MySQL 扩展内容和变量定义

一、 衍生列（generated column）

在基础列的上，通过一定的运算转换而定义生成的列，即为衍生列。

基本语法：

```
CREATE TABLE <t_name> (
  col <col_type>,
  g_col <col_type> GENERATED ALWAYS AS (<computation on col>) [stored | virtual]
[NOT NULL] [PRIMARY KEY]
    );
```

- 有两种类型的衍生列： stored 和 virtual ，默认为 virtual
- 生成列的构建可包含字面量、确定性内建函数和操作符、基本列、创建于其前的生成列。
- 以下不允许出现在生成列定义中
  - 不确性内建函数如 current_date(), current_timestamp(),connection_id() 等
  - 用户自定义存储过程，系统、用户、局部变量等
  - 子查询
  - 在其后定义的生成列
  - auto_increment

- 使用 `create table ... like` 可保留原表中的生成列定义；使用 `create table ... select` 则不能。
- 生成列的使用场景：
  - 简化和整合查询。复杂条件可通过生成列表示，查询可直接指向该列，从而保证查询条件的一致性。
  - 作为复杂条件或运算的物化存储，提前缓存，从而减少查询时间
  - 查询优化器可识别出使用了生成列定义的查询，从而可利用创建在生成列上的索引，即便该查询未直接使用该列。
- 生成列的使用限制
  - 生成列不能作为被参照列。
  - 参照列上的外键约束 `on update` 选项不能设为 `cascade, set_null,set default` ， `on delete` 选项不能设为 `set null,set default`
  - 触发器中不能用 `new.col_name` 和 `old.col_name` 指代生成列

```
drop table if exists person;

CREATE TABLE person(
  id int primary key,
  first_name varchar(20) not null,
  last_name varchar(20) not null,
  `name` varchar(50) generated always as (concat(first_name, ' ', last_name)) stored,
  index idx_name (`name`));

show create table person;

delete from person;
insert into person(id, first_name, last_name)
values (1, 'Mike', 'James');

select * from person;
```

二、变量定义与运算符

```
use purchase;
```

1. MySQL 编程中涉及的常量

- 字符串常量

```
-- MySQL推荐使用单引号表示字符串
select 'I\'m a \teacher' as col1, "you're a stude\nt" as col2;
```

- 数值常量

```
select 123 as number;
```

- 日期时间常量

```
select now() as now;
select year(now()), month(now()), day(now()),
    hour(now()), minute(now()), second(now()) as t_day;
```

- 布尔值

```
select true, false;
```

- 二进制

```
select 0b111001, b'111001';

select CONV('10', 10, 2);   -- 10进制转化为2进制
select bin(8);  -- 把10进制数转化为2进制
```

- 十六进制

```
select x'41', x'4D7953514C';
select 0x41, 0x4d7953514c;

select CONV('42', 10, 16);   -- 10进制转化为16进制
select hex(42), hex(16);  -- 把10进制数转化为16进制
```

- null

```
select null, null > 1;  -- null与任何值进行比较的结果为null
```

2. 用户自定义变量

- 用户会话变量
- 方法1： `set`

```
set @user_name = '张三'; -- 变量数据类型由等号右边表达式的计算结果决定
select @user_name;

set @user_name = b'11', @age = 18;  -- 同时定义多个变量
select @user_name, @age;

set @age = @age * 3 + 1;  -- 对变量进行更新
select @age;
```

- 方法2： `select`

> 有两种语法格式：
> 第一种： `select @user_variable1 := expression1 [, @user_variable2 := expression2, ...];`
> 第二种： `select experession1 into @user_variable1, experession2 into @user_variable2, ...;`
> 注意和 `set` 定义变量的区别：用 `set` 定义变量时，直接使用 `=` ；用 `select` 定义变量时，使

> 用':=`

- 使用 变量名称 := 变量值

示例1: 使用 变量名称 := 变量值, 定义用户变量a，将其赋值为'b'

```
select @a := 'b';  -- 用户定义变量a，赋值为'b'
select @a;
-- 注意：:= 和 = 的区别
select @a = 'a';  --用户定义变量a与'a'进行等值比较的结果，如果a之前定义了，则返回1或0；如果a之前未被定义，则返回null
select @a;
```

- 使用 select 变量值 into @变量名称;

示例2：使用 select 变量值into @变量名称，定义用户变量user_name，赋值为'张三'

```
select '张三',19 into @user_name, @age;
select @user_name, @age;
```

- 使用用户会话变量保存sql查询结果

示例3：将product表中的记录数赋值给用户会话变量@product_count

```
-- 方法1：set @变量名 = select语句
set @product_count = (select count(*) from Product);
select @product_count as 产品数量;

-- 方法2：select @变量名 := select语句
select @product_count2 := (select count(*) from Product);
select @product_count2 as 产品数量;

-- 方法3：select @变量名 := 聚合函数 from 表
select @product_count3 := count(*) from Product;
select @product_count3 as 产品数量;

-- 方法4：select 聚合函数 into @变量名 from 表
select count(*) into @product_count4 from Product;
select @product_count4 as 产品数量;

-- 方法5：select 聚合函数 from 表 into @变量名
select count(*) from Product into @product_count5;
select @product_count5 as 产品数量;
```

示例4: 通过定义用户变量查询product表中的特定行

```
set @product_code = '1101001';
select * from Product where product_code = @product_code;

select * from Product where product_code = '1101001';
```

示例5：通过自定义变量查询sort_name为纸张的所有产品信息

```
select * from sort;
select sort_id into @v_sortid from sort where sort_name = '纸张';
select * from product where sort_id = @v_sortid;

select product.* from product join sort on product.sort_id = sort.sort_id
where sort_name = '纸张';
```

示例6：从 `product` 表所有记录奇数行构成的集合。

> 低版本的 `MySQL` 没有行号函数，可通过定义用户会话变量来模拟实现。

```
SELECT row_num, product_id, product_name, price
FROM (SELECT @row_num := @row_num + 1 AS row_num, product_id, product_name, price
  FROM product, (SELECT @row_num := 0) AS r) AS b_product
WHERE row_num % 2 != 0;
```

3. 运算符

- 算术

```
set @num=15;
select @num + 2, @num - 2, @num * 3, @num / 3;
select @num % 2;
select @num + null, @num - null, @num * null, @num / null, @num % null;都是null
select @num / 0, @num % 0; 都是null
select '2012-12-21' + interval '50' day;
select '2012-12-21' - interval '50' day;  -- 2012年12月21日减去50天

select pow(3, 10);
```

- 比较

```
 select 'ab '='ab', ' ab'='ab', 'b'>'a';  0 0 1
 select '1' = 1, '1' > 0, 1 > '0', '1' > '0'; 1 1 1 1
 select null = null, null < null, null is null, null is not null;  null null 1 0
 select null > 1, null < 1, null = 1;   都null
 select null != null, null <> null;   都null
 select 'b' between 'a' and 'c';    1
 select 10 not between 5 and 9;
 select 1 in (1, 2, 'a');
 select 1 not in (1, 2, 'a');
 select isnull(null); -- 函数
 select 'stud' like 'stud', 'stud' like 'stu_', 'stud' like 'st%';
 select 'student' regexp '^s', 'student' regexp '[a-z]'; -- 字符串是否匹配某一模式
```

在大多数数据库系统中，当进行比较操作时，数据库会尝试将操作数转换为相同的数据类型，以便进行比较。在这种情况下，数据库会将字符串'1'和'0'转换为整数1和0，因为整数比较通常是按照数值进行的。

- 逻辑

```sql
select 1 and 2, 2 and 0, 2 and true, 0 or true, not 2, not false;
select 1 && 2, 2 && 0, 2 && true, 0 || true, ! 2, ! false;
select null && 2, null || 2, ! null;

select 1 and 0 or 1; -- 运算优先级，先运算and，后运算or
select 0 or 0 and 1;
select not 1 or 0 and 1;
```

三、PREPARED STATEMENT预处理语句

MySQL提供了服务器端的预处理语句，有以下优点：

- 预先在服务器端一次定义好语句，产生优化好的执行计划，后面使用的时候只需传递相关参数即可，因此减少了网络传输负担
- 防止SQL注入攻击，预处理语句可传输非字符串变量。

定义：`PREPARE <statement_name> FROM '<SQL>'`

> <SQL>中的变量用?替代，以下SQL可以定义在预处理语句的sql字符串中:
>
> - ALTER TABLE
> - ALTER USER
> - INSERT
> - SELECT
> - UPDATE
> - DELETE
> - TRUNCATE
> - SET
> - RENAME TABLE
> - {CREATE | RENAME | DROP} DATABASE
> - {CREATE | DROP} TABLE
> - {CREATE | RENAME | DROP} USER
> - {CREATE | DROP} VIEW
> - SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
> - ...

执行：`EXECUTE <statement_name> USING @v1, @v2,....`

解绑：`DEALLOCATE  PREPARE <statement_name>`

示例7：定义预处理语句“选取价格大于某一值的所有记录”

```
set @tb_name = 'product'
set @sql = concat("select * from ", @tb_name, ' limit 10')

prepare stmt_select from "select product_id, product_place, price from product
where price > ? order by price";

set @n = 100;
execute stmt_select using @n;

deallocate prepare stmt_select;
```

> 注意，`prepare statement`是用户会话级别定义的，即使如果未解绑(`DEALLOCATE`)一个`prepare statement`，则在用户中断会话时将自动解绑。

示例8：定义预处理语句“插入产品编号、名称和价格至产品表”

```
prepare stmt_insert from "insert into product set product_id=?, product_name=?,
price=?";

set @pid=9999, @pname='矿泉水', @price=2;
execute stmt_insert using @pid, @pname, @price;

deallocate prepare stmt_insert;
```

四、`REPLACE`：插入或更新

基本语法：

```
REPLACE INTO tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...

REPLACE INTO tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list

REPLACE INTO tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

```

```
assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

功能：插入数据时，如果目标表中已存在相同的**主键或有 `unique` 约束的属性值**，则以此作为条件更新剩余属性值；如果不存在，则执行插入操作。如果目标表中没有定义主键或者唯一性约束字段，则 `replace into` 等价于 `insert into` 。

```
CREATE TABLE p (p_id char(5) primary key,
                p_name varchar(50) unique,
                price decimal(7, 2) not null default 0);
replace into p
values(1, 'a', 1.2), (2, 'b', 5.0); -- 插入

replace into p (p_id, p_name, price)
values(1, 'a', 3); -- 更新p_id值为1, p_name为'a'的行的price值为3

replace into p (p_id, p_name, price)
values(3, 'c', 3.0); -- 相当于insert into

replace into p (p_id, p_name, price)
values(3, 'f', 3), (4, 'e', 8); -- 相当于update和insert into

-- 小心以下更新
replace into p (p_id, p_name, price)
values(3, 'e', 6); -- 更新p_id值为3的行(3, 'f', 3) -> (3, 'e', 6)；更新p_name为'e'的行(4, 'e', 8) -> (3, 'e', 6).
```

五、`TEMPORARY TABLE` 临时表

如果需要保存一些查询的中间结果，可以使用临时表 `temporary table` 。用户在某一次会话（session）中创建的临时表只在该次会话可见，会话结束时临时表也将自动删除。因此，两个不同的会话可以在同一个数据库中创建相同名称的临时表。注意，临时表的名称可以与正式表相同，此后数据操作涉及该表名时将优先选择临时表。注意，临时表上不能创建索引。

```
USE purchase;

CREATE TEMPORARY TABLE temp_product (product_id int primary key,
                                     product_name varchar(50),
                                     price decimal(7, 2));

SHOW CREATE TABLE temp_product;
SHOW TABLES;
DROP TEMPORARY TABLE temp_product;

CREATE TEMPORARY TABLE product AS SELECT * FROM product LIMIT 10;
SHOW CREATE TABLE product;
```

练习

1. 将十进制数1239分别转换为二进制和十六进制数
2. 使用示例3中的5种方法，分别定义用户变量 `count_member1, count_member2, count_member3, count_member4, count_member5` ，保存 `member` 表中的记录数
3. 定义表 `budget` ，包含 `项目编号, 项目名称, 交通费, 会议费, 打印费,交通费` 等基础属性，以及 `总预算` virtual衍生属性， `总预算` 为 `交通费, 会议费, 打印费,交通费` 之和。
4. 基于 `sort` 和 `subsort` 定义临时表 `temp_sort_num` ，用于存储根类比编号、名称和对应该类别的子类别数量。