MySQL学习笔记

* 关键字与函数名全部大写
* 数据库名称、表名称、字段名称全部小写
* SQL语句必须以分号结尾
* Prompt提示符：修改提示符。eg: prompt \u@\h \d;
* \h 服务器名称，\D完整的日期，\d当前数据库，\u 当前用户。
* 显示当前服务器版本 SELECT VERSION();
* 显示当前日期时间 SELECT NOW();
* 显示当前用户 SELECT USER();
* 改变客户端编码 SET NAMES gbk;
* 日期格式化 用汉语表示：SELECT DATE\_FORMAT(NOW(),’%Y年%m月%d日%H点：%i分：%s秒’)；
* 数据类型：是指列、存储过程参数、表达式和局部变量的数据特征，它决定了数据的存储格式，代表了不同的信息类型。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 整形 | 数据类型 | TINYINT | SMALLINT | MEDIUMINT | INT | BIGINT |
| 字节 | 1 | 2 | 3 | 4 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 浮点型 | 数据类型 | FLOAT[(M,D)] | DOUBLE[(M,D)] |
| 存储范围 | -3.402823466E+38至-1.175494351E-38、0和1.175494351E-38到3.402823466E+38。M是数字总位数，D是小数点后位数。如果M和D被省略，根据硬件允许的限制来保存值。单精度浮点数精确到大约7位小数位。 | -1.7976931348623157E+308到-2.2250738585072014E-308、0和2.2250738585072014E-308到1.7976931348623157E+308 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 日期时间型 | 列类型 | YEAR | TIME | DATE | DATETIME | TIMESTAMP |
| 存储字节 | 1 | 3 | 3 | 8 | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字符型 | 列类型 | CHAR(M) | VARCHAR(M) | TINYTEXT | TEXT |
| 存储字节 | M个字节，0<=M<=255 | L+1个字节  L<=M且0<=M<=65535 | L+1个字节  L<28 | L+2个字节  L<216 |
|  | MEDIUMTEXT | LONGTEXT | ENUM(‘value1’,’value2’,,,,) | SET(‘value1’,’value2’,,,,) |
|  | L+3个字节  L<224 | L+4个字节  L<232 | 1或2个字节，取决于枚举值的个数（最多65535个值） | 1,2,3,4或者8个字节，取决于SET成员的数目（最多64个成员） |

* 创建数据库：

CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db\_name [DEFAULT] CHARACTER SET [=] charset\_name

Eg: CREATE DATABASE t1; CREATE DATABASE t2 CHARACTER SET gbk;

* 查看当前服务器下的数据表列表

SHOW {DATABASE | SCHEMA} [LIKE ‘pattern’ | WHERE expr]

Eg: SHOW DATABASES;

* 查看警告信息

SHOW WARNINGS;

* 显示数据库创建时所用的指令

SHOW CREATE DATABASE t1;

* 修改数据库

ALTER {DATABASE | SCHEMA} [db\_name] [DEFAULT] CHARACTER SET [=] charset\_name

* 删除数据库

DROP {DATABASE | SCHEMA} [IF EXISTS] db\_name

* 数据表操作

数据表（或称表）是数据库最重要的组成部分之一，是其他的对象的基础。

Eg: USE 数据库名称;

* 创建表

CREATE TABLE [IF NOT EXISTS] table\_name(

column\_name data\_type,

……

)

Eg: CREATE TABLE tb1(

username VARCHAR(20),

age TINYINT UNSIGNED,

salary FLOAT(8,2) UNSIGNED

)

* 查看当前已经打开的数据库

SELECT DATABASE();

* 查看表列表

SHOW TABLES [FROM db\_name] [LIKE ‘pattern’ | WHERE expr]

Eg: SHOW TABLES FROM test;

* 查看表结构

SHOW COLUMNS FROM tb1\_name;

Eg: SHOW COLUMNS FROM tb1;

* 插入记录

INSERT [INTO] tb1\_name [(col\_name,…)] VALUES (value1,…);

Eg: INSERT tb1 VALUES(‘Tom’,25,7863.25);

* 记录查找

SELECT expr,… FROM tb1\_name;

* 编辑数据表的默认存储引擎

MySQL配置文件 default-storage\_engine = INNODB

* 查看数据表索引

SHOW INDEX FROM tb1\G;

* 主键

主键约束：每张数据表只能存在一个主键；主键保证记录的唯一性；主键自动为NOT NULL；Auto\_Increment必须与主键组合使用，使主键可以不与Auto\_Increment一起使用。

唯一约束：UNIQUE KEY，唯一约束可以保证记录的唯一性；唯一约束的字段可以为空值NULL；每张数据表可以存在多个唯一约束。

Eg: CREATE TABLE tb1( id SMALLINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY, usenname VARCHAR(20) NOT NULL UNIQUE KEY, age TINYINT UNSIGNED );

默认值：DEFAULT,当插入记录时，如果没有明确的字段赋值，则自动赋予默认值。

Eg: CREATE TABLE tb1( id SMALLINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(20) NOT NULL UNIQUE KEY, sex ENUM(‘1’,’2’,’3’) DEFAULT 3);

* 约束

约束可保证数据的完整性和一致性；约束分为表级约束和列级约束；

约束类型包括： 1. NOT NULL (非空约束)

2. PRIMARY KEY (主键约束)

3. UNIQUE KEY (唯一约束)

4. DEFAULT (默认约束)

5. FOREIGN KEY (外键约束)

* 外键约束（FOREIGN KEY）

保持数据一致性，完整性；实现一对一或一对多关系。

外键约束的要求：

1. 父表和子表必须使用相同的存储引擎，而且禁止使用临时表；
2. 数据表的存储引擎只能为INNODB
3. 外键列和参照列必须具有相似的数据类型。其中数字的长度或是否有符号位必须相同，而字符的长度则可以不同，
4. 外键咧和参照列必须创建索引，如果外键列不存在索引的话，MySQL将自动创建索引。

Eg: CREATE TABLE provinces( id SMALLINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT, pname VARCHAR(20) NOT NULL);

CREATE TABLE users( id SMALLINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT, username VARCHAR(20) NOT NULL, pid SMALLINT UNSIGNED, FOREIGN KEY (pid) REFERENCES provinces(id) );

* 外键约束的参照操作

1. CASCADE：从父表删除或更新且自动删除或更新字表中匹配的行；
2. SET NULL：从父表删除或更新行，并设置字表中的外键列为NULL。如果使用该选项，必须保证字表列没有指定NOT NULL；
3. RESTRICT：拒绝对父表的删除或更新操作；
4. NO ACTION：标准SQL的关键字，在MySQL中与RESTRICT相同。

Eg： CREATE TABLE users1( id SMALLINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT, username VARCHAR(20) NOT NULL, pid SMALLINT UNSIGNED, FOREIGN KEY (pid) REFERENCES provinces (id) ON DELETE CASCADE );

* 表级约束和列级约束

1. 对一个数据列建立的约束，称为列级约束
2. 对多个数据列建立的约束，称为表级约束
3. 列级约束既可以在列定义时声明，也可以在列定义后声明
4. 表级约束只能在列定义时声明

* 修改数据表

添加单列：ALTER TABLE tb1\_name ADD [COLUMN] col\_name column\_definition [FIRST | AFTER col\_name];

Eg: ALTER TABLE users1 ADD age TINYINT UNSIGNED NOT NULL DEFAULT 10;

添加多列：ALTER TABLE tb1\_name ADD [COLUMN] (col\_name column\_definition, …)

删除列：ALTER TABLE tb1\_name DROP [COLUMN] col\_name;

Eg: ALTER TABLE users1 DROP password, DROP age;

* 添加删除约束

添加主键约束：ALTER TABLE tb1\_name ADD [CONSTRAINT[symbol]] PRIMARY KEY [index\_type] (index\_col\_name,…);

Eg: ALTER TABLE users2 ADD CONSTRAINT pk\_users2\_id PRIMARY KEY (id);

添加唯一约束：可以在一张表中添加多个唯一约束

ALTER TABLE tb1\_name ADD [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY] [index\_name] [index\_type] (index\_col\_name,…);

Eg: ALTER TABLE user2 ADD UNIQUE (username);

添加外键约束：ALTER TABLE tb1\_name ADD [CONSTRAINT [symbol]] FOREIGN KEY [index\_name] (index\_col\_name,…);

Eg: ALTER TABLE user2 ADD FOREIGN KEY (pid) REFERENCES provinces(id);

添加/删除默认约束：ALTER TABLE tb1\_name ALTER [COLUMN] col\_name {SET DEFAULT literal | DROP DEFAULT};

Eg: ALTER TABLE users ALTER age SET DEFAULT 15;

删除主键约束： ALTER TABLE tb1\_name DROP PRIMARY KEY;

删除唯一约束/索引：ALTER TABLE tb1\_name DROP {INDEX | KEY} index\_name;

Eg: ALTER TABLE user2 DROP INDEX username;

删除外键约束：ALTER TABLE tb1\_name DROP FOREIGN KEY fk\_symbol;

修改列定义：ALTER TABLE tb1\_name MODIFY [COLUMN] col\_name column\_definition [FIRST | AFTER col\_name];

修改列名称： ALTER TABLE tb1\_name CHANGE [COLUMN] old\_col\_name new\_col\_name column\_definition [FIRST | AFTER col\_name];

数据库更名：ALTER TABLE tb1\_name RENAME [TO | AS] new\_tb1\_name;

RENAME TABLE tb1\_name TO new\_tb1\_name [, tb1\_name2 TO new\_tb1\_name2,…];

* 记录操作

INSERT:

1. INSERT [INTO] tb1\_name [(col\_name,…)] {VALUES | VALUE} ({expr | DEFAULT},…),(…),…;如果主键设置了自增，给主键赋值可以为NULL或DEFAULT。Eg: INSERT users VALUES(NULL|DEFAULT,’John’,’456’,25,1);
2. INSERT [INTO] tb1\_name SET col\_name = {expr | DEFAULT,}… 说明：与第一种方式的区别在于，此种方法可以使用子查询（SubQuery），且不能一次插入多条数据。
3. INSERT [INTO] tb1\_name [(col\_name,…)] SELECT… 说明：此方法可以将查询结果插入到指定数据表。

Eg: INSERT test (username) SELECT username FROM users WHERE age >= 30;

UPDATE:

1. 更新记录（单表更新）：UPDATE [LOW\_PRIORITY] [IGNORE] table\_reference SET col\_name1 = {expr | DEFAULT} [,col\_name2 = {expr2 | DEFAULT}]… [WHERE where\_condition];

Eg: UPDATE users set age = age+5; 让表中所有记录的年龄+5

1. 删除记录（单表删除）DELETE FROM tb1\_name [WHERE where\_condition];
2. 查找记录

SELECT select\_expr[,select\_expr,…]

[ FROM table\_references

[WHERE where\_condition]

[GROUP BY {col\_name | position} [ASC | DESC],…] 分组

[HAVING where\_condition] 分组条件

[ORDER BY {col\_name } expr |position} [ASC | DESC],…] 对结果排序

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

]

查找某个函数或表达式时可以省略后面的表: SELECT version(); SELECT Now();

* 查询表达式

每一个表达式表示想要的一列，必须有至少一个。多个列之间可以英文逗号分隔。

星号（\*）表示所有列。Tb1\_name.\* 表示命名表的所有列。

查询表达式可以使用[AS] alias\_name 为其赋予别名。

别名可用于GROUP BY, ORDER BY 或HAVING 子句中。

Eg: SELECT id AS userId, username AS uname FROM users;

在WHERE表达式中，，，可以用MySQL支持的函数或运算符。

HAVING: SELECT sex, age FROM users GROUP BY | HAVING age > 35; age要么在结果中，要么HAVING, 子句是针对聚合函数的。

ORDER BY : SELECT \* FROM user ORDER BY id DESC;

LIMIT: SELECT \* FROM user LIMIT 2,2; 从3条开始，截2条记录，即3,4。

* 子查询

Subquery 是指出现在其他SQL语句内的SELECT子句。例如：SELECT \* FROM t1 WHERE col1 = (SELECT col2 FROM t2);

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.

子查询返回可以是标量，一行，一列或子查询。

1. 使用比较运算符的子查询（=,>,<,>=,<=,<>,!=,<=>）

语法结构： operand comparison\_operator subquery

Eg: SELECT AVG(goods\_price) FROM tdb\_goods;

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

SELECT goods\_id,goods\_name,goods\_price FROM tdb\_goods WHERE goods\_price >= (SELECT ROUND(AVG(goods\_price),2) FROM tdb\_goods);

1. 当子查询返回多个结果时，可以使用ANY,SOME或ALL 修饰的比较运算符。

operand comparison\_operator ANY(subquery)

operand comparison\_operator SOME(subquery)

operand comparison\_operator ALL(subquery)

|  |  |  |  |
| --- | --- | --- | --- |
|  | ANY | SOME | ALL |
| >, >= | 最小值 | 最小值 | 最大值 |
| <, <= | 最大值 | 最大值 | 最小值 |
| = | 任意值 | 任意值 |  |
| <>,!= |  |  | 任意值 |

Eg: SELECT goods\_id,goods\_name,goods\_price FROM tdb\_goods WHERE goods\_price > (SELECT goods\_price FROM tdb\_goods WHERE goods\_cate = ‘超极本’); 只要大于子查询最小值即可。

1. 使用[NOT] IN 的子查询

语法结构： operand comparison\_operator [NOT] IN (subquery)

= ANY 运算符与IN 等效

！= ANY 运算符或<> ALL运算符与NOT IN 等效。

1. 使用[NOT] EXISTS的子查询

如果子查询返回任何行，EXISTS将返回TRUE,否则为FALSE。

1. INSERT…SELECT

将查询结果写入数据库

INSERT [INTO] tb1\_name [(col\_name,…)] SELECT …

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

* 多表更新

UPDATE table\_references SET col\_name1 = {expr | DEFAULT} [,col\_name = { expr2 | DEFAULT }]… [WHERE where\_condition];

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

CREATE …SELECT: 创建数据表的同时将查询结果写入到数据表

CREATE TABLE [IF NOT EXISTS] tb1\_name [(create\_definition,…)] select\_statement

Eg: CREATE TABLE tdb\_goods\_brands (brand\_id SMALLINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT, brand\_name VARCHAR(40) NOT NULL) SELECT brand\_name FROM tdb\_goods GROUP BY brand\_name;

Eg: UPDATE tdb\_goods AS g INNER JOIN tdb\_goods\_brands AS b ON g.brand\_name = b.brand\_name SET g.brand\_name = b.brand\_id;

Eg: ALTER TABLE tdb\_goods CHANGE goods\_cate cate\_id SMALLINT UNSIGNED NOT NULL, CHANGE brand\_name brand\_id SMALLINT UNSIGNED NOT NULL;

* 连接

MySQL在SELECT语句，多表更新，多表删除语句中支持JOIN操作。

1. 语法结构：table\_reference { [INNER | CROSS] JOIN | {LEFT | RIGHT} [OUTER] JOIN } table\_reference ON condition\_expr;
2. 数据表参照： table\_reference

Tb1\_name [ [AS] alias ] | table\_subquery [AS] alias

数据表可以使用tb1\_name AS alias\_name 或 tb1\_name alias\_name 赋予别名。 Table\_subquery 可以作为子查询使用在FROM 子句中，这样的子查询必须为其赋予别名。

1. 连接类型： INNER JOIN 内连接，在MySQL中，JOIN, CROSS JOIN 和INNER JOIN 是等价的。 LEFT [OUTER] JOIN 左外连接，RIGHT [OUTER] JOIN 右外连接。
2. 连接条件：使用ON关键字来设定连接条件，也可以使用WHERE来代替，通常使用ON关键来设定连接条件，使用WHERE关键字来进行结果集记录的过滤。
3. 内连接：显示左表及右表符合连接条件的记录。

Eg: SELECT goods\_id, goods\_name, cate\_name FROM tdb\_goods INNER JOIN tdb\_goods\_cates ON tdb\_goods.cate\_id = tdb\_goods\_cates.cate\_id;

1. 左外连接：显示左表的全部记录及右表符合连接条件的记录。
2. 右外连接：显示右表的全部记录及左表符合连接条件的记录。

多表连接：eg: SELECT goods\_id, goods\_name, cate\_name, brand\_name, goods\_price FROM tdb\_goods AS g

INNER JOIN tdb\_goods\_cates AS c ON g.cate\_id = c.cate\_id

INNER JOIN tdb\_goods\_brands AS b ON g.brand\_id = b.brand\_id;

1. 几点说明

A LEFT JOIN B join\_condition

数据表B的结果集依赖数据表A。

数据表A的结果集根据左外连接条件依赖所有数据表（B表除外）。

左外连接条件决定如何检索数据表B（在没有指定WHERE条件的情况下）。

如果数据表A的某条记录符合WHERE条件，但是在数据表B不存在符合连接条件的记录，将生成一个所有列为空的额外的B行。

如果使用内连接查找的记录在连接数据表中不存在，并且在WHERE子句中尝试一下操作：col\_name IS NULL 时，如果col\_name被定义为NOT NULL，MySQL 将在找到符合连接条件的记录后停止搜索更多的行。

1. 无限极分表设计

自身连接：同一个数据表对其自身进行连接

Eg: SELECT s.type\_id, s.type\_name, p.type\_name FROM tdb\_goods\_types AS s LEFT JOIN tdb\_goods\_types p ON s.parent\_id = p.type\_id;

1. 多表删除

DELETE tb1\_name[.\*] [,tb1\_name[.\*]]… FROM table\_references [WHERE where\_condition]

Eg: 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;

* 运算符和函数

1. 字符函数

|  |  |
| --- | --- |
| 函数名称 | 描述 |
| CONCAT() | 字符连接 |
| CONCAT\_WS() | 使用指定的分隔符进行字符连接 |
| FORMAT() | 数字格式化：返回结果字符型 |
| LOWER() | 转换为小写字母 |
| UPPER() | 转换为大写字母 |
| LEFT() | 获取左侧字符 |
| RIGHT() | 获取右侧字符 |
| LENGTH() | 获取字符串长度 |
| LTRIM() | 删除前导空格 |
| RTRIM() | 删除后续空格 |
| TRIM() | 删除前导和后续空格 |
| SUBSTRING() | 字符串截取 |
| [NOT]LIKE | 模式匹配 |
| REPLACE() | 字符串替换 |

Eg: SELECT CONCAT(‘abc’,’-’,’def’); abc-def

Eg: 将两个字段的信息连在一起输出：SELECT CONCAT(first\_name,last\_name) AS fullname FROM test;

Eg: SELECT CONCAT\_WS(‘|’,’A’,’B’,’C’); A|B|C

Eg: SELECT FORMAT(12560.75, 2); 12,560.75

Eg: SELECT FORMAT(12560.75, 1); 12,560.8

Eg: SELECT LOWER(‘MySQL’); mysql

Eg: SELECT UPPER(‘MySQL’); MYSQL

Eg: SELECT LEFT(‘MySQL’,2); My

Eg: SELECT LOWER(LEFT(‘MySQL’,2)); my

Eg: SELECT LENGTH(‘MySQL’); 6

Eg: SELECT LTRIM(‘ MySQL ’); MySQL

Eg: SELECT TRIM(LEADING ‘?’ FROM ‘??MySQL???’); MySQL??? (删除前导?)

Eg: SELECT TRIM(TRAILING ‘?’ FROM ‘??MySQL???’); ??MySQL (删除后续?)

Eg: SELECT TRIM(BOTH ‘?’ FROM ‘??MySQL???’); MySQL (删除前导和后续?)

Eg: SELECT REPLACE(‘??My???SQL???’,’?’,’’); MySQL

Eg: SELECT SUBSTRING(‘MySQL’,1,2); My

Eg: SELECT SUBSTRING(‘MySQL’,-1); L (倒数第几个)

Eg: SELECT ‘MySQL’ LIKE ‘M%’; 1 (即为TRUE )

Eg: SELECT \* FROM test WHERE first\_name LIKE ‘%o%’; 如：tom%

Eg: SELECT \* FROM test WHERE first\_name LIKE ‘%1%%’ ESCAPE ‘1’; 指1后的%不是通配符

%（百分号）：代表任意个字符 \_（下划线）：代表任意一个字符

1. 数值运算符和函数

|  |  |
| --- | --- |
| 名称 | 描述 |
| CEIL() | 进一取整 |
| DIV | 整数除法 |
| FLOOR() | 舍一取整 |
| MOD | 取余数（取模） |
| POWER() | 幂运算 |
| ROUND() | 四舍五入 |
| TRUNCATE() | 数字截取 |

Eg：SELECT CEIL(3.01); 4

Eg: SELECT FLOOR(3.99); 3

Eg: 3 DIV 4; 0

Eg: 5 % 3; 2

Eg: 5 MOD 3; 2

Eg: 5.3 MOD 3; 2.3

Eg: POWER(3,3); 27

Eg: ROUND(3.652, 2); 3.65

Eg: TRUNCATE(125.89, 2); 125.89

Eg: TRUNCATE(125.89, 0); 125

Eg: TRUNCATE(125.89, -1); 120

1. 比较运算符和函数

|  |  |
| --- | --- |
| 名称 | 描述 |
| [NOT] BETWEEN...AND... | [不]在范围之内 |
| [NOT] IN() | [不]在列出值范围之内 |
| IS [NOT] NULL | [不]为空 |

Eg: SELECT 15 BETWEEN 1 AND 22; 1

Eg: SELECT 35 BETWEEN 1 AND 22; 0

Eg: SELECT 10 IN (5,10,15,20); 1

Eg: SELECT 13 IN (5,10,15,20); 0

Eg: SELECT ‘ ’ IS NULL; 0

Eg: SELECT \* FROM test WHERE first\_name IS NULL;

1. 日期时间函数

|  |  |
| --- | --- |
| 名称 | 描述 |
| NOW() | 当前日期的时间 |
| CURDATE() | 当前日期 |
| CURTIME() | 当前时间 |
| DATE\_ADD() | 日期变化 |
| DATEDIFF() | 日期差值 |
| DATE\_FORMAT() | 日期格式化 |

Eg: SELECT DATE\_ADD(‘2014-3-12’, INTERVAL 365 DAY); 2015-3-12

Eg: SELECT DATE\_ADD(‘2014-3-12’, INTERVAL -365 DAY); 2013-3-12

Eg: SELECT DATE\_ADD(‘2014-3-12’, INTERVAL 1 YEAR); 2015-3-12

Eg: SELECT DATE\_ADD(‘2014-3-12’, INTERVAL 3 WEEK); 2014-04-02

Eg: SELECT DATEDIFF(‘2013-3-12’, ‘2014-3-12’); -365

Eg: SELECT DATE\_FORMAT(‘2014-3-12’, ‘%m/%d/%Y’); 03/12/2014

1. 信息函数

|  |  |
| --- | --- |
| 名称 | 描述 |
| CONNECTION\_ID() | 连接ID |
| DATABASE() | 当前数据库 |
| LAST\_INSERT\_ID() | 最后插入记录的ID号 |
| USER() | 当前用户 |
| VERSION() | 版本信息 |

1. 聚合函数：只有一个返回值

|  |  |
| --- | --- |
| 名称 | 描述 |
| AVG() | 平均值 |
| COUNT() | 计数 |
| MAX() | 最大值 |
| MIN() | 最小值 |
| SUM() | 求和 |
| ROW\_COUNT() | 记录表更新后被影响的行数 |

1. 加密函数

|  |  |
| --- | --- |
| 名称 | 描述 |
| MD5() | 信息摘要算法：主要用于为Web页面准备数据加密 |
| PASSWORD() | 密码算法：主要用于改变用户密码 |

Eg: SELECT MD5(‘admin’); 21232f297a57a5a743894a0e4a801fc3

* 自定义函数

自定义函数：用户自定义函数(user-defined function, UDF)是一种对MySQL扩展的途径，其用法与内置函数相同。

两个必要条件：参数和返回值。

函数可以返回任意类型的值，同样可以接收这些类型的参数。

创建自定义函数：CREATE FUNCTION function\_name RETURNS {STRING | INTEGER | REAL | DECIMAL } routine\_body;

关于routine\_body(函数体)：

1. 函数体由合法的SQL语句构成；
2. 函数体可以是简单的SELECT或INSERT语句；
3. 函数体如果为复合结构则使用BEGIN...END语句；
4. 复合结构可以包含声明，循环，控制结构。

Eg: SELECT DATE\_FORMAT(NOW(),’%Y年%m月%d日 %H点：%i分: %s秒’);

对其自定义：CREATE FUNCTION f1() RETURNS VARCHAR(30) RETURN DATE\_FORMAT(NOW(),’%Y年%m月%d日 %H点：%i分: %s秒’);

调用过程：SELECT f1();

带有参数的函数：CREATE FUNCTION f2(num1 SMALLINT UNSIGNED, num2 SMALLINT UNSIGNED) RETURNS FLOAT(10,2) UNSIGNED RETURN (num1+num2)/2;

调用：SELECT f2(10,15); 12.5

创建具有复合结构的自定义函数：DELIMITER //; 用//作为SQL命令的分隔符。

Eg: CREATE FUNCTION addUser(username VARCHAR(20)) RETURNS INT UNSIGNED

BEGIN

INSERT test(username) VALUES (username);

RETURN LAST\_INSERT\_ID();

END

//

调用：SELECT addUser(‘Rose’);

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

* 存储过程

存储过程是SQL 语句和控制语句的预编译集合，以一个名称存储并作为一个单元处理。

存储过程的优点： 增强SQL语句的功能和灵活性；

实现较快的执行速度；

减少网络流量。

CREATE [DEFINER = {user | CURRENT\_USER}] PROCEDURE sp\_name([proc\_parameter[,...] ] ) [ 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：表示该参数的调用时指定，并且可以被改变和返回。

过程体：由合法的SQL语句构成，可以是任意SQL语句，任意一般是指CURD以及多表连接等操作。过程体如果为复合结构则使用BEGIN...END语句，复合结构可以包含声明，循环和控制语句。

特性：COMMENT ‘string’ | {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA } | SQL SECURITY {DEFINER | INVOKER}

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

创建不带参数的存储过程：CREATE PROCEDURE sp1() SELECT VERSION();

调用： 1. CALL sp\_name([parameter,[,...]])

2. CALL sp\_name[()]

Eg: CALL sp1; CALL sp();

创建带有IN参数的存储过程：CREATE PROCEDURE removeUserById(IN p\_id INT UNSIGNED)

BEGIN

DELETE FROM users WHERE id=p\_id;

END

//

调用： CALL removeUserById(3);

删除存储过程：DROP PROCEDURE [IF EXISTS] sp\_name;

创建带有IN和OUT类型参数的存储过程：

CREATE PROCEDURE removeUserAndReturnUserNums (IN p\_id INT UNSIGNED, OUT userNums INT UNSIGNED) BEGIN DELETE FROM users WHERE id=p\_id; SELECT COUNT(id) FROM users INTO userNums; END //

调用：CALL removeUserAndReturnUserNums(27,@nums);

SELECT @nums;

创建带有多个OUT类型参数的存储过程：

CREATE PROCEDURE removeUserByAgeAndReturnInfos(IN p\_age SMALLINT UNSIGNED,OUT deleteUsers SMALLINT UNSIGNED,OUT userCounts SMALLINT UNSIGNED) BEGIN DELETE FROM users WHERE age=p\_age; SELECT ROW\_COUNT() FROM users INTO deleteUsers; SELECT COUNT(id) FROM users INTO userCounts; END //

调用：CALL removeUserByAgeAndReturnInfos(20,@a,@b);

SELECT @a,@b;

存储过程与自定义函数的区别：

1. 存储过程实现的功能要复杂一些，而函数的针对性更强。
2. 存储过程可以返回多个值；函数只能有一个返回值。
3. 存储过程一般独立的来执行；而函数可以作为其他SQL语句的组成部分来出现。

修改存储过程：只有修改一些简单特性，不能修改过程体，若想修改过程体，删除重建。

ALTER PROCEDURE sp\_name [characteristic...]

COMMENT ‘string’ | {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA } | SQL SECURITY {DEFINER | INVOKER}

注意事项：

1. 创建存储过程或者自定义函数需要通过DELIMITER语句修改定界符。

2. 如果函数体或者过程体有多个语句，需要包含在BEGIN...END语句块中。

3. 存储过程通过CALL来调用。

* 存储引擎

存储引擎：MySQL可以将数据以不同的技术存储在文件（内存）中，这种技术就称为存储引擎。每一种存储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。

MYSQL支持的存储引擎：MyISAM , InnoDB, Memory, CSV, Archive.

并发控制锁颗粒： 1. 表锁：是一种开销最小的锁策略

2. 行锁：是一种开销最大的锁策略

事务：事务用于保证数据库的完整性。

事务特性：原子性（Atomicity）一致性（consistency）隔离性（Isolation）持久性（Durability）

索引：是对数据表中一列或多列的值进行排序的一种结构。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特性 | MyISAM | InnoDB | Memory | Archive |
| 存储限制 | 256TB | 64TB | 有 | 无 |
| 事务安全 | -- | 支持 | -- | -- |
| 支持索引 | 支持 | 支持 | 支持 | -- |
| 锁颗粒 | 表锁 | 行锁 | 表锁 | 行锁 |
| 数据压缩 | 支持 | -- | -- | 支持 |
| 支持外键 | -- | 支持 | -- | -- |

通过修改MySQL配置文件实现修改存储引擎：

-default-storage-engine = engine

创建表时：CREATE TABLE table\_name(...) ENGINE=engine;

通过修改数据表实现：ALTER TABLE table\_name ENGINE [=] engine\_name;

lock tables 命令是为当前线程锁定表.这里有2种类型的锁定，一种是读锁定，用命令 lock tables tablename read;另外一种是写锁定，用命令lock tables tablename write.下边分别介绍：

1. lock table 读锁定

如果一个线程获得在一个表上的read锁，那么该线程和所有其他线程只能从表中读数据，不能进行任何写操作。

2. lock table 写锁定

如果一个线程在一个表上得到一个WRITE锁，那么只有拥有这个锁的线程可以从表中读取和写表。其它的线程被阻塞。