DQL,DML，DDL的区别
一、基础
二、创建表
三、修改表
四、插入
五、更新
六、删除
七、查询
DISTINCT
LIMIT
八、排序
九、过滤
十、通配符
十一、计算字段
十二、函数
汇总
文本处理
日期和时间处理
数值处理
十三、分组
十四、子查询
十五、连接
内连接
自连接
自然连接
外连接
十六、组合查询
十七、视图
十八、存储过程
十九、游标
二十、触发器
二十一、事务管理
二十二、字符集
二十三、权限管理
参考资料

DQL,DML，DDL的区别

数据查询语言DQL
数据查询语言DQL基本结构是由SELECT子句，FROM子句，WHERE
子句组成的查询块：
SELECT <字段名表>
FROM <表或视图名>
WHERE <查询条件>

DML数据操纵语言：对数据库中数据进行一些简单操作，`insert,delete,update,select`
DDL数据定义语言：对数据库中的某些对象，如`database和table`，进行管
理，如`Create,Alter和Drop.`
区别：

DML操作是可以手动控制事务的开启、提交和回滚的。
DDL操作是隐性提交的，不能rollback

一、基础

模式定义了数据如何存储、存储什么样的数据以及数据如何分解等信息，数据库和表都有模式。

主键的值不允许修改，也不允许复用（不能将已经删除的主键值赋给新数据行的主键）。

SQL（Structured Query Language），标准 SQL 由 ANSI 标准委员会管理，从而称为 ANSI SQL。各个 DBMS 都有自己的实现，如 PL/SQL、Transact-SQL 等。

SQL 语句不区分大小写，但是数据库表名、列名和值是否区分依赖于具体的 DBMS 以及配置。

SQL 支持以下三种注释：

```
# 注释
SELECT *
FROM mytable; -- 注释
/* 注释1
   注释2 */
```

数据库创建与使用：

```
CREATE DATABASE test;
USE test;
```

二、创建表

```
CREATE TABLE mytable (
  # int 类型，不为空，自增
  id INT NOT NULL AUTO_INCREMENT,
  # int 类型，不可为空，默认值为 1，不为空
  col1 INT NOT NULL DEFAULT 1,
  # 变长字符串类型，最长为 45 个字符，可以为空
  col2 VARCHAR(45) NULL,
  # 日期类型，可为空
  col3 DATE NULL,
  # 设置主键为 id
  PRIMARY KEY (`id`));
```

三、修改表

添加列

```
ALTER TABLE mytable
ADD col CHAR(20);
```

删除列

```
ALTER TABLE mytable
DROP COLUMN col;
```

删除表

```
DROP TABLE mytable;
```

四、插入

普通插入

```
INSERT INTO mytable(col1, col2)
VALUES(val1, val2);
```

插入检索出来的数据

```
INSERT INTO mytable1(col1, col2)
SELECT col1, col2
FROM mytable2;
```

将一个表的内容插入到一个新表

```
CREATE TABLE newtable AS
SELECT * FROM mytable;
```

五、更新

```
UPDATE mytable
SET col = val
WHERE id = 1;
```

六、删除

DELETE

直接删除表中的某一行数据，并且同时将该行的删除操作作**为事务记录在日志中**保存以便进行进行回滚操作。所以delete相比较truncate更加占用资源，数
据空间不释放，因为需回滚。**对table和view都能操作**

```
DELETE FROM mytable
WHERE id = 1;
```

TRUNCATE TABLE

一次性地从表中删除所有的数据(释放存储表数据所用的数据页来删除数据)，因此也**不能回滚，不能恢复数据，占用资源更加少，速度更快。**数据空间会释放，这个表和索引所占用的空间会恢复到初始大小。只能操作没有关联视图的table。对于外键约束引用的表，不能使用 truncate table，而应使用不带 where 子句的 delete 语句。

```
TRUNCATE TABLE mytable;
```

使用更新和删除操作时一定要用 WHERE 子句，不然会把整张表的数据都破坏。可以先用 SELECT 语句进行测试，防止错误删除。

七、查询

DISTINCT

相同值只会出现一次。它作用于**所有列，也就是说所有列的值都相同才算相同。**

```
SELECT DISTINCT col1, col2
FROM mytable;
```

LIMIT

限制返回的行数。可以有两个参数，**第一个参数为起始行，从 0 开始；第二个参数为返回的总行数。**

返回前 5 行：

```
SELECT *
FROM mytable
LIMIT 5;
```

```
SELECT *
FROM mytable
LIMIT 0, 5;
```

返回第 3 ~ 5 行：

```
SELECT *
FROM mytable
LIMIT 2, 3;
```

八、排序

- **ASC** ：升序（默认）
- **DESC** ：降序

可以按多个列进行排序，并且为每个列指定不同的排序方式：

```
SELECT *
FROM mytable
ORDER BY col1 DESC, col2 ASC;
```

九、过滤

不进行过滤的数据非常大，导致通过网络传输了多余的数据，从而浪费了网络带宽。因此尽量使用 SQL 语句来过滤不必要的数据，而不是传输所有的数据到客户端中然后由客户端进行过滤。

```
SELECT *
FROM mytable
WHERE col IS NULL;
```

下表显示了 WHERE 子句可用的操作符

操作符	说明
=	等于
<	小于
>	大于
<> !=	不等于
<= !>	小于等于
>= !<	大于等于
BETWEEN （包含前后边界值）	在两个值之间
IS NULL	为 NULL 值

应该注意到，NULL 与 0、空字符串都不同。

AND 和 OR 用于连接多个过滤条件。优先处理 AND，当一个过滤表达式涉及到多个 AND 和 OR 时，可以使用 () 来决定优先级，使得优先级关系更清晰。

IN 操作符用于匹配一组值，其后也可以接一个 SELECT 子句，从而匹配子查询得到的一组值。

NOT 操作符用于否定一个条件。

十、通配符

通配符也是用在过滤语句中，但它只能用于文本字段。

- **%** 匹配 >=0 个任意字符；
- _ 匹配 ==1 个任意字符；
- **[ ]** 可以匹配集合内的字符，例如 [ab] 将匹配字符 a 或者 b。用脱字符 ^ 可以对其进行否定，也就是不匹配集合内的字符。

使用 Like 来进行通配符匹配。

```
SELECT *
FROM mytable
WHERE col LIKE '[^AB]%'; -- 不以 A 和 B 开头的任意文本
```

不要滥用通配符，通配符位于开头处匹配会非常慢。

十一、计算字段

在数据库服务器上完成数据的转换和格式化的工作往往比客户端上快得多，并且转换和格式化后的数据量更少的话可以减少网络通信量。

计算字段通常需要使用 **AS** 来取别名，否则输出的时候字段名为计算表达式。

```
SELECT col1 * col2 AS alias
FROM mytable;
```

CONCAT() 用于连接两个字段。许多数据库会使用空格把一个值填充为列宽，因此连接的结果会出现一些不必要的空格，使用 **TRIM()** 可以去除首尾空格。

```
SELECT CONCAT(TRIM(col1), '(', TRIM(col2), ')') AS concat_col
FROM mytable;
```

十二、函数

各个 DBMS 的函数都是不相同的，因此不可移植，以下主要是 MySQL 的函数。

汇总

函 数	说 明
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列值之和

AVG() 会忽略 NULL 行。

使用 DISTINCT 可以汇总不同的值。

```
SELECT AVG(DISTINCT col1) AS avg_col
FROM mytable;
```

文本处理

函数	说明
LEFT()	左边的字符
RIGHT()	右边的字符
LOWER()	转换为小写字符
UPPER()	转换为大写字符
LTRIM()	去除左边的空格
RTRIM()	去除右边的空格
LENGTH()	长度
SOUNDEX()	转换为语音值

其中， **SOUNDEX()** 可以将一个字符串转换为描述其语音表示的字母数字模式。

```
SELECT *
FROM mytable
WHERE SOUNDEX(col1) = SOUNDEX('apple')
```

日期和时间处理

- 日期格式：YYYY-MM-DD
- 时间格式：HH:MM:SS

函 数	说 明
ADDDATE()	增加一个日期（天、周等）
ADDTIME()	增加一个时间（时、分等）
CURDATE()	返回当前日期
CURTIME()	返回当前时间
DATE()	返回日期时间的日期部分
DATEDIFF()	计算两个日期之差
DATE_ADD()	高度灵活的日期运算函数
DATE_FORMAT()	返回一个格式化的日期或时间串
DAY()	返回一个日期的天数部分
DAYOFWEEK()	对于一个日期，返回对应的星期几
HOUR()	返回一个时间的小时部分
MINUTE()	返回一个时间的分钟部分
MONTH()	返回一个日期的月份部分
NOW()	返回当前日期和时间
SECOND()	返回一个时间的秒部分
TIME()	返回一个日期时间的时间部分
YEAR()	返回一个日期的年份部分

```
mysql> SELECT NOW();
```

```
2018-4-14 20:25:11
```

数值处理

函数	说明
SIN()	正弦
COS()	余弦
TAN()	正切
ABS()	绝对值
SQRT()	平方根
MOD()	余数
EXP()	指数
PI()	圆周率
RAND()	随机数

十三、分组

把具有相同的数据值的行放在同一组中。

可以对同一分组数据使用汇总函数进行处理，例如求分组数据的平均值等。

指定的分组字段除了能按该字段进行分组，也会自动按该字段进行排序。

```
SELECT col, COUNT(*) AS num
FROM mytable
GROUP BY col;
```

GROUP BY 自动按分组字段进行排序，ORDER BY 也可以按汇总字段来进行排序。

```
SELECT col, COUNT(*) AS num
FROM mytable
GROUP BY col
ORDER BY num;
```

WHERE 过滤行，HAVING 过滤分组，行过滤应当先于分组过滤。

```
SELECT col, COUNT(*) AS num
FROM mytable
WHERE col > 2
GROUP BY col
HAVING num >= 2;
```

分组规定：

- GROUP BY 子句出现在 WHERE 子句之后，ORDER BY 子句之前；
- 除了汇总字段外，SELECT 语句中的每一字段都必须在 GROUP BY 子句中给出；
- NULL 的行会单独分为一组；
- 大多数 SQL 实现不支持 GROUP BY 列具有可变长度的数据类型。

十四、子查询

子查询中只能返回一个字段的数据。

可以将子查询的结果作为 WHRER 语句的过滤条件：

```
SELECT *
FROM mytable1
WHERE col1 IN (SELECT col2
               FROM mytable2);
```

下面的语句可以检索出客户的订单数量，子查询语句会对第一个查询检索出的每个客户执行一次：

```
SELECT cust_name, (SELECT COUNT(*)
                   FROM Orders
                   WHERE Orders.cust_id = Customers.cust_id)
                   AS orders_num
FROM Customers
ORDER BY cust_name;
```

十五、连接

连接用于连接多个表，使用 JOIN 关键字，并且条件语句使用 ON 而不是 WHERE。

连接可以替换子查询，并且比子查询的效率一般会更快。

可以用 AS 给列名、计算字段和表名取别名，给表名取别名是为了简化 SQL 语句以及连接相同表。

内连接

内连接又称等值连接，使用 INNER JOIN 关键字。

```
SELECT A.value, B.value
FROM tablea AS A INNER JOIN tableb AS B
ON A.key = B.key;
```

可以不明确使用 INNER JOIN，而使用普通查询并在 WHERE 中将两个表中要连接的列用等值方法连接起来。

```
SELECT A.value, B.value
FROM tablea AS A, tableb AS B
WHERE A.key = B.key;
```

自连接

自连接可以看成内连接的一种，只是连接的表是自身而已。

一张员工表，包含员工姓名和员工所属部门，要找出与 Jim 处在同一部门的所有员工姓名。

子查询版本

```
SELECT name
FROM employee
WHERE department = (
      SELECT department
      FROM employee
      WHERE name = "Jim");
```

自连接版本

```
SELECT e1.name
FROM employee AS e1 INNER JOIN employee AS e2
ON e1.department = e2.department
      AND e2.name = "Jim";
```

自然连接

自然连接是把同名列通过等值测试连接起来的，同名列可以有多个。

内连接和自然连接的区别：**内连接提供连接的列，而自然连接自动连接所有同名列。**

```
SELECT A.value, B.value
FROM tablea AS A NATURAL JOIN tableb AS B;
```

外连接

外连接保留了没有关联的那些行。分为左外连接，右外连接以及全外连接，左外连接就是**保留左表没有关联的行。**

检索所有顾客的订单信息，包括还没有订单信息的顾客。

```
SELECT Customers.cust_id, Customer.cust_name, Orders.order_id
FROM Customers LEFT OUTER JOIN Orders
ON Customers.cust_id = Orders.cust_id;
```

customers 表：

cust_id	cust_name
1	a
2	b
3	c

orders 表：

order_id	cust_id
1	1
2	1
3	3
4	3

结果：

cust_id	cust_name	order_id
1	a	1
1	a	2
3	c	3
3	c	4
2	b	Null

十六、组合查询

使用 **UNION** 来组合两个查询，如果第一个查询返回 M 行，第二个查询返回 N 行，那么组合查询的结果一般为 M+N 行。

每个查询必须包含相同的列、表达式和聚集函数。

默认会去除相同行，如果需要保留相同行，使用 UNION ALL。

只能包含一个 ORDER BY 子句，并且必须位于语句的最后。

```
SELECT col
FROM mytable
WHERE col = 1
UNION
SELECT col
FROM mytable
WHERE col =2;
```

十七、视图

视图是虚拟的表，本身不包含数据，也就不能对其进行索引操作。其内容由查询定义。具有普通表的结构，但是**不实现数据存储**。

对视图的操作和对普通表的操作一样。

视图具有如下好处：

- 简化复杂的 SQL 操作，比如复杂的连接；
- 只使用实际表的一部分数据；

- 通过只给用户访问视图的权限，保证数据的安全性；
- 更改数据格式和表示。

```
CREATE VIEW myview AS
SELECT Concat(col1, col2) AS concat_col, col3*col4 AS compute_col
FROM mytable
WHERE col5 = val;
```

作用：

1. **简化了操作**，把经常使用的数据定义为视图。

我们在使用查询时，在很多时候我们要使用聚合函数，同时还要 显示其它字段的信息，可能还会需要关联到其它表，这时写的语句可能会很长，如果这个动作频繁发生的话，我们可以创建视图，这以后，我们只需要`select * from view`就可以啦，这样很方便。

2. 安全性，用户只能查询和修改能看到的数据。

因为视图是虚拟的，物理上是不存在的，只是存储了数据的集合，我们可以将基表中重要的字段信息，可以不通过视图给用户，视图是动态的数据的集合，数据是随着基表的更新而更新。同时，用户对视图不可以随意的更改和删除，可以保证数据的安全性。

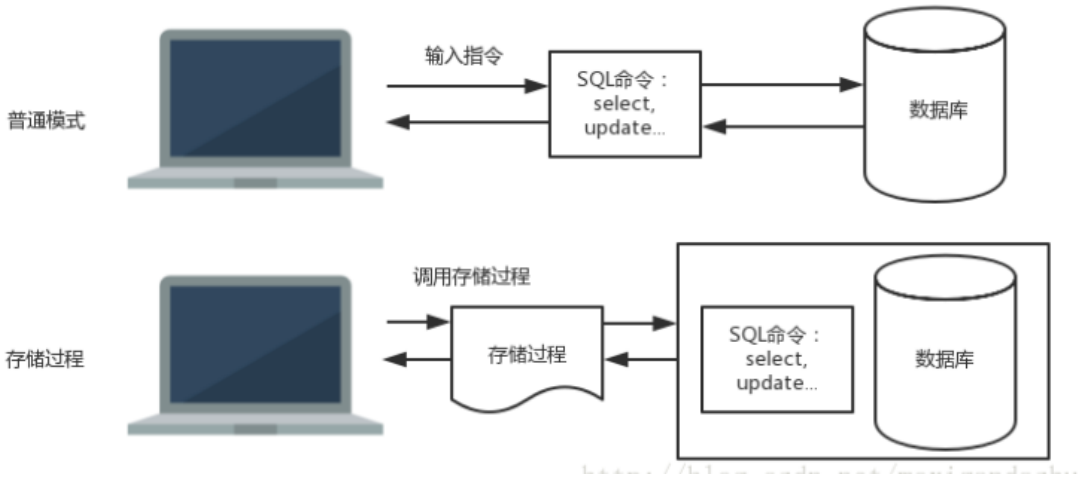
3. 逻辑上的独立性，屏蔽了真实表的结构带来的影响。

视图可以使应用程序和数据库表在一定程度上独立。如果没有视图，应用一定是建立在表上的。有了视图之后，程序可以建立在视图之上，从而程序与数据库表被视图分割开来。

十八、存储过程

存储过程可以看成是对一系列 SQL 操作的批处理。

> 存储过程（Stored Procedure）是在大型数据库系统中，**一组为了完成特定功能的SQL语句集**，存储在数据库中，经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象。
> ——百度百科



使用存储过程的好处：

- 代码封装，保证了一定的安全性(可以防止对表的直接访问，只需要赋予用户存储过程的访问权限)；
- 代码复用；
- 由于是预先编译，因此具有很高的性能。

命令行中创建存储过程需要自定义分隔符(`delimiter //`)，因为命令行是以；为结束符，而存储过程中也包含了分号，因此会错误把这部分分号当成是结束符，造成语法错误。

存储过程中具体的处理内容放在`BEGIN`和`END`之间；

存储过程需要制定参数，包括**种类**（`IN,OUT,INOUT`，分别代表输入参数，输出参数和既是输入也是输出的参数），**参数名和数据类型**。【和函数不同，函数指定输入参数即可】

给变量赋值都需要用 select into 语句。

每次只能给一个变量赋值，不支持集合的操作。

```
delimiter //

create procedure myprocedure( out ret int )
    begin
        declare y int;
        select sum(col1)
        from mytable
        into y;
        select y*y into ret;
    end

delimiter ;
```

执行存储过程 CALL
调用存储过程使用CALL 存储过程名命令，具体如下：

CALL 存储过程名(参数,...)
eg:通过创建好的存储过程sp_search_cusotmer来执行存储过程：
检索'王'姓顾客：

```
CALL sp_search_customer('王%');
```

检索所有顾客：

```
CALL sp_search_customer('');
```

```
call myprocedure(@ret);
select @ret;
/*在创建存储过程的时候，如果制定了OUT护着INOUT，在调用存储过程时请在输出参数前面加上@，这样结果将保存到“@变量名“中。*/
```

十九、游标

在存储过程中使用游标可以对一个结果集进行移动遍历。

游标主要用于交互式应用，其中用户需要对数据集中的任意行进行浏览和修改。

使用游标的四个步骤：

1. 声明游标，这个过程没有实际检索出数据；
2. 打开游标；
3. 取出数据；
4. 关闭游标；

```
delimiter //
create procedure myprocedure(out ret int)
    begin
        declare done boolean default 0;

        declare mycursor cursor for
        select col1 from mytable;
        # 定义了一个 continue handler，当 sqlstate '02000' 这个条件出现时，会执行 set
done = 1
        declare continue handler for sqlstate '02000' set done = 1;

        open mycursor;

        repeat
            fetch mycursor into ret;
            select ret;
        until done end repeat;

        close mycursor;
    end //
 delimiter ;
```

二十、触发器

触发器会在某个表执行以下语句时而自动执行：DELETE、INSERT、UPDATE。

触发器必须指定在语句执行之前还是之后自动执行，之前执行使用 BEFORE 关键字，之后执行使用 AFTER 关键字。BEFORE 用于数据验证和净化，AFTER 用于审计跟踪，将修改记录到另外一张表中。

INSERT 触发器包含一个名为 NEW 的虚拟表。

```
CREATE TRIGGER mytrigger AFTER INSERT ON mytable
FOR EACH ROW SELECT NEW.col into @result;

SELECT @result; -- 获取结果
```

DELETE 触发器包含一个名为 OLD 的虚拟表，并且是只读的。

UPDATE 触发器包含一个名为 NEW 和一个名为 OLD 的虚拟表，其中 NEW 是可以被修改的，而 OLD 是只读的。

MySQL 不允许在触发器中使用 CALL 语句，也就是不能调用存储过程。

二十一、事务管理

基本术语：

- 事务（transaction）指一组 SQL 语句；
- 回退（rollback）指撤销指定 SQL 语句的过程；

- 提交（commit）指将未存储的 SQL 语句结果写入数据库表；
- 保留点（savepoint）指事务处理中设置的临时占位符（placeholder），你可以对它发布回退（与回退整个事务处理不同）。

不能回退 SELECT 语句，回退 SELECT 语句也没意义；也不能回退 CREATE 和 DROP 语句。

MySQL 的事务提交默认是隐式提交，每执行一条语句就把这条语句当成一个事务然后进行提交。当出现 START TRANSACTION 语句时，会关闭隐式提交；当 COMMIT 或 ROLLBACK 语句执行后，事务会自动关闭，重新恢复隐式提交。

设置 autocommit 为 0 可以取消自动提交；autocommit 标记是针对每个连接而不是针对服务器的。

如果没有设置保留点，ROLLBACK 会回退到 START TRANSACTION 语句处；如果设置了保留点，并且在 ROLLBACK 中指定该保留点，则会回退到该保留点。

```
START TRANSACTION
// ...
SAVEPOINT delete1
// ...
ROLLBACK TO delete1
// ...
COMMIT
```

二十二、字符集

基本术语：

- 字符集为字母和符号的集合；
- 编码为某个字符集成员的内部表示；
- 校对字符指定如何比较，主要用于排序和分组。

除了给表指定字符集和校对外，也可以给列指定：

```
CREATE TABLE mytable
(col VARCHAR(10) CHARACTER SET latin COLLATE latin1_general_ci )
DEFAULT CHARACTER SET hebrew COLLATE hebrew_general_ci;
```

可以在排序、分组时指定校对：

```
SELECT *
FROM mytable
ORDER BY col COLLATE latin1_general_ci;
```

二十三、权限管理

MySQL 的账户信息保存在 mysql 这个数据库中。

```
USE mysql;
SELECT user FROM user;
```

创建账户

新创建的账户没有任何权限。

```
CREATE USER myuser IDENTIFIED BY 'mypassword';
```

修改账户名

```
RENAME USER myuser TO newuser;
```

删除账户

```
DROP USER myuser;
```

查看权限

```
SHOW GRANTS FOR myuser;
```

授予权限

账户用 username@host 的形式定义，username@% 使用的是默认主机名。

```
GRANT SELECT, INSERT ON mydatabase.* TO myuser;
```

删除权限

GRANT 和 REVOKE 可在几个层次上控制访问权限：

- 整个服务器，使用 GRANT ALL 和 REVOKE ALL；
- 整个数据库，使用 ON database.*；
- 特定的表，使用 ON database.table；
- 特定的列；
- 特定的存储过程。

```
REVOKE SELECT, INSERT ON mydatabase.* FROM myuser;
```

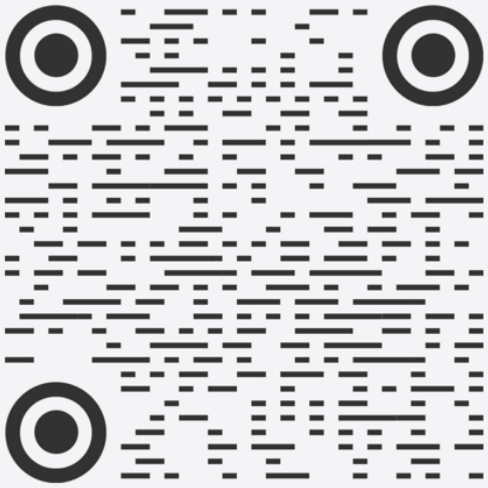
更改密码

必须使用 Password() 函数进行加密。

```
SET PASSWROD FOR myuser = Password('new_password');
```

参考资料

- BenForta. SQL 必知必会 [M]. 人民邮电出版社, 2013.



公众号 CyC2018
帮助你快速学习成长
并拿到心仪的 Offer
回复 CyC 即可领取学习资料