关的目录及其目录内容

```

表的操作

```

-- 创建表
    CREATE [TEMPORARY] TABLE[ IF NOT EXISTS] [库名.]表名 ( 表的结构定义 )[ 表选项]
    每个字段必须有数据类型
    最后一个字段后不能有逗号
    TEMPORARY 临时表, 会话结束时表自动消失
    对于字段的定义:
        字段名 数据类型 [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'string']
-- 表选项
    -- 字符集
        CHARSET = charset_name
        如果表没有设定, 则使用数据库字符集
    -- 存储引擎
        ENGINE = engine_name
        表在管理数据时采用的不同的数据结构, 结构不同会导致处理方式、提供的特性操作等不同
        常见的引擎: InnoDB MyISAM Memory/Heap BDB Merge Example CSV MaxDB Archive
        不同的引擎在保存表的结构和数据时采用不同的方式
        MyISAM表文件含义: .frm表定义, .MYD表数据, .MYI表索引
        InnoDB表文件含义: .frm表定义, 表空间数据和日志文件
        SHOW ENGINES -- 显示存储引擎的状态信息
        SHOW ENGINE 引擎名 {LOGS|STATUS} -- 显示存储引擎的日志或状态信息
    -- 自增起始数
        AUTO_INCREMENT = 行数
    -- 数据文件目录
        DATA DIRECTORY = '目录'
    -- 索引文件目录
        INDEX DIRECTORY = '目录'
    -- 表注释

```

```

        COMMENT = 'string'
    -- 分区选项
        PARTITION BY ... (详细见手册)
-- 查看所有表
    SHOW TABLES[ LIKE 'pattern']
    SHOW TABLES FROM 库名
-- 查看表结构
    SHOW CREATE TABLE 表名 (信息更详细)
    DESC 表名 / DESCRIBE 表名 / EXPLAIN 表名 / SHOW COLUMNS FROM 表名 [LIKE
'PATTERN']
    SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
-- 修改表
    -- 修改表本身的选项
        ALTER TABLE 表名 表的选项
        eg: ALTER TABLE 表名 ENGINE=MYISAM;
    -- 对表进行重命名
        RENAME TABLE 原表名 TO 新表名
        RENAME TABLE 原表名 TO 库名.表名 (可将表移动到另一个数据库)
        -- RENAME可以交换两个表名
    -- 修改表的字段机构 (13.1.2. ALTER TABLE语法)
        ALTER TABLE 表名 操作名
        -- 操作名
            ADD[ COLUMN] 字段定义          -- 增加字段
                AFTER 字段名                -- 表示增加在该字段名后面
                FIRST                        -- 表示增加在第一个
            ADD PRIMARY KEY(字段名)         -- 创建主键
            ADD UNIQUE [索引名] (字段名) -- 创建唯一索引
            ADD INDEX [索引名] (字段名) -- 创建普通索引
            DROP[ COLUMN] 字段名            -- 删除字段
            MODIFY[ COLUMN] 字段名 字段属性 -- 支持对字段属性进行修改, 不能修改字
段名(所有原有属性也需写上)
            CHANGE[ COLUMN] 原字段名 新字段名 字段属性 -- 支持对字段名修改
            DROP PRIMARY KEY                -- 删除主键(删除主键前需删除其AUTO_INCREMENT属性)
            DROP INDEX 索引名              -- 删除索引
            DROP FOREIGN KEY 外键          -- 删除外键
-- 删除表
    DROP TABLE[ IF EXISTS] 表名 ...
-- 清空表数据
    TRUNCATE [TABLE] 表名
-- 复制表结构
    CREATE TABLE 表名 LIKE 要复制的表名
-- 复制表结构和数据
    CREATE TABLE 表名 [AS] SELECT * FROM 要复制的表名
-- 检查表是否有错误
    CHECK TABLE tbl_name [, tbl_name] ... [option] ...
-- 优化表
    OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
-- 修复表
    REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ... [QUICK]
[EXTENDED] [USE_FRM]
-- 分析表
    ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...

```

数据操作

```

/* 数据操作 */ -----
-- 增
    INSERT [INTO] 表名 [(字段列表)] VALUES (值列表)[, (值列表), ...]
        -- 如果要插入的值列表包含所有字段并且顺序一致, 则可以省略字段列表。
        -- 可同时插入多条数据记录!
        REPLACE 与 INSERT 完全一样, 可互换。
    INSERT [INTO] 表名 SET 字段名=值[, 字段名=值, ...]
-- 查
    SELECT 字段列表 FROM 表名[ 其他子句]
        -- 可来自多个表的多个字段
        -- 其他子句可以不使用
        -- 字段列表可以用*代替, 表示所有字段
-- 删
    DELETE FROM 表名[ 删除条件子句]
        没有条件子句, 则会删除全部
-- 改
    UPDATE 表名 SET 字段名=新值[, 字段名=新值] [更新条件]
    
```

字符集编码

```

/* 字符集编码 */ -----
-- MySQL、数据库、表、字段均可设置编码
-- 数据编码与客户端编码不需一致
SHOW VARIABLES LIKE 'character_set_%'    -- 查看所有字符集编码项
    character_set_client      客户端向服务器发送数据时使用的编码
    character_set_results     服务器端将结果返回给客户端所使用的编码
    character_set_connection  连接层编码
SET 变量名 = 变量值
    SET character_set_client = gbk;
    SET character_set_results = gbk;
    SET character_set_connection = gbk;
SET NAMES GBK;  -- 相当于完成以上三个设置
-- 校对集
    校对集用以排序
    SHOW CHARACTER SET [LIKE 'pattern']/SHOW CHARSET [LIKE 'pattern']  查看所有字符集
    SHOW COLLATION [LIKE 'pattern']  查看所有校对集
    CHARSET 字符集编码      设置字符集编码
    COLLATE 校对集编码      设置校对集编码
    
```

数据类型(列类型)

```

/* 数据类型 (列类型) */ -----
1. 数值类型
-- a. 整型 -----
    类型          字节          范围 (有符号位)
    
```

```

tinyint    1字节    -128 ~ 127    无符号位: 0 ~ 255
smallint   2字节    -32768 ~ 32767
mediumint  3字节    -8388608 ~ 8388607
int        4字节
bigint     8字节
int(M)    M表示总位数

```

- 默认存在符号位, unsigned 属性修改
- 显示宽度, 如果某个数不够定义字段时设置的位数, 则前面以0补填, zerofill 属性修改
例: int(5) 插入一个数'123', 补填后为'00123'
- 在满足要求的情况下, 越小越好。
- 1表示bool值真, 0表示bool值假。MySQL没有布尔类型, 通过整型0和1表示。常用tinyint(1)表示布尔型。

-- b. 浮点型 -----

```

类型          字节    范围
float(单精度)   4字节
double(双精度)  8字节

```

浮点型既支持符号位 unsigned 属性, 也支持显示宽度 zerofill 属性。

不同于整型, 前后均会补填0。

定义浮点型时, 需指定总位数和小数位数。

```
float(M, D)    double(M, D)
```

M表示总位数, D表示小数位数。

M和D的大小会决定浮点数的范围。不同于整型的固定范围。

M既表示总位数 (不包括小数点和正负号), 也表示显示宽度 (所有显示符号均包括)。

支持科学计数法表示。

浮点数表示近似值。

-- c. 定点数 -----

decimal -- 可变长度

```
decimal(M, D)    M也表示总位数, D表示小数位数。
```

保存一个精确的数值, 不会发生数据的改变, 不同于浮点数的四舍五入。

将浮点数转换为字符串来保存, 每9位数字保存为4个字节。

2. 字符串类型

-- a. char, varchar -----

char 定长字符串, 速度快, 但浪费空间

varchar 变长字符串, 速度慢, 但节省空间

M表示能存储的最大长度, 此长度是字符数, 非字节数。

不同的编码, 所占用的空间不同。

char, 最多255个字符, 与编码无关。

varchar, 最多65535字符, 与编码有关。

一条有效记录最大不能超过65535个字节。

utf8 最大为21844个字符, gbk 最大为32766个字符, latin1 最大为65532个字符

varchar 是变长的, 需要利用存储空间保存 varchar 的长度, 如果数据小于255个字节, 则采用一个字节来保存长度, 反之需要两个字节来保存。

varchar 的最大有效长度由最大行大小和使用的字符集确定。

最大有效长度是65532字节, 因为在varchar存字符串时, 第一个字节是空的, 不存在任何数据, 然后还需两个字节来存放字符串的长度, 所以有效长度是65535-1-2=65532字节。

例: 若一个表定义为 CREATE TABLE tb(c1 int, c2 char(30), c3 varchar(N))

charset=utf8; 问N的最大值是多少? 答: (65535-1-2-4-30*3)/3

-- b. blob, text -----

blob 二进制字符串 (字节字符串)

```
tinyblob, blob, mediumblob, longblob
```

text 非二进制字符串 (字符字符串)

```
tinytext, text, mediumtext, longtext
```

text 在定义时, 不需要定义长度, 也不会计算总长度。

text 类型在定义时, 不可给default值

```
-- c. binary, varbinary -----
    类似于char和varchar, 用于保存二进制字符串, 也就是保存字节字符串而非字符串。
    char, varchar, text 对应 binary, varbinary, blob.
```

3. 日期时间类型

一般用整型保存时间戳, 因为PHP可以很方便的将时间戳进行格式化。

	字节数	范围/描述	范围/描述
datetime	8字节	日期及时间	1000-01-01 00:00:00 到 9999-12-31 23:59:59
date	3字节	日期	1000-01-01 到 9999-12-31
timestamp	4字节	时间戳	19700101000000 到 2038-01-19 03:14:07
time	3字节	时间	-838:59:59 到 838:59:59
year	1字节	年份	1901 - 2155

```
datetime    YYYY-MM-DD hh:mm:ss
timestamp   YY-MM-DD hh:mm:ss
            YYYYMMDDhhmmss
            YYMMDDhhmmss
            YYYYMMDDhhmmss
            YYMMDDhhmmss
date        YYYY-MM-DD
            YY-MM-DD
            YYYYMMDD
            YYMMDD
            YYYYMMDD
            YYMMDD
time        hh:mm:ss
            hhmmss
            hhmmss
year        YYYY
            YY
            YYYY
            YY
```

4. 枚举和集合

-- 枚举(enum) -----

```
enum(val1, val2, val3...)
```

在已知的值中进行单选。最大数量为65535。

枚举值在保存时, 以2个字节的整型(smallint)保存。每个枚举值, 按保存的位置顺序, 从1开始逐一递增。

表现为字符串类型, 存储却是整型。

NULL值的索引是NULL。

空字符串错误值的索引值是0。

-- 集合 (set) -----

```
set(val1, val2, val3...)
```

```
create table tab ( gender set('男', '女', '无') );
insert into tab values ('男, 女');
```

最多可以有64个不同的成员。以bigint存储, 共8个字节。采取位运算的形式。

当创建表时, SET成员值的尾部空格将自动被删除。

列属性(列约束)

```
/* 列属性 (列约束) */ -----
```

1. PRIMARY 主键

- 能唯一标识记录的字段, 可以作为主键。
- 一个表只能有一个主键。

- 主键具有唯一性。
- 声明字段时，用 primary key 标识。
也可以在字段列表之后声明

例：create table tab (id int, stu varchar(10), primary key (id));

- 主键字段的值不能为null。
- 主键可以由多个字段共同组成。此时需要在字段列表后声明的方法。

例：create table tab (id int, stu varchar(10), age int, primary key (stu, age));

2. UNIQUE 唯一索引（唯一约束）

使得某字段的值也不能重复。

3. NULL 约束

null不是数据类型，是列的一个属性。

表示当前列是否可以不为null，表示什么都没有。

null，允许为空。默认。

not null，不允许为空。

insert into tab values (null, 'val');

-- 此时表示将第一个字段的值设为null，取决于该字段是否允许为null

4. DEFAULT 默认值属性

当前字段的默认值。

insert into tab values (default, 'val'); -- 此时表示强制使用默认值。

create table tab (add_time timestamp default current_timestamp);

-- 表示将当前时间的时间戳设为默认值。

current_date, current_time

5. AUTO_INCREMENT 自动增长约束

自动增长必须为索引（主键或unique）

只能存在一个字段为自动增长。

默认为1开始自动增长。可以通过表属性 auto_increment = x进行设置，或 alter table tbl auto_increment = x;

6. COMMENT 注释

例：create table tab (id int) comment '注释内容';

7. FOREIGN KEY 外键约束

用于限制主表与从表数据完整性。

alter table t1 add constraint `t1_t2_fk` foreign key (t1_id) references t2(id);

-- 将表t1的t1_id外键关联到表t2的id字段。

-- 每个外键都有一个名字，可以通过 constraint 指定

存在外键的表，称之为从表（子表），外键指向的表，称之为表（父表）。

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（从表）中的数据。

MySQL中，可以对InnoDB引擎使用外键约束：

语法：

foreign key (外键字段) references 主表名 (关联字段) [主表记录删除时的动作] [主表记录更新时的动作]

此时需要检测一个从表的外键需要约束为主表的已存在的值。外键在没有关联的情况下，可以设置为null。前提是外键列，没有not null。

可以不指定主表记录更改或更新时的动作，那么此时主表的操作被拒绝。

如果指定了 on update 或 on delete：在删除或更新时，有如下几个操作可以选择：

1. cascade, 级联操作。主表数据被更新（主键值更新），从表也被更新（外键值更新）。主表记录被删除，从表相关记录也被删除。

2. set null, 设置为null。主表数据被更新（主键值更新），从表的外键被设置为null。主表记录被删除，从表相关记录外键被设置成null。但注意，要求该外键列，没有not null属性约束。

3. restrict, 拒绝父表删除和更新。

注意，外键只被InnoDB存储引擎所支持。其他引擎是不支持的。

建表规范

```

/* 建表规范 */ -----
-- Normal Format, NF
  - 每个表保存一个实体信息
  - 每个具有一个ID字段作为主键
  - ID主键 + 原子表
-- 1NF, 第一范式
  字段不能再分, 就满足第一范式。
-- 2NF, 第二范式
  满足第一范式的前提下, 不能出现部分依赖。
  消除复合主键就可以避免部分依赖。增加单列关键字。
-- 3NF, 第三范式
  满足第二范式的前提下, 不能出现传递依赖。
  某个字段依赖于主键, 而有其他字段依赖于该字段。这就是传递依赖。
  将一个实体信息的数据放在一个表内实现。

```

SELECT

```

/* SELECT */ -----
SELECT [ALL|DISTINCT] select_expr FROM -> WHERE -> GROUP BY [合计函数] -> HAVING -
> ORDER BY -> LIMIT
a. select_expr
  -- 可以用 * 表示所有字段。
    select * from tb;
  -- 可以使用表达式 (计算公式、函数调用、字段也是个表达式)
    select stu, 29+25, now() from tb;
  -- 可以为每个列使用别名。适用于简化列标识, 避免多个列标识符重复。
    - 使用 as 关键字, 也可省略 as.
    select stu+10 as add10 from tb;
b. FROM 子句
  用于标识查询来源。
  -- 可以为表起别名。使用as关键字。
    SELECT * FROM tb1 AS tt, tb2 AS bb;
  -- from子句后, 可以同时出现多个表。
    -- 多个表会横向叠加到一起, 而数据会形成一个笛卡尔积。
    SELECT * FROM tb1, tb2;
  -- 向优化符提示如何选择索引
    USE INDEX、IGNORE INDEX、FORCE INDEX
    SELECT * FROM table1 USE INDEX (key1,key2) WHERE key1=1 AND key2=2 AND
key3=3;
    SELECT * FROM table1 IGNORE INDEX (key3) WHERE key1=1 AND key2=2 AND
key3=3;
c. WHERE 子句
  -- 从from获得的数据源中进行筛选。
  -- 整型1表示真, 0表示假。
  -- 表达式由运算符和运算数组成。
    -- 运算数: 变量 (字段)、值、函数返回值
    -- 运算符:

```



```

=, <=>, <>, !=, <=, <, >=, >, !, &&, ||,
in (not) null, (not) like, (not) in, (not) between and, is (not), and,
or, not, xor
    is/is not 加上ture/false/unknown, 检验某个值的真假
    <=>与<>功能相同, <=>可用于null比较
d. GROUP BY 子句, 分组子句
    GROUP BY 字段/别名 [排序方式]
    分组后会进行排序。升序: ASC, 降序: DESC
    以下[合计函数]需配合 GROUP BY 使用:
    count 返回不同的非NULL值数目 count(*), count(字段)
    sum 求和
    max 求最大值
    min 求最小值
    avg 求平均值
    group_concat 返回带有来自一个组的连接的非NULL值的字符串结果。组内字符串连接。
e. HAVING 子句, 条件子句
    与 where 功能、用法相同, 执行时机不同。
    where 在开始时执行检测数据, 对原数据进行过滤。
    having 对筛选出的结果再次进行过滤。
    having 字段必须是查询出来的, where 字段必须是数据表存在的。
    where 不可以使用字段的别名, having 可以。因为执行WHERE代码时, 可能尚未确定列值。
    where 不可以使用合计函数。一般需用合计函数才会用 having
    SQL标准要求HAVING必须引用GROUP BY子句中的列或用于合计函数中的列。
f. ORDER BY 子句, 排序子句
    order by 排序字段/别名 排序方式 [,排序字段/别名 排序方式]...
    升序: ASC, 降序: DESC
    支持多个字段的排序。
g. LIMIT 子句, 限制结果数量子句
    仅对处理好的结果进行数量限制。将处理好的结果的看作是一个集合, 按照记录出现的顺序, 索引从0开始。
    limit 起始位置, 获取条数
    省略第一个参数, 表示从索引0开始。limit 获取条数
h. DISTINCT, ALL 选项
    distinct 去除重复记录
    默认为 all, 全部记录

```

UNION

```

/* UNION */ -----
    将多个select查询的结果组合成一个结果集合。
    SELECT ... UNION [ALL|DISTINCT] SELECT ...
    默认 DISTINCT 方式, 即所有返回的行都是唯一的
    建议, 对每个SELECT查询加上小括号包裹。
    ORDER BY 排序时, 需加上 LIMIT 进行结合。
    需要各select查询的字段数量一样。
    每个select查询的字段列表(数量、类型)应一致, 因为结果中的字段名以第一条select语句为准。

```

子查询

```

/* 子查询 */ -----
- 子查询需用括号包裹。
-- from型
    from后要求是一个表，必须给子查询结果取个别名。
    - 简化每个查询内的条件。
    - from型需将结果生成一个临时表格，可用以原表的锁定的释放。
    - 子查询返回一个表，表型子查询。
    select * from (select * from tb where id>0) as subfrom where id>1;
-- where型
    - 子查询返回一个值，标量子查询。
    - 不需要给子查询取别名。
    - where子查询内的表，不能直接用以更新。
    select * from tb where money = (select max(money) from tb);
-- 列子查询
    如果子查询结果返回的是一列。
    使用 in 或 not in 完成查询
    exists 和 not exists 条件
        如果子查询返回数据，则返回1或0。常用于判断条件。
        select column1 from t1 where exists (select * from t2);
-- 行子查询
    查询条件是一个行。
    select * from t1 where (id, gender) in (select id, gender from t2);
    行构造符: (col1, col2, ...) 或 ROW(col1, col2, ...)
    行构造符通常用于与对能返回两个或两个以上列的子查询进行比较。
-- 特殊运算符
!= all()    相当于 not in
= some()    相当于 in。any 是 some 的别名
!= some()   不等于 not in, 不等于其中某一个。
all, some 可以配合其他运算符一起使用。
    
```

连接查询(join)

```

/* 连接查询(join) */ -----
    将多个表的字段进行连接，可以指定连接条件。
-- 内连接(inner join)
    - 默认就是内连接，可省略inner。
    - 只有数据存在时才能发送连接。即连接结果不能出现空行。
    on 表示连接条件。其条件表达式与where类似。也可以省略条件（表示条件永远为真）
    也可用where表示连接条件。
    还有 using，但需字段名相同。 using(字段名)
-- 交叉连接 cross join
    即，没有条件的内连接。
    select * from tb1 cross join tb2;
-- 外连接(outer join)
    - 如果数据不存在，也会出现在连接结果中。
-- 左外连接 left join
    如果数据不存在，左表记录会出现，而右表为null填充
-- 右外连接 right join
    如果数据不存在，右表记录会出现，而左表为null填充
-- 自然连接(natural join)
    
```

```

    自动判断连接条件完成连接。
    相当于省略了using, 会自动查找相同字段名。
    natural join
    natural left join
    natural right join
select info.id, info.name, info.stu_num, extra_info.hobby, extra_info.sex from
info, extra_info where info.stu_num = extra_info.stu_id;
    
```

TRUNCATE

```

/* TRUNCATE */ -----
TRUNCATE [TABLE] tbl_name
清空数据
删除重建表
区别：
1, truncate 是删除表再创建, delete 是逐条删除
2, truncate 重置auto_increment的值。而delete不会
3, truncate 不知道删除了几条, 而delete知道。
4, 当被用于带分区的表时, truncate 会保留分区
    
```

备份与还原

``mysql /* 备份与还原 */ ----- 备份, 将数据的结构与表内数据保存起来。利用 mysqldump 指令完成。 -- 导出 mysqldump [options] db_name [tables] mysqldump [options] ---database DB1 [DB2 DB3...] mysqldump [options] --all--database 1. 导出一张表 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 -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 -u用户名 -p密码 库名 > 文件名(D:/a.sql) 4. 导出一个库 mysqldump -u用户名 -p密码 --lock-all-tables --database 库名 > 文件名(D:/a.sql) 可以-w携带WHERE条件 -- 导入 1. 在登录mysql的情况下: 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 ``

视图

什么是视图:

视图是一个虚拟表, 其内容由查询定义。同真实的表一样, 视图包含一系列带有名称的列和行数据。但是, 视图并不在数据库中以存储的数据值集形式存在。行和列数据来自由定义视图的查询所引用的表, 并且在引用视图时动态生成。

视图具有表结构文件, 但不存在数据文件。

对其中所引用的基础表来说, 视图的作用类似于筛选。定义视图的筛选可以来自当前或其它数据库的一个或多个表, 或者其它视图。通过视图进行查询没有任何限制, 通过它们进行数据修改时的限制也很少。

视图是存储在数据库中的查询的sql语句, 它主要出于两种原因: 安全原因, 视图可以隐藏一些数据, 如: 社会保险基金表, 可以用视图只显示姓名, 地址, 而不显示社会保险号和工资数等, 另一原因是可使复杂的查询易于理解和使用。

-- 创建视图

```

CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW view_name
[(column_list)] AS select_statement
    - 视图名必须唯一, 同时不能与表重名。
    - 视图可以使用select语句查询到的列名, 也可以自己指定相应的列名。
    
```

- 可以指定视图执行的算法，通过ALGORITHM指定。
- column_list如果存在，则数目必须等于SELECT语句检索的列数
- 查看结构
 - SHOW CREATE VIEW view_name
- 删除视图
 - 删除视图后，数据依然存在。
 - 可同时删除多个视图。
 - DROP VIEW [IF EXISTS] view_name ...
- 修改视图结构
 - 一般不修改视图，因为不是所有的更新视图都会映射到表上。
 - ALTER VIEW view_name [(column_list)] AS select_statement
- 视图作用
 1. 简化业务逻辑
 2. 对客户端隐藏真实的表结构
- 视图算法(ALGORITHM)
 - MERGE 合并
 - 将视图的查询语句，与外部查询需要先合并再执行！
 - TEMPTABLE 临时表
 - 将视图执行完毕后，形成临时表，再做外层查询！
 - UNDEFINED 未定义(默认)，指的是MySQL自主去选择相应的算法。

事务(transaction)

事务是指逻辑上的一组操作，组成这组操作的各个单元，要不全成功要不全失败。

- 支持连续SQL的集体成功或集体撤销。
- 事务是数据库在数据完整性方面的一个功能。
- 需要利用 InnoDB 或 BDB 存储引擎，对自动提交的特性支持完成。
- InnoDB被称为事务安全型引擎。
- 事务开启
 - START TRANSACTION; 或者 BEGIN;
 - 开启事务后，所有被执行的SQL语句均被认作当前事务内的SQL语句。
- 事务提交
 - COMMIT;
- 事务回滚
 - ROLLBACK;
 - 如果部分操作发生问题，映射到事务开启前。
- 事务的特性
 1. 原子性 (Atomicity)
 - 事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
 2. 一致性 (Consistency)
 - 事务前后数据的完整性必须保持一致。
 - 事务开始和结束时，外部数据一致
 - 在整个事务过程中，操作是连续的
 3. 隔离性 (Isolation)
 - 多个用户并发访问数据库时，一个用户的事务不能被其它用户的事物所干扰，多个并发事务之间的数据要相互隔离。
 4. 持久性 (Durability)
 - 一个事务一旦被提交，它对数据库中的数据改变就是永久性的。
- 事务的实现
 1. 要求是事务支持的表类型
 2. 执行一组相关的操作前开启事务

```

3. 整组操作完成后，都成功，则提交；如果存在失败，选择回滚，则会回到事务开始的备份点。
-- 事务的原理
    利用InnoDB的自动提交(autocommit)特性完成。
    普通的MySQL执行语句后，当前的数据提交操作均可被其他客户端可见。
    而事务是暂时关闭“自动提交”机制，需要commit提交持久化数据操作。
-- 注意
    1. 数据定义语言（DDL）语句不能被回滚，比如创建或取消数据库的语句，和创建、取消或更改表或存储的子程序的语句。
    2. 事务不能被嵌套
-- 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
-- InnoDB自动提交特性设置
    SET autocommit = 0|1;    0表示关闭自动提交，1表示开启自动提交。
    - 如果关闭了，那普通操作的结果对其他客户端也不可见，需要commit提交后才能持久化数据操作。
    - 也可以关闭自动提交来开启事务。但与START TRANSACTION不同的是，
        SET autocommit是永久改变服务器的设置，直到下次再次修改该设置。（针对当前连接）
        而START TRANSACTION记录开启前的状态，而一旦事务提交或回滚后就需要再次开启事务。（针对当前事务）

```

锁表

``mysql /* 锁表 */ 表锁定只用于防止其它客户端进行不正当地读取和写入 MyISAM 支持表锁，InnoDB 支持行锁 -- 锁定 LOCK TABLES tbl_name [AS alias] -- 解锁 UNLOCK TABLES `` ### 触发器

```

/* 触发器 */ -----
    触发程序是与表有关的命名数据库对象，当该表出现特定事件时，将激活该对象
    监听：记录的增加、修改、删除。
-- 创建触发器
CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH ROW
trigger_stmt
    参数：
        trigger_time是触发程序的动作时间。它可以是 before 或 after，以指明触发程序是在激活
        它的语句之前或之后触发。
        trigger_event指明了激活触发程序的语句的类型
            INSERT：将新行插入表时激活触发程序
            UPDATE：更改某一行时激活触发程序
            DELETE：从表中删除某一行时激活触发程序
        tbl_name：监听的表，必须是永久性的表，不能将触发程序与TEMPORARY表或视图关联起来。
        trigger_stmt：当触发程序激活时执行的语句。执行多个语句，可使用BEGIN...END复合语句结
        构
-- 删除
DROP TRIGGER [schema_name.]trigger_name
    可以使用old和new代替旧的和新的数据
        更新操作，更新前是old，更新后是new。
        删除操作，只有old。
        增加操作，只有new。
-- 注意

```

```

1. 对于具有相同触发程序动作时间和事件的给定表，不能有两个触发程序。
-- 字符连接函数
concat(str1,str2,...])
concat_ws(separator,str1,str2,...)
-- 分支语句
if 条件 then
    执行语句
elseif 条件 then
    执行语句
else
    执行语句
end if;
-- 修改最外层语句结束符
delimiter 自定义结束符号
    SQL语句
自定义结束符号
delimiter ;      -- 修改回原来的分号
-- 语句块包裹
begin
    语句块
end
-- 特殊的执行
1. 只要添加记录，就会触发程序。
2. Insert into on duplicate key update 语法会触发：
    如果没有重复记录，会触发 before insert, after insert;
    如果有重复记录并更新，会触发 before insert, before update, after update;
    如果有重复记录但是没有发生更新，则触发 before insert, before update
3. Replace 语法 如果有记录，则执行 before insert, before delete, after delete, after insert

```

SQL编程

```

/* SQL编程 */ -----
--// 局部变量 -----
-- 变量声明
    declare var_name[,...] type [default value]
    这个语句被用来声明局部变量。要给变量提供一个默认值，请包含一个default子句。值可以被指定为一个表达式，不需要为一个常数。如果没有default子句，初始值为null。
-- 赋值
    使用 set 和 select into 语句为变量赋值。
    - 注意：在函数内是可以使用全局变量（用户自定义的变量）
--// 全局变量 -----
-- 定义、赋值
set 语句可以定义并为变量赋值。
set @var = value;
也可以使用select into语句为变量初始化并赋值。这样要求select语句只能返回一行，但是可以是多个字段，就意味着同时为多个变量进行赋值，变量的数量需要与查询的列数一致。
还可以把赋值语句看作一个表达式，通过select执行完成。此时为了避免=被当作关系运算符看待，使用:=代替。（set语句可以使用= 和 :=）。
select @var:=20;
select @v1:=id, @v2=name from t1 limit 1;

```

```

select * from tbl_name where @var:=30;
select into 可以将表中查询获得的数据赋给变量。
    -| select max(height) into @max_height from tb;
-- 自定义变量名
为了避免select语句中，用户自定义的变量与系统标识符（通常是字段名）冲突，用户自定义变量在
变量名前使用@作为开始符号。
@var=10;
    - 变量被定义后，在整个会话周期都有效（登录到退出）
--// 控制结构 -----
-- if语句
if search_condition then
    statement_list
[elseif search_condition then
    statement_list]
...
[else
    statement_list]
end if;
-- case语句
CASE value WHEN [compare-value] THEN result
[WHEN [compare-value] THEN result ...]
[ELSE result]
END
-- while循环
[begin_label:] while search_condition do
    statement_list
end while [end_label];
- 如果需要在循环内提前终止 while循环，则需要使用标签；标签需要成对出现。
    -- 退出循环
        退出整个循环 leave
        退出当前循环 iterate
        通过退出的标签决定退出哪个循环
--// 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

-- 流程函数
case when [condition] then result [when [condition] then result ...] [else result]
end    多分支
if(expr1,expr2,expr3)  双分支。

-- 聚合函数
count()
sum();
max();
min();
avg();
group_concat()
-- 其他常用函数
md5();
default();
--// 存储函数, 自定义函数 -----
-- 新建
    CREATE FUNCTION function_name (参数列表) RETURNS 返回值类型
        函数体
    - 函数名, 应该合法的标识符, 并且不应该与已有的关键字冲突。
    - 一个函数应该属于某个数据库, 可以使用db_name.funciton_name的形式执行当前函数所属数据库, 否则为当前数据库。
    - 参数部分, 由"参数名"和"参数类型"组成。多个参数用逗号隔开。
    - 函数体由多条可用的mysql语句, 流程控制, 变量声明等语句构成。
    - 多条语句应该使用 begin...end 语句块包含。
    - 一定要有 return 返回值语句。
-- 删除
    DROP FUNCTION [IF EXISTS] function_name;
-- 查看
    SHOW FUNCTION STATUS LIKE 'partten'
    SHOW CREATE FUNCTION function_name;
-- 修改
    ALTER FUNCTION function_name 函数选项
--// 存储过程, 自定义功能 -----
-- 定义
存储存储过程 是一段代码(过程), 存储在数据库中的sql组成。
一个存储过程通常用于完成一段业务逻辑, 例如报名, 交班费, 订单入库等。
```


而一个函数通常专注与某个功能，视为其他程序服务的，需要在其他语句中调用函数才可以，而存储过程不能被其他调用，是自己执行 通过call执行。

-- 创建

CREATE PROCEDURE sp_name (参数列表)

过程体

参数列表：不同于函数的参数列表，需要指明参数类型

IN, 表示输入型

OUT, 表示输出型

INOUT, 表示混合型

注意，没有返回值。

存储过程

/* 存储过程 */ -----

存储过程是一段可执行性代码的集合。相比函数，更偏向于业务逻辑。

调用：CALL 过程名

-- 注意

- 没有返回值。

- 只能单独调用，不可夹杂在其他语句中

-- 参数

IN|OUT|INOUT 参数名 数据类型

IN 输入：在调用过程中，将数据输入到过程体内部的参数

OUT 输出：在调用过程中，将过程体处理完的结果返回到客户端

INOUT 输入输出：既可输入，也可输出

-- 语法

CREATE PROCEDURE 过程名 (参数列表)

BEGIN

过程体

END

用户和权限管理

/* 用户和权限管理 */ -----

-- root密码重置

1. 停止MySQL服务

2. [Linux] /usr/local/mysql/bin/safe_mysqld --skip-grant-tables &
[Windows] mysqld --skip-grant-tables

3. use mysql;

4. UPDATE `user` SET PASSWORD=PASSWORD("密码") WHERE `user` = "root";

5. FLUSH PRIVILEGES;

用户信息表：mysql.user

-- 刷新权限

FLUSH PRIVILEGES;

-- 增加用户

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 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 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
    GRANT ALL PRIVILEGES ON `pms`.* TO 'pms'@'%' IDENTIFIED BY 'pms0817';
-- 查看权限
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
-- 权限层级
-- 要使用GRANT或REVOKE，您必须拥有GRANT OPTION权限，并且您必须用于您正在授予或撤销的权
限。
全局层级：全局权限适用于一个给定服务器中的所有数据库，mysql.user
    GRANT ALL ON *.*和 REVOKE ALL ON *.*只授予和撤销全局权限。
数据库层级：数据库权限适用于一个给定数据库中的所有目标，mysql.db，mysql.host
    GRANT ALL ON db_name.*和REVOKE ALL ON db_name.*只授予和撤销数据库权限。
表层级：表权限适用于一个给定表中的所有列，mysql.tables_priv
    GRANT ALL ON db_name.tbl_name和REVOKE ALL ON db_name.tbl_name只授予和撤销表权
限。
列层级：列权限适用于一个给定表中的单一列，mysql.columns_priv
    当使用REVOKE时，您必须指定与被授权列相同的列。
-- 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

```
/* 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] ...
```

杂项

```
/* 杂项 */ -----
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
7. 清除已有语句：\c
```