位串类型是可以存放一系列二进制位的类型， 相对于二进制类型来说， 此类型在做一些位操作更方便、 更直观。

位串就是一串由0和1组成的字符串。 下面是两种位类型：

bit(n)

bit varying(n)

其中n是一个正整数。

bit(n)类型的数据必须准确匹配长度n， 试图存储短一些或者长一些的数据都是错误的。

bit varying(n)类型的数据是最长为n的变长类型， 更长的串会被拒绝。 写一个没有长度的bit等效于bit(1)， 没有长度的bit varying表示没有长度限制。

如果明确地把一个位串值转换成bit(n)， 那么它的右边将被截断， 或者在右边补齐0到刚好为n位， 而不会抛出任何错误。 类似地， 如果明确地把一个位串数值转换成bit varying(n)， 而其超过n位， 那么它的右边将被截断。

示例：

CREATE TABLE t1 (a BIT(3), b BIT VARYING(5));

INSERT INTO t1 VALUES (B‘101’, B‘00’);

对于bit(n)字段， 如果插入的数据的长度小于定义的长度n或超过了定义的长度n都将报错：

INSERT INTO t1 VALUES (B‘10’, B‘101’);--报错

INSERT INTO test VALUES (B‘11110’, B‘101’);--报错

对于bit varying(n)， 如果插入的数据超过了匹配的长度n也会报错：

INSERT INTO test VALUES (B‘110’, B‘111101’);--报错

**位串的操作符**

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | 例子 |
| bit || bit → bit | 连接 | B'10001' || B'011' → 10001011 |
| bit & bit → bit | 按位与（输入的长度必须相等） | B'10001' & B'01101' → 00001 |
| bit | bit → bit | 按位或 （输入的长度必须相等） | B'10001' | B'01101' → 11101 |
| bit # bit → bit | 按位异或 （输入的长度必须相等） | B'10001' # B'01101' → 11100 |
| ~ bit → bit | 按位求反 | ~ B'10001' → 01110 |
| bit << integer → bit | 按位左移（字符串长度被保留） | B'10001' << 3 → 01000 |
| bit >> integer → bit | 按位右移（字符串长度被保留） | B'10001' >> 2 → 00100 |

位串的函数

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | 例子 |
| bit\_length | 返回位字符串中的位数 | bit\_length(B'10111') → 5 |
| length | 返回位字符串中的位数 | length(B'10111') → 5 |
| octet\_length | 返回位字符串中的字节数 | octet\_length(B'1011111011') → 2 |
| overlay | 替换子串 | overlay(B'01010101010101010' placing B'11111' from 2 for 3) → 0111110101010101010 |
| position | 返回字串位置 | position(B'010' in B'000001101011') → 8 |
| substring | 提取子字符串 | substring(B'110010111111',3,2) → 00 |

其他位串操作符与函数：http://www.postgres.cn/docs/14/functions-bitstring.html

可以在整数和bit之间进行转换。 示例如下：

select 66::bit(10);

select 66::bit(3);

select cast(-66 as bit(12));

下面来看一下PostgreSQL中十进制、 十六进制、 二进制之间的转换示例。

select 85::bit(8);--十进制转二进制

select 'xff'::bit(8);--十六进制转二进制

select 'xff'::bit(8)::int;--十六进制转十进制

select to\_hex(255);--十进制转十六进制