# GP无符号tinyint开发

小组成员：刘小捷51164500225 刘小兵51164500224 刘智51164500226

## 一、背景

### 1.1 有符号tinyint的使用

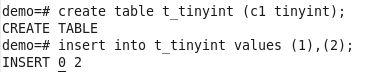
有符号tinyint这一数据类型在数据库中只占用1个字节，相对于4个字节的int或更高8个字节的long数据类型来说能够节省大量的存储空间。而在现实生活中存在着大量的范围较小的整数数据，对于这些数据如果使用tinyint存储在数据库中对于节省数据的存储空间有着重要的意义。因此，其在数据库中的应用非常的广泛。

在Postgresql中可以支持tinyint这一数据类型的使用。首先，从GitHub下载tinyint的扩展包，再将有符号tinyint扩展包放入到contrib目录下，创建拓展，然后就可以很方便的使用这一数据类型了。具体步骤如下：

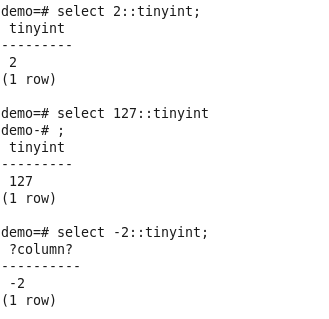
（1）创建扩展

C:\Users\admin\AppData\Local\YNote\data\qq38E179743F54BB51C6CCAF69CA49DE7F\8bd262bc37444b0b99a76f94f78cec7c\clipboard.png

（2）建表与插入数据



（3）数据大小测验



（4）数据范围越界



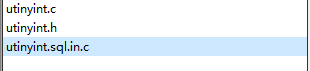
### 1.2 有符号tinyint的扩展

相对于有符号的tinyint这一数据类型，现实生活中更多的是一些无符号的小范围整数，比如年龄、工龄、家庭人口等，故参考有符号tinyint构建无符号的tinyint这一数据类型。对于这些非负的数据，使用无符号tinyint可以有效降低数据的出错概率，比如可以避免出现类似于年龄为-20岁这样的错误。这在一定程度上可以减少脏数据的产生，对于提高数据质量有着重大意义。另外，相对于有符号tinyint最大数值为127，无符号tinyint的范围在0到255。扩展了正整数的使用范围，无符号tinyint将会适用于更多正整数数据存储的应用场景。因此，通过对Greenplum源码的修改，我们在Greenplum中实现了无符号tinyint这一数据类型的扩展。

## 二、无符号tinyint设计与编码

### 2.1 无符号tinyint的设计

（1）代码结构



（2）编译文件



EXTENSION = utinyint

EXTVERSION = 0.1.1

EXTSQL = $(EXTENSION)--$(EXTVERSION).sql

MODULES = utinyint

OBJS = utinyint.o

DATA\_built = utinyint.sql

DATA = uninstall\_utinyint.sql

DOCS = doc/utinyint.md

REGRESS = utinyint

SQL\_IN = utinyint.sql.in

EXTRA\_CLEAN += $(SQL\_IN) $(EXTSQL)

USE\_EXTENSION = $(shell pg\_config --version | grep -qE " 8\.|9\.0" && echo no || echo yes)

ifeq ($(USE\_EXTENSION),yes)

all: $(EXTSQL)

$(EXTSQL): $(EXTENSION).sql

cp $< $@

DATA = $(EXTSQL)

endif

PGXS := $(shell pg\_config --pgxs)

include $(PGXS)

$(SQL\_IN): utinyint.sql.in.c

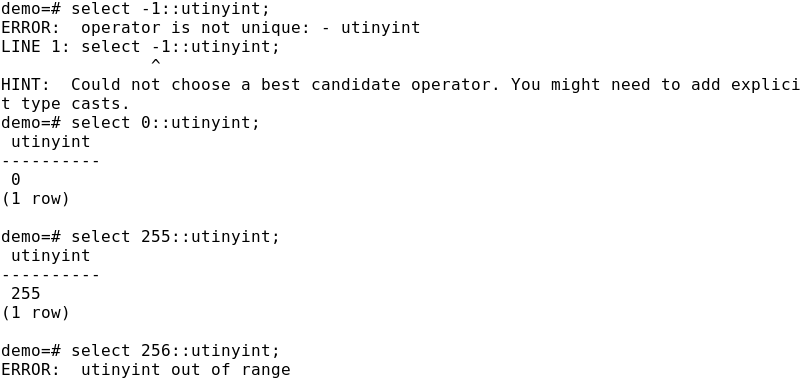
$(CC) -E -P $(CPPFLAGS) $< > $@

（3）使用方式

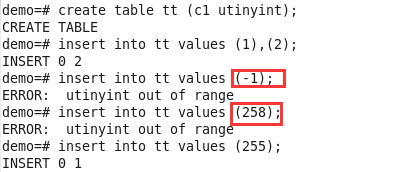
**安装utinyint拓展模块**



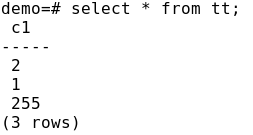
**测试utinyint的取值范围**



**使用utinyint创建一张表**

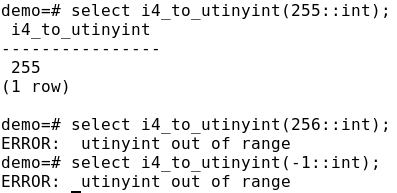


**查看表中数据**

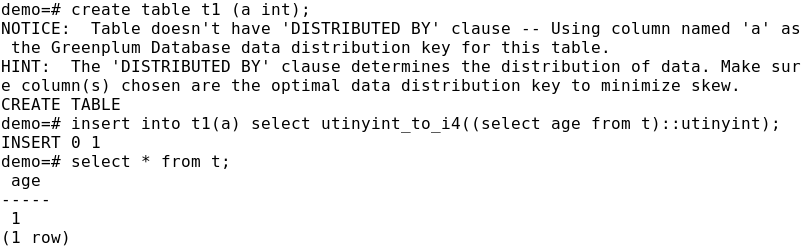


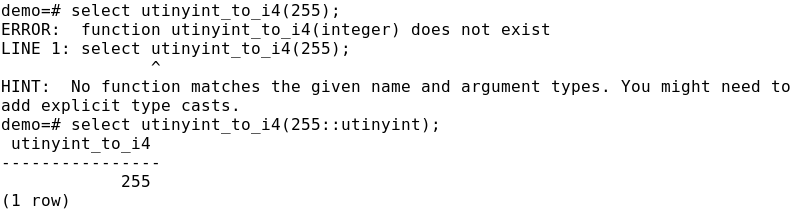
**调用函数**

int类型转utinyint类型函数

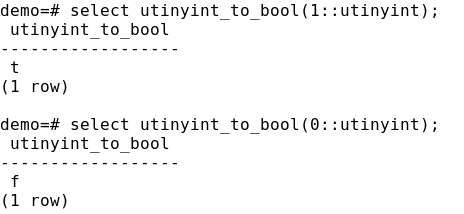


utinyint类型转int类型函数

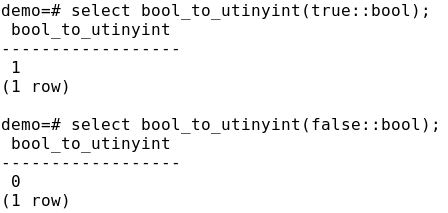




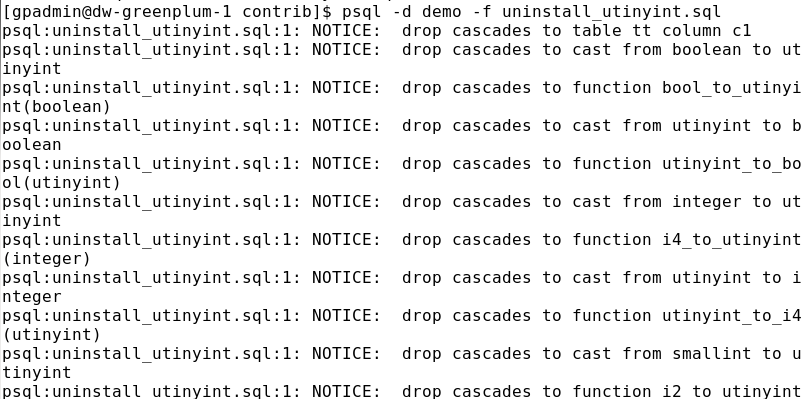
utinyint类型转bool类型函数

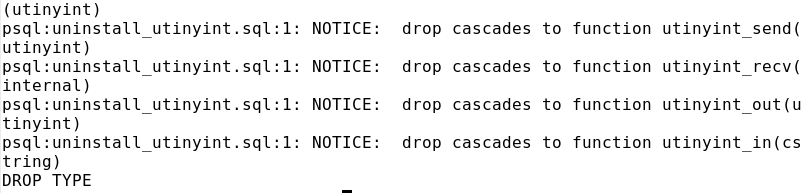


bool类型转utinyint类型函数



**卸载utinyint扩展模块**





### 2.2 无符号tinyint编码

（1）utinyint.h

typedef unsigned char ***utinyint***;

// the functions of SET\_1\_BYTE() and GET\_1\_BYTE() implement in include/postgres.h

#define DatumGetutinyint(X) ((utinyint) (((Datum) (X)) & 0x000000ff))

#define utinyintGetDatum(X) ((Datum) (((Datum) (X)) & 0x000000ff))

#define PG\_GETARG\_utinyint(n) DatumGetutinyint(PG\_GETARG\_DATUM(n))

#define PG\_RETURN\_utinyint(x) return utinyintGetDatum(x)

extern Datum ***utinyint\_in***(PG\_FUNCTION\_ARGS);

extern Datum ***utinyint\_out***(PG\_FUNCTION\_ARGS);

extern Datum ***utinyint\_recv***(PG\_FUNCTION\_ARGS);

extern Datum ***utinyint\_send***(PG\_FUNCTION\_ARGS);

extern Datum ***utinyint\_to\_i2***(PG\_FUNCTION\_ARGS);

extern Datum ***i2\_to\_utinyint***(PG\_FUNCTION\_ARGS);

extern Datum ***utinyint\_to\_i4***(PG\_FUNCTION\_ARGS);

extern Datum ***i4\_to\_utinyint***(PG\_FUNCTION\_ARGS);

extern Datum ***utinyint\_to\_bool***(PG\_FUNCTION\_ARGS);

extern Datum ***bool\_to\_utinyint***(PG\_FUNCTION\_ARGS);

（2）utinyint.c

/\*

\* utinyint\_in - converts "num" to utinyint

\*/

Datum

utinyint\_in(PG\_FUNCTION\_ARGS)

{

char \*num = PG\_GETARG\_CSTRING(0);

PG\_RETURN\_utinyint(pg\_atoi(num, sizeof(utinyint), '\0'));

}

/\*

\* utinyint\_out - converts utinyint to "num"

\*/

Datum

utinyint\_out(PG\_FUNCTION\_ARGS)

{

utinyint arg1 = PG\_GETARG\_utinyint(0);

char \*result = (char \*) palloc(4); /\* 3 digits, '\0' \*/

pg\_itoa(arg1, result);

PG\_RETURN\_CSTRING(result);

}

/\*

\* utinyint\_recv - converts external binary format to utinyint

\*/

Datum

utinyint\_recv(PG\_FUNCTION\_ARGS)

{

StringInfo buf = (StringInfo) PG\_GETARG\_POINTER(0);

PG\_RETURN\_utinyint((utinyint) pq\_getmsgint(buf, sizeof(utinyint)));

}

/\*

\* utinyint\_send - converts utinyint to binary format

\*/

Datum

utinyint\_send(PG\_FUNCTION\_ARGS)

{

utinyint arg1 = PG\_GETARG\_utinyint(0);

StringInfoData buf;

pq\_begintypsend(&buf);

pq\_sendint(&buf, arg1, sizeof(utinyint));

PG\_RETURN\_BYTEA\_P(pq\_endtypsend(&buf));

}

/\*

\* ===================

\* CONVERSION ROUTINES

\* ===================

\*/

Datum

utinyint\_to\_i2(PG\_FUNCTION\_ARGS)

{

utinyint arg1 = PG\_GETARG\_utinyint(0);

PG\_RETURN\_UINT16((uint16) arg1);

}

Datum

i2\_to\_utinyint(PG\_FUNCTION\_ARGS)

{

uint16 arg1 = PG\_GETARG\_UINT16(0);

if (arg1 < 0 || arg1 > UCHAR\_MAX)

ereport(ERROR,

(errcode(ERRCODE\_NUMERIC\_VALUE\_OUT\_OF\_RANGE),

errmsg("utinyint out of range")));

PG\_RETURN\_utinyint((utinyint) arg1);

}

Datum

utinyint\_to\_i4(PG\_FUNCTION\_ARGS)

{

utinyint arg1 = PG\_GETARG\_utinyint(0);

PG\_RETURN\_UINT32((uint32) arg1);

}

Datum

i4\_to\_utinyint(PG\_FUNCTION\_ARGS)

{

uint32 arg1 = PG\_GETARG\_UINT32(0);

if (arg1 < 0 || arg1 > UCHAR\_MAX)

ereport(ERROR,

(errcode(ERRCODE\_NUMERIC\_VALUE\_OUT\_OF\_RANGE),

errmsg("utinyint out of range")));

PG\_RETURN\_utinyint((utinyint) arg1);

}

Datum

utinyint\_to\_bool(PG\_FUNCTION\_ARGS)

{

if (PG\_GETARG\_utinyint(0) == 0)

PG\_RETURN\_BOOL(false);

else

PG\_RETURN\_BOOL(true);

}

Datum

bool\_to\_utinyint(PG\_FUNCTION\_ARGS)

{

if (PG\_GETARG\_BOOL(0) == false)

PG\_RETURN\_utinyint(0);

else

PG\_RETURN\_utinyint(1);

}

（3）utinyint.sql.in.c

CREATE FUNCTION utinyint\_in(cstring) RETURNS utinyint AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE FUNCTION utinyint\_out(utinyint) RETURNS cstring AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE FUNCTION utinyint\_recv(internal) RETURNS utinyint AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE FUNCTION utinyint\_send(utinyint) RETURNS bytea AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE TYPE utinyint (

INPUT = utinyint\_in,

OUTPUT = utinyint\_out,

RECEIVE = utinyint\_recv,

SEND = utinyint\_send,

INTERNALLENGTH = 1,

PASSEDBYVALUE,

ALIGNMENT = char,

STORAGE = plain

);

CREATE FUNCTION utinyint\_to\_i2(utinyint) RETURNS smallint AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE CAST (utinyint AS smallint)

WITH FUNCTION utinyint\_to\_i2(utinyint) AS IMPLICIT;

CREATE FUNCTION i2\_to\_utinyint(smallint) RETURNS utinyint AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE CAST (smallint AS utinyint)

WITH FUNCTION i2\_to\_utinyint(smallint) AS ASSIGNMENT;

CREATE FUNCTION utinyint\_to\_i4(utinyint) RETURNS integer AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE CAST (utinyint AS integer)

WITH FUNCTION utinyint\_to\_i4(utinyint) AS IMPLICIT;

CREATE FUNCTION i4\_to\_utinyint(integer) RETURNS utinyint AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE CAST (integer AS utinyint)

WITH FUNCTION i4\_to\_utinyint(integer) AS ASSIGNMENT;

CREATE FUNCTION utinyint\_to\_bool(utinyint) RETURNS bool AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE CAST (utinyint AS bool)

WITH FUNCTION utinyint\_to\_bool(utinyint) AS ASSIGNMENT;

CREATE FUNCTION bool\_to\_utinyint(bool) RETURNS utinyint AS

'MODULE\_PATHNAME'

LANGUAGE c IMMUTABLE STRICT;

CREATE CAST (bool AS utinyint)

WITH FUNCTION bool\_to\_utinyint(bool) AS ASSIGNMENT;

## 三、遇到的问题

### 3.1 刚开始编译扩展包的时候编译不成功

由于最开始是在eclipse下编译安装，如果再到终端编译安装扩展包则不成功，只有同时在终端编译才行。

### 3.2 climits (limits.h)

列出一些常量值：

name expresses min. magnitude\*

CHAR\_BIT Number of bits for a char object (byte) 8

SCHAR\_MIN Minimum value for an object of type signed char -127

SCHAR\_MAX Maximum value for an object of type signed char 127

UCHAR\_MAX Maximum value for an object of type unsigned char 255

CHAR\_MIN Minimum value for an object of type char either SCHAR\_MIN or 0

CHAR\_MAX Maximum value for an object of type char either SCHAR\_MAX or UCHAR\_MAX

MB\_LEN\_MAX Maximum number of bytes in a multibyte character, for any locale 1

SHRT\_MIN Minimum value for an object of type short int -32767

SHRT\_MAX Maximum value for an object of type short int 32767

USHRT\_MAX Maximum value for an object of type unsigned short int 65535

INT\_MIN Minimum value for an object of type int -32767

INT\_MAX Maximum value for an object of type int 32767

UINT\_MAX Maximum value for an object of type unsigned short int 65535

LONG\_MIN Minimum value for an object of type long int -2147483647

LONG\_MAX Maximum value for an object of type long int 2147483647

ULONG\_MAX Maximum value for an object of type unsigned int 4294967295

## 四、附加内容

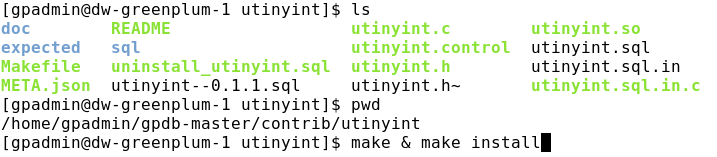
### 4.1 无符号tinyint扩展包的编译安装

（1）先关闭服务端



（2）make & make install

直接进入到contrib目录下找对应扩展包，进入扩展包文件进行编译安装；

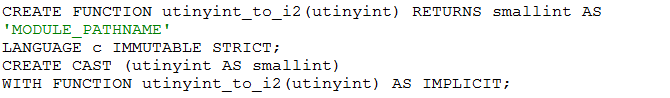


（3）重新开启服务端

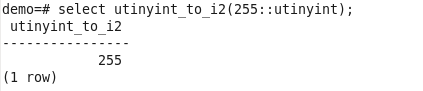


### 4.2 create function

创建函数，如：



调用函数：



### 4.3 安装扩展模块

只有在创建扩展之后才能使用扩展包的功能；



若不使用，也可将该扩展模块删除，但是需要谨慎；



如果已有表使用了该数据类型，则删除不成功。

## 五、总结与展望

### 5.1 总结

本次课程，我们小组通过组内学习合作，共同完成了这次的课程设计。对于该次组内合作，简略将本次收获分享。我们对于简单的学习内容，自己独立学习并查找资料，从而快速解决相应具体问题；而复杂的知识内容，则需要我们一起讨论摸索，所以合作内容的确定需要我们每一个组员慎重考虑。在平时实际运用过程中，总会出现一些问题，通过大家共同努力完成了这次课程设计。我们经验总结如下：

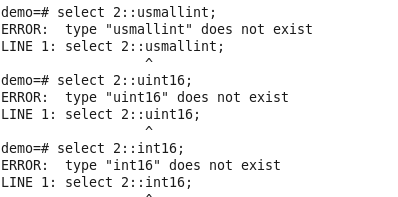
（1）通过此次修改源码，学习到扩展程序的使用，为以后学习提供了铺垫。

（2）熟练掌握了两种数据类型的转换需要修改的地方，转换之间的相同点和不同点。

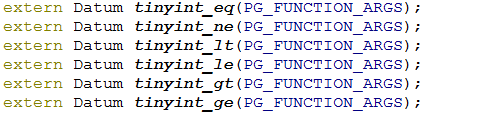
（3）对应GP的使用更熟练了。

### 5.2 下一步扩展

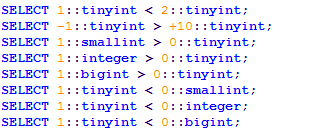
（1）smallint、integer、bigint转换成无符号的smallint、integer、bigint



（2）无符号tinyint和其他的类型的数据之间进行增删查改



执行sql：



（3）pg\_atoi()和pg\_itoa()的进一步完善

