# 数据类型

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 大小 范围 |  | 适用范围 |
| tinyint | 01 byte;(-128~127) | 小整数值 |  |
| smallint | 02 byte;(-32,768~32.767) | 大整数值 |  |
| mediumint | 03 byte;(-8,388,608~8,388,607) | 大整数值 |  |
| int/integer | 04 byte;(-2,147,483~648,2,147,483,647) | 大整数值 |  |
| bigint | 08 byte;(-9,223,372,036,854,775,808~9,223,372,036,854,775,807) | 极大整数值 |  |
| float | 04 byte;范围过大 | 单精度浮点数 |  |
| double | 08 byte;范围过大 | 双精度浮点数 |  |
| decimal |  |  |  |

时间日期类型

|  |  |  |
| --- | --- | --- |
| 类型 | 大小 范围 格式 |  |
| date | 03 byte;(1000-01-01~9999-12-31);YYYY-MM-DD | 日期 |
| time | 03 byte;(-838:59:59~838:59:59);HH:MM:SS | 时间值 |
| year | 01 byte;1901~2155;YYYY | 年份 |
| datetime | 08 byte;1000-01-01 00:00:00~9999-12-31 23:59:59;YYYY-MM-DD HH:MM:SS | 日期+时间 |
| timestamp | 04 byte;1970-01-01 00:00:00~2038-01-19 03:14:07;YYYYMMDD HHMMSS | 时间戳 |

字符串类型

|  |  |  |
| --- | --- | --- |
| 类型 | 大小 范围 |  |
| char | 自定义长度，但不可超过范围;0~255 byte | 定长字符串 |
| varchar | 自定义长度，但不可超过范围;0~65535 byte | 变长字符串 |
| tinytext | 自定义长度，但不可超过范围;0~255 byte | 短文本字符串 |
| blog | 自定义长度，但不可超过范围;0~65,535 byte | 长文本(2进制) |
| text | 自定义长度，但不可超过范围;自定义长度，  但不可超过范围;0~65,535 byte | 长文本 |
| mediumblog | 自定义长度，但不可超过范围;0~16,777,215 byte | 中文本(2进制) |
| mediumtext | 自定义长度，但不可超过范围;0~16,777,215 byte | 中文本 |
| longblog | 自定义长度，但不可超过范围;0~4,294,967,295 byte | 极大文本(2进制) |
| longtext | 自定义长度，但不可超过范围;0~4,294,967,295 byte | 极大文本 |

# 基本操作

## 创建数据库

**CREATE DATABASE** `数据库名`

## 创建表

**CREATE TABLE IF NOT EXISTS** `表名`(

…

)**ENGINE =** InnoDB **CHARSET SET utf8;**

**———————————————————————————————————————————————**

**DEFAULT CHARSET**:默认编码格式

字段定义：

AUTO\_INCREMENT：自增序列

PRIMARY KEY：主键约束

ENGING:数据库引擎，mysql支持了InnoDB，memory，archive，myisam。

NOT NULL:约束字段非空

DEFAULT value:默认值

COMMONT 'value':字段注释

## 查看建表语句

**SHOW CREATE TABLE** table\_name;

## 删除数据库表

**DROP TABLE** table\_name;

## 删除表字段

**ALTER TABLE** table\_name **DROP COLUMN** column\_name;

## 新增表字段

**ALTER TABLE** table\_name **ADD** column\_name varchar(255) not null;

## 修改表字段

**ALTER TABLE** table\_name **MODIFY** column\_name …;【例：ALTER TABLE testalter MODIFY name CHAR(10);】

## 查看表字段

**SHOW COLUMNS FROM** checkdb;

## 插入数据

**INSERT INTO** table\_name(,,…,) **VALUES**

(,,…,),(,,…,),…(,,…,)

将A表数据插入B表

**INSERT INTO** table\_A(,,…,) **SELECT** ,,…, **FROM** table\_B

在插入数据项足够填充目标表字段的前提下可以使用简略写法，字段顺序按创建顺序排列：如下

**INSERT INTO** table\_A **VALUES** (,,…,),(,,…,),…(,,…,)

和

**INSERT INTO** table\_A **SELECT** ,,…, **FROM** table\_B

## 删除数据

**DELETE FROM** table\_name **WHERE** …

## 合并查询结果（集合）

**UNION**：查询集合的并集，重复则取一条

**UNION ALL**：将两个集合合并，重复数据不做任何处理

## 排序

**ORDER BY … ASC/DESC**：按某个字段的升序/降序排列查询结果顺序

## 分组

**GROUP BY …**：使用到MySQL函数速查表中统计中的函数时常用分组

## 连接（结果拼合）

**INNER JOIN...ON...**：内连接，按照关联条件获取2张表中都存在的数据（交集）

**LEFT JOIN...ON...**：左连接，获取左表所有记录，按条件关联右表数据，即使右表没有对应匹配的记录

**RIGHT JOIN...ON...**：右连接，获取右表所有记录，按条件关联左表数据，即使右表没有对应匹配的记录

## NULL值处理

MySQL数据库中where条件中只能使用**IS NULL** / **IS NOT NULL**

## 正则表达式

## MySQL函数速查表

|  |  |  |
| --- | --- | --- |
|  | 字符串函数 | 说明 |
| 1 | ascii(s) | 返回字符串s的第一个字符的ASCII码值 |
| 长短 | char\_length(s)  character\_length(s) | 这两个函数等效。返回字符串s的字符数（有多少汉字长度为多少） |
| 验证 | position(s1 in s) | 获取s1在s中开始的位置（从1计数），没有返回0； |
|  |  |
| field(s,s1,s2,…sn) | 返回s在s1…sn中的位置，如a在c,a,r,d中匹配第3位；没有则返回0 |
| find\_in\_set(s1,slist) | 返回s1在slist中匹配的位置，如lo在hello,world匹配第4位；没有则返回0；[注意：slist是字符串，且其中的数据以“，”分割 |
| 大小写转换 | lcase(s1)、lower(s1) | 将s1中所有大写字母转小写（两个函数等效） |
| ucase(s1)、upper(s1) | 将s1中所有大写字母转大写（两个函数等效） |
| 截取 | left(s1,num) | 截取s1中的左n位数据 |
| right(s1,num) | 截取s1中的右n位数据 |
| mid(s,index,len)  substring(s,index,len)  substr(s,index,len) | 从字符串s的index位开始截取len长度的子字符串出来 |
| 拼接/替代/连接 | repeat(s,n) | 将字符串s重复n次拼接到一起 |
| replace(s1,s2,s3) | 将字符串s1中的s2字符串用s3字符串替代 |
| insert(s1,x,len,s2) | 将s1中第x位开始往后len位用s2替代 |
| locate(s1,s2) | 查询s1在s2中的位置，比如c在abcdefg中的位置 |
| concat(s1,s2,…sn) | 连接多个字符串，s1,s2,s3,…,sn |
| concat\_ws(x,s1,s2…sn) | 同concat(s1,s2,...) 函数，但是两个字符串之间要加上x，x可以是分隔符 |
| 填充 | lpad(s1,len,s2) | 从s1开头开始填充s2，使s1长度达到len |
| rpad(s1,len,s2) | 从s2末尾开始填充s2，使s1长度达到len |
| 清除空格 | ltrim(s1) | 清除s1开头空格 |
| rtrim(s1) | 清除s1结尾空格 |
| trim(s1) | 清除s1首尾空格 |
| 倒置 | reverse(s) | s倒置，abcdefg=>gfedcba |
| 18 | space(n) | 返回n个空格 |
| 比较 | strcmp(str1,str2) | str1=str2返回0，str1>str2返回1，str1<str2返回-1 |

|  |  |  |
| --- | --- | --- |
|  | 数字函数 | 说明 |
| 三角函数 | 1. acos(x) 2. acos(x) 3. asin(x) 4. atan(x) 5. atan2(x,m) 6. cos(x) 7. cot(x) 8. tan(x) 9. sin(x) 10. degrees(x) 11. radians(x) 12. PI() | 1. 求 x 的反余弦值(参数是弧度) 2. 求反正弦值(参数是弧度) 3. 求反正切值(参数是弧度) 4. 求反正切值(参数是弧度) 5. 求反正切值(参数是弧度) 6. 求余弦值(参数是弧度) 7. 求余切值(参数是弧度) 8. 求正切值(参数是弧度) 9. 求正弦值(参数是弧度) 10. 将弧度转化为角度(参数是弧度) 11. 将角度转化为弧度(参数是角度) 12. 返回圆周率3.141593 |
| 统计 | greatest(,,…) | 返回~中的最大值 |
| least(,,…) | 返回~中的最小值 |
| max(expression) | 返回expression中的最大值 |
| min(expression) | 返回expression中的最小值 |
| count(express) | 返回查询的总记录数 |
| sum(expression) | 计算总和 |
| avg(expression) | 返回一个表达式的平均值，也可以是某个数字类型的字段 |
| 计算 | n div m | 除法取整，计算的取整结果 |
| mod(n,m) | 除法取余，计算的取余结果 |
| ceil(x)  ceiling(x) | 返回≥x的最小整数，如1.8返回2 |
| floor(x) | 返回≤x的最大整数，如3.1返回3 |
| round(x) | 返回离x最近的整数（四舍五入） |
| abs(x) | 返回x绝对值 |
| truncate(x,y) | x保留y位小数返回(无法四舍五入) |
| 对数 | 1. exp(x) 2. ln(x) 3. log(x) 4. log(base,x) 5. log10(x) 6. log2(x) | 1. 返回E的x次方 2. 返回x的自然对数 3. 返回自然对数(以 e 为底的对数) 4. 带有 base 参数，则 base 为指定带底数 5. 返回以10为底的对数 6. 返回以2为底的对数 |
| 指数 | pow(x,y)  power(x,y) | 计算指数 |
| 平方根 | sqrt(x) | 计算 |
| 随机数 | rand() | 返回0~1的随机数 |
| 正负判断 | sign(x) | 返回x的符号，为负返回-1，正数返回1，0返回0 |

|  |  |  |
| --- | --- | --- |
|  | 日期函数 | 说明 |
| 加天 | adddate(d,n) | 以d日期为准加上n天返回日期 |
| 加秒 | addtime(t,n) | 时间t加上n秒 |
| 当前日期 | curdate()  current\_date() | 格式YYYY-MM-DD |
| 当前时间 | curtime()  current\_time() | 返回时间，格式：HH:mm:SS |
| 当前日期时间 | current\_timestamp() | 返回当前日期和时间 |
| 从日期或日期时间表达式中提取日期值 | date() |  |
| 时间差 | datediff(d1,d2) | 计算日期d1->d2之间相隔天数 |
| 加时间 | date\_add(d,interva n lexprType) | 计算起始日期d加上一个时间段后的日期。select date\_add(date('2020-01-01'),interval 5 day); |
| 减时间 | date\_sub(d,intervaln lexprType) | 计算起始日期d减去一个时间断后的日期。 |
| 格式化 | date\_format(d,f) | 按表达式要求显示日期。  select date\_format(date('2020-01-01'),'%y年%m月%d日'); |
| 获取日 | day(d) | 返回d的日期部分  select day("2017-06-15"); |
| 获取周 | week(d) | 返回d的月份 |
| 获取年 | year(d) | 返回d的年份 |
| 返回星期 | dayname(d) | 返回日期是星期几。 |
| 返回月份中第几天 | dayofmonth(d) | 计算d是本月第几天 |
| 返回周中第几天 | dayofweek(d) | 计算d是本周第几天，星期天是第一天 |
| 返回年中第几天 | dayofyear(d) | 计算d是本年第几天 |

日期格式

|  |  |  |  |
| --- | --- | --- | --- |
| %a | 缩写星期名 | %T | 时间, 24-小时 (hh:mm:ss) |
| %b | 缩写月名 | %U | 周 (00-53) 星期日是一周的第一天 |
| %c | 月，数值 | %u | 周 (00-53) 星期一是一周的第一天 |
| %D | 带有英文前缀的月中的天 | %V | 周 (01-53) 星期日是一周的第一天，与 %X 使用 |
| %d | 月的天，数值(00-31) | %v | 周 (01-53) 星期一是一周的第一天，与 %x 使用 |
| %e | 月的天，数值(0-31) | %W | 星期名 |
| %f | 微秒 | %w | 周的天 （0=星期日, 6=星期六） |
| %H | 小时 (00-23) | %X | 年，其中的星期日是第一天，4 位，与 %V 使用 |
| %h | 小时 (01-12) | %x | 年，其中的星期一是周的第一天，4 位，与 %v 使用 |
| %I | 小时 (01-12) | %Y | 年，4 位 |
| %i | 分钟，数值(00-59) | %y | 年，2 位 |
| %j | 年的天 (001-366) |  |  |
| %k | 小时 (0-23) |  |  |
| %l | 小时 (1-12) |  |  |
| %M | 月名 |  |  |
| %m | 月，数值(00-12) |  |  |
| %p | AM 或 PM |  |  |
| %r | 时间，12-小时（hh:mm:ss AM 或 PM） |  |  |
| %S | 秒(00-59) |  |  |
| %s | 秒(00-59) |  |  |

|  |  |  |
| --- | --- | --- |
|  | 高级函数 | 说明 |
| 类型过渡 | cast(x as type) | 将x转为type类（type可选值：） |
| if(条件,…,…)判断函数 | if(条件,为真时所取，为假时所取,与Oracle的decode一样使用) | 对字段进行判断然后重新返回一个相反集值，如真假，男女，对错。  SELECT IF(LENGTH(email)>10,'大于10','小于10') AS len FROM checkdb; |
| 多元判断case…when | 多元判断 | SELECT email,CASE name          WHEN 'tommy' THEN '汤米'          WHEN 'sam' THEN '山姆'          ELSE '其他'  END as name FROM checkdb |
| 空判断  ifnull(字段,…,…) | 空判断 | 如果字段为空，则取第一个数，反之第二个 |
| 比较判断nullif(val1,val2) | 比较判断 | 比较val1和val2是否相等，相等返回true，否则返回false |
|  |  |  |

## MySQL事务

MySQL中仅Innodb引擎支持事务；事务是用来管理增删查改的业务逻辑。【注意：存储过程不是事务，这是两个概念的东西】

事务特性

* 原子性：一个事务的内容的执行，要么全部完成，要么全部不完成。
* 一致性：事务执行不会破坏数据库完整性。
* 隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
* 持久性：事务执行后，对数据的改变是永久的。

|  |  |
| --- | --- |
| 命令 | 说明 |
| BEGIN / START TRANSACTION | 开启事务 |
| COMMIT / COMMIT WORK | 提交事务 |
| ROLLBACK / ROLLBACK WORK | 回滚并结束事务 |
| SAVEPOINT | 创建保存点 |
| RELEASE SAVEPOINT identifier | 删除保存点 |
| ROLLBACK TO identifier | 回滚到保存点 |
| SET transaction | 隔离级别。InnoDB 存储引擎提供的隔离级别有READ UNCOMMITTED、READ COMMITTED、REPEATABLE READ 和 SERIALIZABLE。 |

简单范例：

START TRANSACTION; --开启事务

DELETE FROM checkdb; --删除

INSERT INTO checkdb(name,email) values ('jack','101'),('sam','102'),('tom','103'), ('tommy','104'),('alice','101');--插入数据

UPDATE checkdb SET email = CONCAT(email,'@qq.com'); --更新数据

COMMIT WORK; --提交事务

## 临时表

没写

## MySQL索引

索引可以简单理解为对数据表某个字段加上目录以提高查询速度；缺点是表加上索引后也降低了表数据的更新速度。

### 新建索引

**CREATE INDEX** nameOfIndex **ON** tableName**(**columnsNameList**)**

或者

**ALTER TABLE** tableName **ADD INDEX** nameOfIndex(columnsNameList)

nameOfIndex:索引名

tableName:表名

columnsNameList:对表中建索引的字段，用“，”隔开

### 查看索引

**SHOW INDEX FROM** checkdb**;**

### 删除索引

**DROP INDEX** nameOfIndex **ON** tableName;

## MySQL存储过程

一组完成特定功能的SQL语句集合，调用call 存储过程名(…参数)执行，可以传递参数，返回结果。【注：存储过程一旦新建便无法修改，只能删了重建】，查询结果只能允许是单一一条数据。

存储过程执行过程中，如果有任意一条sql出错了，也回继续执行下去。

### 新建存储过程

**CREATE PROCEDURE** nameOfProcedure(in type ,out type)

**BEGIN**

…

**END;**

nameOfProcedure:存储过程名称

简单示例：

CREATE PROCEDURE updateMoney(in name\_1 VARCHAR(20),in name\_2 varchar(20),in money double)

BEGIN

UPDATE person set money = money - mon WHERE name = name\_a;

UPDATE person set money = money + mon where name = name\_b;

end;

CALL updateMoney('jack','jacky',100);//调用存储过程

参数详解：

IN:代表参数作为传入使用，

OUT:代表参数承接输出结果；通常用于统计方面，如count，sum，avg；结果仅1行，多行会报错；如果返回结果集则不需要out定义参数；varchar类型的参数数据必须带上长度。out参数定义时需携带@。如下：

CALL countPerson(@sum\_val);

SELECT @sum\_val;

INOUT:代表参数既做传入也做输出使用。函数使用时需要将参数定义好。定义时遵守out的定义方式。如

SET @A = 1, @B = 2;

CALL testPro(@A,@B);

SELECT @A,@B;

### 声明&使用变量

DECLARE a INT DEFAULT(1);

使用时只用变量名即可

### 执行存储过程

**CALL** nameOfProcedure(…);

### 带返回参数执行

call nameofprocedure(@value)

select @value;

### 删除存储过程

**DROP PROCEDURE** nameOfProcedure;

## MySQL函数（自定义）

create function funName() returns INT/VARCHAR/

## mysql分支

### case when

语法一：明确值范围

case

when … then …;

when … then…;

else …;

例子

CREATE DEFINER=`root`@`localhost` PROCEDURE `testPro`(OUT na VARCHAR(20))

BEGIN

DECLARE str VARCHAR(3) DEFAULT '10';

CASE

WHEN str = '100' THEN SET na = '大于100' ;

ELSE SET na = '小于100' ;

END CASE;

END

语法二：明确具体值

case …

when … then …;

when … then…;

else …;

例子

CREATE DEFINER=`root`@`localhost` PROCEDURE `testPro`(OUT na VARCHAR(20))

BEGIN

DECLARE str VARCHAR(3) DEFAULT '10';

CASE str

WHEN '100' THEN SET na = '大于100' ;

ELSE SET na = '小于100' ;

END CASE;

END