# 操作数据库

## 1. MySQL语句的规范

1. 关键字与函数名称全部大写。
2. 数据库名称、表名称、字段名称全部小写。
3. SQL语句必须以分号结尾。

## 2. 操作数据库(清空命令行：cls)

1. CREATE创建：CREATE {DATABASE | SCHEMA} [IF NOT EXISTS](可有可无) db\_name [DEFAULT] CHARACTER SET [=] charset\_name
2. SHOW 查询：SHOW DATABASES(查看全部数据库)、SHOW WARNINGS(查看警告)、SHOW CREATE DATABASE db\_name(查看数据库创建时的信息)
3. ALTER 修改：ALTER {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name [DEFAULT] CHARACTER SET [=] charset\_name(用来修改数据库的编码方式)
4. DROP 删除：DROP {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name

# 二、操作数据表

## 1. 数据表中的数据类型

整型，浮点型，字符型，日期时间型









## 2. 创建数据表

(1) 指定数据库：USE db\_name;

查看数据表：SHOW TABLES [FROM db\_name];

查看数据表结构：SHOW COLUMNS FROM table\_name;

CREATE TABLE students

（

id INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY COMMENT’编号’,

name CHAR(8) NOT NULL COMMENT’名字’,

sex CHAR(4) NOT NULL COMMENT’性别’,

age TINYINT UNSIGNED NOT NULL COMMENT’年龄’,

tel CHAR(13) NULL DEFAULT "-" COMMENT’电话’

) COMMENT=’学生表’;

(2) 以 "id int unsigned not null auto\_increment primary key" 行进行介绍:

"id" 为列的名称;

"INT" 指定该列的类型为 int(取值范围为 -8388608到8388607), 在后面我们又用 "UNSIGNED" 加以修饰, 表示该类型为无符号型, 此时该列的取值范围为 0到16777215;

"NOT NULL" 说明该列的值不能为空, 必须要填, 如果不指定该属性, 默认可为空;

"AUTO INCREMENT" 需在整数列中使用, 其作用是在插入数据时若该列为 NULL, MySQL将自动产生一个比现存值更大的唯一标识符值。在每张表中仅能有一个这样的值且所在列必须为索引列。需要和主键组合使用。

"PRIMARY KEY" 表示该列是表的主键, 本列的值必须唯一, MySQL将自动索引该列。

下面的 CHAR(8) 表示存储的字符长度为8, TINYINT的取值范围为 -127到128, default 属性指定当该列值为空时的默认值。

“COMMENT”每一行的注释，对表也可以注释

## 3. 表名称的修改

(1) 修改单个表：ALTER TABLE 表名 RENAME [TO|AS] 新表名

(2) 可以修改多个表：RENAME TABLE tbl\_name TO new\_tbl\_name [, tbl\_name2 TO new\_tbl\_name2] ...

## 4. 表结构列的添加\删除

(1) 添加单列：ALTER TABLE user1 ADD 列名声明AFTER/FIRST username(列名字);

单列添加可以指定某个字段之前或者之后，不使用小括号

(2) 添加多列：ALTER TABLE user1 ADD (列名声明，列名声明，……)；

多列添加使用小括号，而且不能指定位置，只能位于所有字段之后。

(3) 删除列：ALTER TABLE user1 DROP 列名，DROP 列名；

(4) 修改列定义 MODIFY：ALTER TABLE 表名 MODIFY 列名 数据类型 其他属性(FIRST|AFTER);

1. 修改列名称 CHANGE：ALTER TABLE 表名 CHANGE 列名 新列名 数据类型 其他属性

## 5. 表的删除

(1) DROP TABLE table\_name；

# 三、约束

## 1. 约束的分类

(1) 约束分为表级约束和列级约束

表级约束：对多个数据列建立的约束。表级约束只能在列定义后声明

列级约束：对一个数据列建立的约束。列级约束既可以在列定义时声明，也可以在列定以后声明

(2) 约束类型包括

* 非空约束 NOT NULL
* 主键约束 PRIMARY KEY
* 唯一约束 UNIQUE KEY
* 默认约束 DEFAULT
* 外键约束 FOREIGN KEY（物理层面的外键，现在一般用逻辑外键）

## 2. 约束的添加\删除

(1) 添加主键约束：ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)

删除主键约束：ALTER TABLE tb\_name DROP PRIMARY KEY;

(2) 添加唯一约束：ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] UNIQUE [index\_type] (index\_col\_name,...)

删除唯一约束：ALTER TABLE tbl\_name DROP { INDEX |KEY} Iindex\_name

查询约束名字：SHOW INDEXES FROM tb\_name\G;

删除唯一约束用名字。

(3) 添加外键约束：ALTER TABLE tbl\_name ADD [CONSTRAINT [symbol]] FOREIGN KEY [index\_type] (index\_col\_name,...) reference\_definition

删除外键约束：ALTER TABLE tb\_name DROP FOREIGN KEY fk\_symbol;

删除外键约束后，外键的索引仍然存在

通过查看SHOW CREATE TABLE tb\_name 中的外键名删除外键

ALTER TABLE tb\_name DROP INDEX pid;删除外键PID字段的索引。

1. 添加默认约束：ALTER TABLE tbl\_name ALTER [COLUMN] col\_name SET DAFAULT literal

删除默认约束：ALTER TABLE tbl\_name ALTER [COLUMN] col\_name DROP DEFAULT

# 四、操作数据表中的记录

## 1、插入记录 INSERT

1.1 标准用法

(1) INSERT [INTO] table\_name [(column\_name,...)] {VALUES|VALUE} ({expr(表达式）|DEFAULT（默认值）},...),(...),...;

INSERT [INTO] 表名 [(列名1,列名2,...)] VALUES(列值1,列值2,...),(列值3,列值4,…),…;

(2) 可同时插入多条记录，多条记录括号间用逗号“,”隔开。

(3) 对于自动编号的字段，插入“NULL”或“DEFAULT”系统将自动依次递增编号；

(4) 对于有默认值的字段，可以插入“DEFAULT”表示使用默认值；

(5) 列值可传入数值（1，2，3....）、表达式（如3\*7+1）或函数。

如：密码可以用md5()函数(PHP中的函数，求字符串哈希值函数)（如md5('123')）；

* 1. INSERT 的三种方法

1. INSERT tbl\_name VALUES(值1,值二,值三,表达式1,……)[,(值1,值二,值三,……)]

可以同时插入多条记录，但不能使用子查询,记录的值可以是值，也可以是表达式（不加单引号）

1. INSERT tbl\_name SET col\_name1=值1,col\_name2=值2。

只能一次性插入一条记录，但可以使用子查询subquery。

1. INSERT tbl\_name[col\_name] SELECT……

将查询结果插入表中

插入的表名和查询的结果要匹配

比如：INSERT users(username) SELECT username FROM users3;

## 2. 单表更新 UPDATE

(1) UPDATE table\_reference SET col\_name1={expr1|DEFAULT} [,col\_name2={expr2|DEFAULT}]... [WHERE where\_condition]

(2) 如：对id=3的记录，将其age值加5，sex值设为0。

UPDATE user SET age = age + 5,sex = 0 WHERE id = 3;

如果不加WHERE条件就是更新全部数据。

## 3. 单表删除 DELETE

(1) DELETE FROM table\_name [WHERE ...]

DELETE FROM users WHERE id=6; -----删除id为6的数据。

(2) 当删除了一个纪录时原有纪录的ID号也被删除，则新加的记录的ID号为当前所存在的最大的ID号再加一；

## 4. 查找记录 SELECT

(1) 当要查询多个数据表的数据时可以采用：

SELECT tbl\_name.col\_name FROM tbl\_name;

可以在查询多个表的同时分清字段是哪个数据表的。

查询顺序不一样，结果集顺序也不一样。查询多个列用逗号，隔开。

(2) 当查询的字段名字复杂时可以用AS方法

SELECT col\_name [AS] newName from tbl\_name;

来记录别名，同时查询的结果的名称也会随着改变

AS不建议省略，修改别名会影响结果集的显示。

## 5. 查询关键字

5.1 条件表达式 WHERE

对记录进行过滤，如果没有指定WHERE子句，则显示所有记录。

在WHERE表达式中，可以使用MySQL支持的函数或者运算符。

5.2 查询结果分组 GROUP BY

(1) SELCET \* FROM 表名 [GROUP BY {col\_name列名或位置} [ASC|DESC],....]

----默认为ASC （ASC 升序 DESC 降序）

GROUP BY优先指定列名SELECT sex FROM users GROUP BY sex；

(2) GROUP BY本质是对查询结果集中相同列进行合并分组处理。

(3) 对于SELECT使用\*的时候，不使用GROUP BY。一般结果集中存在聚合函数，除了聚合函数其他字段都要GROUP BY。例如SELECT有三个字段，这三个字段必须都GROUP BY。

* 1. 分组条件 HAVING

1. 在GROUP BY后接HAVING使用时，HAVING后的表达式必须满足以下条件之一：

HAVING后的字段出现在SELECT所查询的字段中；

HAVING后的字段没有出现在SELECT查询的字段中，但是这是一个**聚合函数**（只有一个值的函数，如：count()、max()、avg()、sum()...）

1. HAVING本质是对GROUP BY之后的结果集进行再次筛选，所以字段是必须GROUP BY之内的字段。
   1. 对查询结果进行排序 ORDER BY
2. SECLECT \* FROM 表名 [ORDER BY 列名 ASC|DESC，列名 ASC|DESC];

对查询结果进行排序，可以有多个条件。

如果第一个条件还不足排序，就拿下一个条件在原来的基础上在进行一次排序。

5.5 限制查询结果返回值 LIMIT

(1) [LIMIT{[offset,]row\_count|row\_count OFFSET offset}]

如：LIMIT 偏移量，数量；

LIMIT 2,3；

顺序以结果集为准，从0开始排序。

1. LIMIT一般用作分页。例如每页十条数据，LIMIT 0,10 LIMIT 9,10

# 五、子查询和连接

## 1. 子查询

1.1 基本子查询

(1) 子查询是指出现在其他SQL语句内的SELECT子句

如：

SELECT \* FROM t1 WHERE column1 = (SELECT column1 FROM t2);

其中，SELECT \* FROM t1 ...称为Outer Query[外查询](或者Outer Statement)

SELECT column1 FROM t2 称为Sub Query[子查询]

(2) 子查询指嵌套在查询内部，且必须始终出现在圆括号内。

子查询可以包含多个关键字或者条件，如DISTINCT,GROUP BY,ORDER BY,LIMIT,函数等

子查询的外层查询可以是：【SELECT,INSERT,UPDATE,SET或DO】

子查询的返回值：标量、一行、一列或者子查询

1.2 使用比较运算符的子查询（比较运算符：> , >= , < , <= , = , !=）

1. 当子查询返回多个结果时，需要使用修饰运算符，

* SELECT id FROM tb1 WHERE user = ANY/SOME/ALL (SELECT user FROM tb2);

其中：ANY和SOME时等价的，只要满足一个条件，ALL需要满足所以条件

* 除此还有IN和NOT IN

SELECT id FROM tb1 WHERE user = NOT IN/IN (SELECT user FROM tb2);

其中：NOT IN 等价于 != ALL;而 IN 等价于 = ANY/SOME

1. SELECT AVG(goods\_price)FROM tdb\_goods; 求平均值

SELECT ROUNDE(AVG(goods\_price),2)FROM tdb\_goods; 平均值留2小数位数4舍5入

* 子查询如：

SELECT goods\_id,goods\_name,goods\_price FROM tdb\_goods WHERE

goods\_price>(SELECT ROUND(AVG(goods\_price),2) FROM tdb\_goods);

1.3 使用 [NOT] IN 的子查询

(1) ANY运算符与IN等效,等于任意一个，从这个范围中选择。

！=ALL或<>ALL运算符NOT IN等效，不等于全部，除了ALL之外。不在这个范围中选择。

(2) 如：SELECT goods\_id,goods\_name,goods\_price FROM tdb\_goods

WHERE goods\_price IN (SELECT goods\_price FROM tdb\_goods WHERE goods\_cate = '超级本') ORDER BY goods\_price DESC;

1.4 使用 [NOT] EXISTS 的子查询

如果子查询返回任何行，EXISTS将返回TRUE，否则返回FALSE。

1.5 使用 子查询 INSERT SELECT 插入记录

(1) 新设表

CREATE TABLE IF NOT EXISTS tdb\_goods\_cates(

cate\_id SMALLINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

cate\_name VARCHAR(40) NOT NULL

);

(2) 在新表中按照商品表填充数据

SELECT goods\_cate FROM tdb\_goods GROUP BY goods\_cate;

SHOW COLUMNS FROM tdb\_goods\_cates;

INSERT tdb\_goods\_cates(cate\_name) SELECT goods\_cate FROM tdb\_goods GROUP BY goods\_cate;

## 2. 连接

(1) **连接的本质是，第一步先将多张表合并成一张临时表，再根据连接的规则筛选临时表，最后返回筛选完成的临时表**。

* 1. 连接类型：

1. 内连接 INNER JOIN，在MySQL中JOIN,CROSS JOIN,INNER JOIN是等价的

左外连接 LEFT[OUTER] JOIN

右外连接 RIGHT[OUTER] JOIN

全外连接 MySql没有全外连接

笛卡尔积 全匹配结果集

合并结果集 UNION 或 UNION ALL

1. 关键字ON一般用于设定连接条件。
   1. 内连接

SELECT \* FROM student INNER JOIN teacher ON id = t\_id;

1. 表示将表A和B连接形成一张临时表，筛选规则是两张表的交集，筛选条件是id=t\_id
2. INNER JOIN ON和WHERE的效果其实是一样的，INNER JOIN是显示连接，WHERE是隐式连接。当用WHERE筛选一张表的信息时，其实是将该表复制一份然后将两张表进行连接，再取出交集的结果。
3. 筛选条件的执行顺序在FROM之后，所以可以使用FROM当中的别名，但是不能使用SELECT中的别名。
   1. 左外连接：LEFT JOIN 右外连接：RIGHT JOIN

SELECT \* FROM student LEFT JOIN teacher ON id = t\_id;

SELECT \* FROM student RIGHT JOIN teacher ON id = t\_id;

1. 将表A和B连接形成一张临时表，左外连接的筛选规则是取左边表的全部，再加上右边表交集的部分。如果右边表没有符合左边表列的部分，全部默认为NULL。右外连接则是相反。
   1. 全外连接（MySql中没有全外连接）
2. 全外连接的规则是，取左边表的全部，右边表没有符合的默认为NULL。取右边表的全部，左边表没有符合的默认为NULL。由于结果集出现列的顺序不一样，即使结果一样但是也不会被认为是重复的结果集。
   1. 笛卡尔积

SELECT \* FROM student , teacher;（没有设定任何条件）

1. 笛卡尔积的规则是，如果表A有3条记录，表B有4条记录。则笛卡尔积就是有3 X 4条记录，相当于A中的每一条记录都会匹配到B中的所有记录。
   1. 合并结果集 UNION 和 UNION ALL

SELECT id,`name` FROM student b

UNION

SELECT t\_id,t\_name FROM teacher a ;

1. UNION表示将已经筛选出的结果集进行合并显示，执行顺序是在结果集出来之后，和GROUP BY一样。而连接是在结果集还没出来之前，是辅助筛选结果集的。
2. 因为要合并显示，所以要求两个结果集的列数必须相等，列名也最好相等。
3. UNION 表示会去除重复的结果，UNION ALL 表示不会去除重复的结果。
   1. 连接表的数量
4. 自我连接，相当于复制一张相同的表，将两张表连接起来。
5. 多表连接，相当于组成三张表（也可以别的数量）的临时表，再根据筛选规则和筛选条件进行筛选。
   1. 连接的其他应用
6. 多表删除

多表删除(单表模拟多表删除重复数据)

DELETE t1 FROM tdb\_goods AS t1 LEFT JOIN (SELECT goods\_id,goods\_name FROM tdb\_goods GROUP BY goods\_name HAVING count(goods\_name) >= 2) AS t2 ON t1.goods\_name = t2.goods\_name WHERE t1.goods\_id > t2.goods\_id;

1. 多表更新

参照另外一张表 来更新 本表。

UPDATE table\_1 连接方式 table\_2 ON 连接条件 SET 更新值；

如：UPDATE tdb\_goods INNER JOIN tdb\_goods\_cates ON goods\_cate = cate\_name SET goods\_cate = cate\_id;

# 六、运算符和函数

## 1. 字符函数

(1) CONCAT() ,字符连接

SELECT CONCAT('a','-','b'); 结果为:a-b

(2) CONCAT\_WS(), 使用指定的分隔符进行字符连接

SELECT CONCAT\_WS('|','A','B','C'); 结果为： A|B|C

1. FORMAT() 数字格式化

SELECT FORMAT(12560.7,2); 结果：12，560.70

SELECT FORMAT(12560.78,1); 结果：12，560.8

1. LOWER() 转换成小写字母

UPPER() 转换成大写字母

1. LEFT() 获取左侧字符

SELECT LEFT('mysql',2); 结果：my

RIGHT() 获取右侧字符

1. LTRIM() 删除前导空格(=LEFT TRIM())

RTRIM() 删除后续空格

1. TRIM（）删除前后两边的空格，还可以删除指定的前导和后续的字符,不能删除中间的字符

SELECT TRIM(LEADING'?','??MYSQL????'); 结果：MYSQL????

SELECT TRIM(TRAILING'?','??MYSQL????'); 结果：??MYSQL

SELECT TRIM(BOTH'?','??MYSQL???'); 结果：MYSQL

1. REPLACE() 替换字符

SELECT REPLACE('??MYSQL???','?','-'); 结果：--MYSQL---

1. SUBSTRING(string，offset，length) 截取字符串

SELECT SUBSTRING('MYSQL',2,3); 结果：SQL

(10 NOT]LIKE 模糊匹配

(%)：代表任意个字符，0个或多个

(\_)：代表任意一个字符，只有一个

SELECT name FROM test WHERE name LIKE'%O%'; 结果：输入name 中带‘O’的name

SELECT name FROM test WHERE name LIKE'%1%%' ESCAPE'1'; 找到中间带% 的匹配name

## 2. 数值运算符和函数



1. WER(数值,数值)

例子：SELECT POWER(3,3); 结果为27；

1. ROUND(数值,小数的位数)
2. TRUNCATE(数值，截取位数)

说明：和ROUND()相似，只是不四舍五入，截取位数还能是负数，如：SELECT TRUNCATE(125.68,-1); 结果为120；

## 3. 比较运算符和函数

都是返回布尔值1或者0

[NOT]BETWEEN...AND... ：在一个范围区间

[NOT]IN() ：在特定的几个点，SELECT 10 IN(5，10，15，20);

IS [NOT] NULL : 判断是否为空

## 4. 日期时间函数

1. NOW() 当前时间 含日期时间
2. CURDATE() 当前日期 只有日期
3. CURTIME() 当前时间 值有时间
4. DATE\_ADD() 时间增减或减少

DATE\_ADD('2014-3-12',INTERVAL +-365 DAY/YEAR/WEEK/MONTH) 加减365天

1. DATEDIFF() 两个时间的差值

DATEDIFF('2014-3-12','2014-6-12')

1. DATE\_FORMAT() 时间的格式化

DATE\_FORMAT('2014-3-12','%m/%d/%Y')

Y要大写。

## 5. 信息函数

CONNECTION\_ID() //连接ID

DATABASE() //当前数据库

LAST\_INSERT\_ID() //最后插入记录的ID号。

表中必须存在一个自动编号的字段，设置为主键，名字不一定叫ID。而且如果最后一条记录一次插入了多个数据，显示的是插入的多个数据的第一个数据的ID号。

USER() //当前用户

VERSION() //版本信息

## 6. 聚合函数—返回值只有一个值

AVG() 平均值

COUNT() 计数

MAX() 最大值

MIN() 最小值

SUM() 求和

## 7. 加密函数

MD5(); //信息摘要算法

PASSWORD(); //加密算法，主要用途修改当前用户密码

SET PASSWORD=PASSWORD(‘dimiter’);

# 七、自定义函数

调用自定义函数就和其他函数一样，不能用CALL调用。

## 1. 创建函数

(1) 函数可以返回任意类型的值,同样可以接受这些类型的参数；

函数的参数与返回值之间，没有必然的联系。

(2) 创建自定义函数

CREATE FUNCTION function\_name

RETURNS

{STRING|INTEGER|REAL|DECIMAL}

routine\_body - 函数体

(3) 函数体

函数体由合法的SQL语法构成;

函数体可以是简单的SELECT或INSERT语句;

函数体如果为复合结构则使用BEGIN...END语句；

复合结构可以包括声明，循环，控制结构。

## 创建不带参数的函数

1. CREATE FUNCTION f1() RETURNS VARCHAR(30)

RETURN DATE\_FORMAT(now(),'%Y年%m月%d日 %H点:%i分:%s秒');

调用：SELECT f1();

## 创建带有参数的函数

1. CREATE FUNCTION f2（num1 SMALLINT UNSIGNED,num2 SMALLINT UNSIGNED）

RETURNS FLOAT(10，2) UNSIGNED

RETURN(num1+num2)

删除函数：DROP FUNCTION fun\_name;

## 创建复合结构的函数

修改分隔符：DELEMITER 分隔符

删除函数：DROP FUNCTION [IF EXISTS] function\_name

DELIMITER //

CREATE FUNCTION adduser(username VARCHAR(20))

RETURNS INT UNSIGNED

BEGIN

INSERT test(username) VALUES(username);

RETURN LAST\_INSERT\_ID();

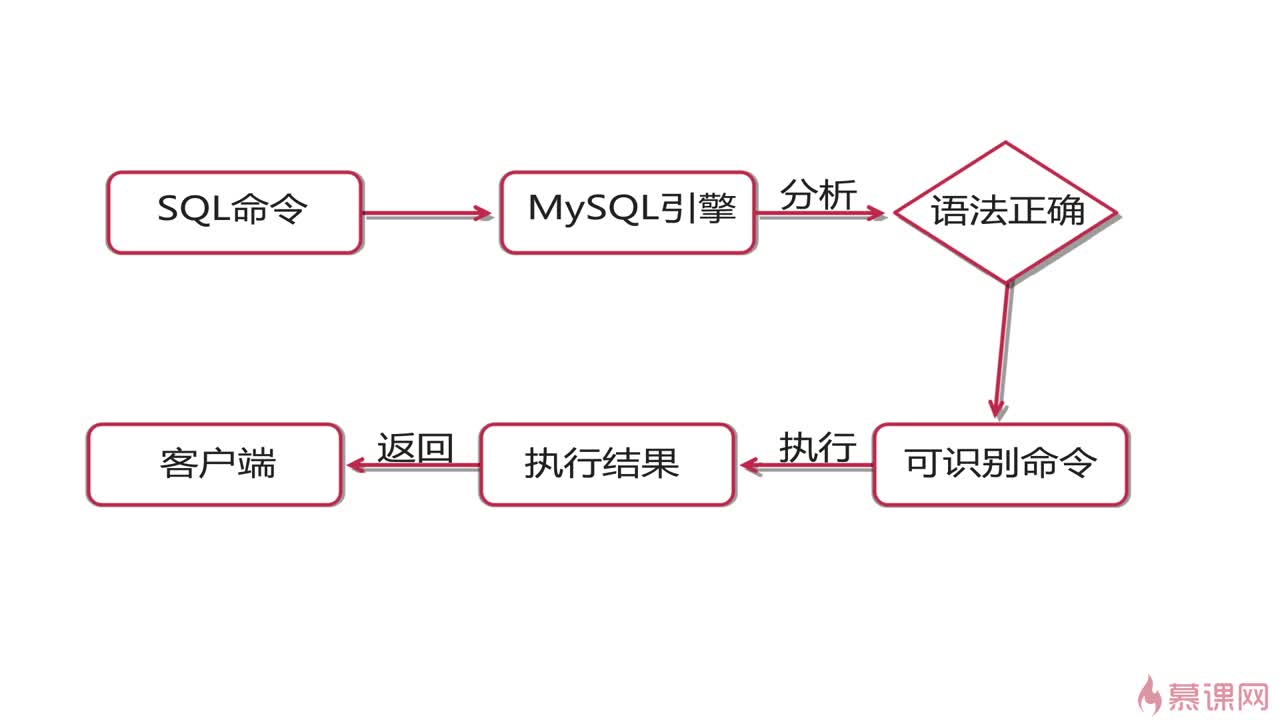
END

//

# 八、不用MySQL的存储过程

1、存储过程

* 是SQL语句与控制语句的【预编译集合】，以【一个名称存储】作为【一个单元处理】
* 优点：
* 增强了语句的功能和灵活性：可以通过控制语句对流程进行控制和判断
* 实现较快的执行速度，只在【第一次调用时进行语法分析和编译】 ，以后直接从内存中得到结果
* 减少网络流量



2、创建存储过程

* 存储过程语法结构分析
* CREATE

[DEFINER = {user|CURRENT\_USER}]//定义时的用户，若是不写就默认为当前用户

PROCEDURE sp\_name ([proc\_parameter[,...]]) //可以带0到多个参数 sp\_name存储过程的名字

[characteristic ...] routine\_body

* proc\_parameter:

[IN|OUT|INOUT] param\_name type

IN, 表示该参数的值必须在调用存储过程时指定

OUT, 表示该参数的值可以被存储过程改变，并且可以返回

INOUT, 表示该参数的值调用时指定，并且可以被改变和返回

* 特性
* COMMENT :注释
* CONTAINS SQL :包含SQL语句, 但不包含读或写数据的语句
* NO SQL :不包含SQL语句
* READS SQL DATA :包含读数据的语句
* MODIFIES SQL DATA :包含写数据的语句
* SQL SECURITY {DEFINER|INVOKER}：指明谁有权限来执行
* .过程体
* 过程体由合法的SQL语句构成；
* 过程体可以是任意SQL语句;对表格进行增删，连接，但是不能创建数据表<br>
* 过程体如果为复合结构则使用BEGIN...END语句
* 复合结构可以使用条件、循环等控制语句

3、创建没有参数的存储过程

* 创建：CREATE PROCEDURE sp1() SELECT VERSION();
* 调用存储过程：

CALL sp\_name([parameter[,...]])

CALL sp\_name[()]

无参的过程调用可以不写小括号如第二种，但是带参的过程必须使用小括号带参数

4、创建带有IN类型参数的存储过程

* 修改存储过程：

ALTER PROCEDURE sp\_name[characteristic]

跟之前的自定义函数相同只能简单修改注释、当前内容类型，不能修改过程体。

* 删除存储过程：

DROP PROCEDURE [IF EXISTS] sp\_name

* CREATE PROCEDURE removeUserById(IN p\_id INT UNSIGNED)

BEGIN

DELETE FROM users WHERE id=p\_id;

END

参数中的变量名字不要跟表中名字一样

5、创建带有IN 和 OUT 参数的存储过程

* DELIMITER //
* CREATE PROCEDURE removeUserAndReturnUserNums(IN p\_id INT UNSIGNED,OUT userNums INT UNSIGNED)

BEGIN

DELETE FROM users3 WHERE id = p\_id;

SELECT count(id) FROM users3 INTO userNums;

END

//

调用：CALL removeUserAndReturnUserNums(3,@nums);

查询：SELECT @nums;

* 关于变量：
* 用户变量：以"@"开始，形式为"@变量名"

用户变量跟mysql客户端是绑定的，设置的变量，只对当前用户使用的客户端生效

* 全局变量：定义时，以如下两种形式出现，set GLOBAL 变量名 或者 set @@global.变量名

对所有客户端生效。只有具有super权限才可以设置全局变量

* 会话变量：只对连接的客户端有效。
* 局部变量：作用范围在begin到end语句块之间。在该语句块里设置的变量

declare语句专门用于定义局部变量。set语句是设置不同类型的变量，包括会话变量和全局变量

declare可以声明变量，比如 declare @a int，只不过声明全局变量可以省略declare，而在begin--end里面声明需要用declare，而且要放在第一行

6、创建多个OUT 的存储过程ROW\_COUNT 插入删除或者更新的记录总数

* ROW\_COUNT 插入删除或者更新的记录总数
* 实例：删除相应的字段，并显示被删除的行数，也显示删除后剩下的行数

DELIMITER //

CREATE PROCEDURE rmUserByAgeAndRtInfos(IN p\_age SMALLINT UNSIGNED, OUT delNums SMALLINT UNSIGNED, OUT leftNums SMALLINT UNSIGNED)

BEGIN

DELETE FROM users WHERE age = p\_age; //注意变量不同

SELECT ROW\_COUNT() INTO delNums;//此处的SELECT... INTO…只是在赋值

SELECT COUNT(id) FROM users INTO leftNums;//同上

END

//

DELIMITER ;

CALL rmUserByAgeAndRtInfos(20, @a, @b); //删除记录数，剩下记录数

SELECT @a, @b;

7、存储过程和自定义函数的区别

* 存储过程实现的功能相对复杂，函数针对性较强
* 存储过程可以返回多个值，函数只能有一个返回值
* 存储过程一般独立执行，函数可以作为 sql 语句的组成部分来出现

另外，存储过程也比通过API接口调用程序要快。

# 九、不用视图

# 十、不用MySQL 存储引擎

1、存储引擎---一种存储技术

* MySQL可以将数据以不同的技术存储在文件（内存）中，不同的技术就称为不同的存储引擎。

每种存数引擎使用不同的存储机制、索引技巧、锁定水平，最终提供广泛且不同的功能。

使用不同的存储引擎也可以说不同类型的表

* MySQL支持的存储引擎
* MyISAM
* InnoDB
* Memory
* CSV
* Archive

2、并发控制

* 当多个连接对记录进行修改时保证数据的一致性和完整性。系统使用锁系统来解决这个并发控制，这种锁分为：
* 共享锁（读锁）---在同一时间内，多个用户可以读取同一个资源，读取过程中数据不会发生任何变化。
* 排他锁（写锁）---在任何时候只能有一个用户写入资源，当进行写锁时会阻塞其他的读锁或者写锁操作。
* 锁的力度（也叫锁的颗粒）

锁颗粒(锁定时的单位）

---表锁，是一种开销最小的锁策略。得到数据表的写锁

---行锁，是一种开销最大的锁策略。并行性最大

* 表锁的开销最小，因为使用锁的个数最小，行锁的开销最大，因为可能使用锁的个数比较多。

3、事务

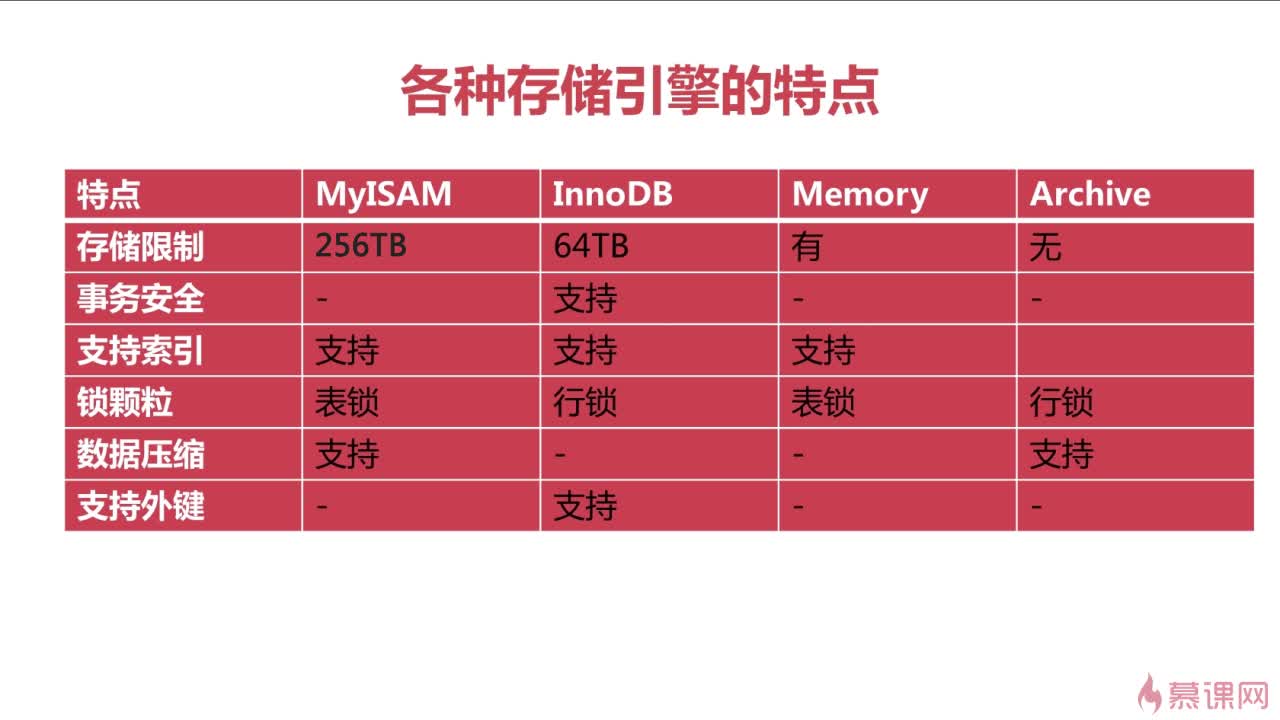
* 数据库系统区别于文件系统的重要特性
* 事务处理：整个过程每一个单元全部完成才算事务处理成功，某一个单元失败事务就会回滚.
* 主要作用：保证数据库的完整性
* 事务的特性：

原子性、一致性、隔离性、持久性

简称：A(Atomic)C(Consistency)I(Isolation)D(Durable)

4、外键和索引

* 外键：是保证数据一致性的策略。
* 索引：是对数据表中一列或者多列的值进行排序的一种结构。使用索引可以快速访问数据表的特定信息。索引是进行记录快速定位的一种方法。索引好比目录。索引可以划分为：普通索引、唯一索引、全文索引。



5、各个存储引擎

* Memory的存储限制是由内存的大小来决定。
* 索引：普通索引、唯一索引、全文索引、btree索引、hash索引……
* CSV(分隔符值)存储引擎不支持索引。
* BlackHole:黑洞引擎，写入的数据都会消失，一般用做数据复制的中继。
* 使用最广泛的两种存储引擎：MyISAM/InnoDB
* MyISAM：适用于事务的处理不多的情况。
* InnoDB：适用于事务处理比较多，需要有外键支持的情况。

6、设置存储引擎

* 通过修改MySQL配置文件实现

default-storage-engine = engine

* 通过创建数据表命令来实现

CREATE TABLE tbl\_name(

...

...

) ENGINE = engine;

* 通过修改数据表命令实现

ALTER TABLE tbl\_name ENGINE [=] engine\_name;