set @a=1 a :=1;

类型转换：

`type1::type2(eg:  '12'::int)`

```
char(n)                    char(n)
varchar(n)               text
int                            int
bigint                       bigint
datetime                  timestamp
money                     numeric(19,4)
date                         date
double                     double
float                          float
numeric                    numeric    --经纬度
image                       bytea
bool                          bool
uniqueidentifier        uuid
varbinary                   bytea
timestamp                 interval
datetime                    current_timestamp(0)::timestamp;
                                  /now()::timestamp事务时间;
                      clock_timestamp()::timestamp;
                      ## 2017-08-24 09:44:19.507167
                      ## 不加timestamp会带时区
```

function函数转换:

```
isnull()              coalesce()
datalength()          octet_length()
len()                 length()
convert()             ::type  /(cast())
'str1'+'str2'         'str1'||'str2';
exists()              exists()
ltrim()/rtrim()       trim()
lower()               lower()
upper()               upper()
round(arg1,0)         round(arg1)        [round(2.22)==2]
floor(arg1)           floor(arg1)        [取<=arg1最小的整数]
ceil(arg1)            ceil(arg1)         [取>=arg1最小的整数]
substring('dfadf', 1, 2)[df]             substr()
```

```
charindex(';', 'fasd;fds', 2)            position(';' in 'fasd;fds')
getdate()            now();
dateadd()            now()+interval '3 months'; --如果数字是参数形式传入，需要使用类似：now() + make_interval(days => 1-i)
DATEDIFF(day, starttime, endtime)       date_part('day', endtime - starttime))
datepart()            date_part('week', now());
--获取到周
split_part('d1,d2,d3',',', 2)
```

数组与字符串之间的转换:

```
arg text [];
arg :=string_to_array('df,asd,fa', ',')

arg text;
arg :=array_to_string(array['dfa','dfas','fds'], '?')
    dfa?dfas?fds
postgres=# select t1, array_agg(t2) from t03 group by t1;
t1 | array_agg
----+------------
b  | {xxx,yyy}
a  | {10,20,30}
(2 rows)
postgres=# select t1, array_to_string(array_agg(t2), ',') from t03 group by t1;
t1 | array_to_string
----+-----------------
b  | xxx,yyy
a  | 10,20,30
(2 rows)
```

长度不足补'0'(lpad\rpad)

```
greenerp2018=> select lpad('9', 6, '0');
  lpad
--------
000009
(1 row)
```

异常捕获

```
begin
    statment
exception
    when **(others) then
        out_result:=0;
end;
```

select ** into ** from tab1;
IF NOT FOUND THEN
RAISE EXCEPTION 'No flight at %.', $1;
END IF;

临时表：

```
select * into #t12 from table01;
改成:
create temp table tmp_t12 on commit drop as select * from table01;

with … as…
```

影响的行数：

```
rowcount ==>GET DIAGNOSTICS v_count = ROW_COUNT;
while 1=1 loop --批量删除数据
    delete from t_inofaout_zcq  where trade_id=in_trade_id and
in_trade_id>0 and rq> in_start_date and rq<=in_end_date ;
    GET DIAGNOSTICS v_count = ROW_COUNT;
    if v_count<10000 then
        exit;            --while循环中退出
    end if;
end loop;
```

mssql中是用newid()函数生成uuid的，而PG中需要先装载模块：

```
create extension uuid-ossp
```

然后用uuid_generate_v1()生成uuid。

自增

`File_ID int NOT NULL IDENTITY(1,1)==> File_ID SERIAL NOT NULL`

`returning 赋值的变量需要返回`

`参数默认值 in aaa int default 1`

with用法:

```
with ntb as (select a,b,c from tab2) select b,c from ntb where ntb=1;
```

如果返回的数据根据判断条件不同返回不同的数据集，就不要在function参数中写out，returns record就行（返回相同类型的数据集时returns setof record），然后在不同的条件中return query

选择结构：

```
case when a=1 then b:=2
    when …
    when …
    else …
    end
```

存储过程中调用存储过程？

```
sqlserver中是exec，转为如果这个函数是返回单值的，就直接调用函数即可：v_info := f_getinfo('aaaaa');
如果只返回一行数据，应该也可以这样。
如果是返回多行的，用
for rec in select * from func_name('xxxx') loop
…
end loop;
```

returns tables(a text, b text)

返回单行多列的数据集，直接写out，return record就行，不需要setof;

```
create or replace function GetDate(
  in in_month int,
  out v_date1 date,
  out v_date2 date
  ) returns record
```

```
as $$
BEGIN
    v_date1 :=now();
    v_date2 :=now()+make_interval(months => in_month);
    return;
END;
$$ LANGUAGE plpgsql;
```

查看表字段类型：

```
select
table_schema,table_name,column_name,data_type,column_default,is_nullable
from information_schema.columns
where table_name = 'nginx_log';

```

当只返回单个字段时，returns type要和输出的数据的类型一致。

```
create or replace function GetDate2(
    in in_date varchar(23),
    in in_day_num int,
    out v_date1 varchar(30)
)returns varchar
as $$
begin
    v_date1 :=to_char(to_date(in_date, 'yyyy-MM-DD')+make_interval(days
=> in_day_num), 'yyyy-MM-DD');
end ;
$$
language 'plpgsql';
```

`SELECT to_number('21.44', '99D99')  换成 ::float`

调用单个返回值的函数，不要用select f_xxxx()，而要用v_result := f_xxxx();

游标替换：

```
CREATE or replace function pro_effuse_dtl_gw_update() returns void
as $$
declare
    rec record;
    arg int;
```

```
begin
    for rec in select effuse_detail_id, to_char(stock_date, 'yyyy-MM-DD') as stock_date
        from sell_effuse m, sell_effuse_detail d
        where m.effuse_id=d.effuse_id and m.sell_month='2017-07-15' and m.status=1
    loop
        arg :=pro_effuse_detail_update_dep(rec.effuse_detail_id);
        -- 函数调用时，有返回值的一定要用参数承接，不用select调用
    end loop;
end;
$$
language 'plpgsql';
```

```
select * from function1(arg1..) a, tab2 b where a.col1=b.col2;
```

如果不想处理函数的返回值可以使用perform关键字:

```
CREATE OR REPLACE FUNCTION f_test16(in in_id1 int)
RETURNS int
AS $$
DECLARE
rec record;
BEGIN
    perform f_test15(10);
    return 10;
END;
$$ LANGUAGE plpgsql;
```

查看执行过程，一般看看执行的过程时间：

```
EXPLAIN select * from tab;          --伪执行
EXPLAIN ANALYZE  select * from tab;
```

'pub_doc','pub_employee','pub_employee1' --这几张表涉及null、image信息需要使用psycopg2.Binary(content)将image类型字符串转义

键和索引：

```
添加主键：alter table sell_effuse add primary key (effuse_id);
添加外键：alter table sell_effuse_detail add constraint fk_sell_effuse_detail_foreign_effuseid
```

```
            foreign key(effuse_id) references sell_effuse(effuse_id);
删除外键：alter table sell_effuse_detail drop constraint fk_sell_effuse_detail_foreign_effuseid ;

pg创建索引
单字段索引：
CREATE INDEX index_name ON table_name (field1);
联合索引：
CREATE INDEX index_name ON table_name (field1,field2);
条件(部分)索引：
CREATE INDEX index_name ON table_name (field1) WHERE field1 > 100;
注意：想要使用条件索引，在使用select语句时，where条件最好加上field>100（最少100）的条件，否则索引很可能利用不到。

块范围索引
create index idx_t1_id on t1 using brin (id);  --默认情况下pages_per_range为128
brin索引的使用方法：create index idx_t03_id on t03 using brin(id) with(pages_per_range=256);

创建索引时，可以通过包含选项ASC, DESC, NULLS FIRST, 和/或 NULLS LAST调整B-tree索引的顺序；例如：
CREATE INDEX test2_info_nulls_low ON test2 (info NULLS FIRST);
CREATE INDEX test3_desc_index ON test3 (id DESC NULLS LAST);

数组字段创建索引，建议gin索引？

各类索引的使用场景和优缺点
CREATE TABLE test1 (
        id integer,
        content varchar
    );
普通索引
CREATE INDEX test1_id_index ON test1 (id);

hash
CREATE INDEX name ON table using hash (column);

唯一索引
CREATE UNIQUE INDEX name ON table (column [, …]);


表达式索引主要用于在查询条件中存在基于某个字段的函数或表达式的结果与其他值进行比较的情况，如：
SELECT * FROM test1 WHERE lower(col1) = 'value';
```

```
此时，如果我们仅仅是在col1字段上建立索引，那么该查询在执行时一定不会使用该索引，而是直接进行全表扫描。如果该表的数据量较大，那么执行该查询也将会需要很长时间。解决该问题的办法非常简单，在test1表上建立基于col1字段的表达式索引，如：
    CREATE INDEX test1_lower_col1_idx ON test1 (lower(col1));
```

修改函数的属主：

```
ALTER FUNCTION "public"."insert_user"(i_username text, i_emailaddress text) OWNER TO "inofa";
```

`to_char函数，第一个参数的类型必须是date类型，不能是varchar`

读写分离示例：

```
#代理proxydb上插数据的函数：
CREATE OR REPLACE FUNCTION insert_user(i_username text, i_emailaddress text)
RETURNS integer AS $$
    CLUSTER 'write_cluster';
    RUN ON ANY;
$$ LANGUAGE plproxy;

#代理proxydb上查询数据的函数
CREATE OR REPLACE FUNCTION get_user_email(i_username text)
RETURNS SETOF text AS $$
    CLUSTER 'read_cluster';
    RUN ON ANY;
$$ LANGUAGE plproxy;


CREATE or REPLACE FUNCTION sumtest(
    username text,
    out out_result INT,
    out em text,
    out im INT
)returns record
as $$
BEGIN
    BEGIN
        em :=get_user_email(username);
        im :=insert_user('xiaoming', em);
        out_result:=1;
    EXCEPTION
```

```
        when OTHERS THEN
            out_result:=0;
    end;
end;
$$
LANGUAGE 'plpgsql';
```

1. 下面是PostgreSQL中支持的时间/日期操作符的列表：

操作符	例子	结果
+	date '2001-09-28' + integer '7'	date '2001-10-05'
+	date '2001-09-28' + interval '1 hour'	timestamp '2001-09-28 01:00'
+	date '2001-09-28' + time '03:00'	timestamp '2001-09-28 03:00'
+	interval '1 day' + interval '1 hour'	interval '1 day 01:00'
+	timestamp '2001-09-28 01:00' + interval '23 hours'	timestamp '2001-09-29 00:00'
+	time '01:00' + interval '3 hours'	time '04:00'
-	- interval '23 hours'	interval '-23:00'
-	date '2001-10-01' - date '2001-09-28'	integer '3'
-	date '2001-10-01' - integer '7'	date '2001-09-24'
-	date '2001-09-28' - interval '1 hour'	timestamp '2001-09-27 23:00'
-	time '05:00' - time '03:00'	interval '02:00'
-	time '05:00' - interval '2 hours'	time '03:00'
-	timestamp '2001-09-28 23:00' - interval '23 hours'	timestamp '2001-09-28 00:00'
-	interval '1 day' - interval '1 hour'	interval '23:00'
-	timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00'	interval '1 day 15:00'
*	interval '1 hour' * double precision '3.5'	interval '03:30'
/	interval '1 hour' / double precision '1.5'	interval '00:40'

2. 日期/时间函数：

函数	返回类型	描述	例子	结果
age(timestamp, timestamp)	interval	减去参数，生成一个使用年、月的"符号化"的结果	age('2001-04-10', timestamp '1957-06-13')	43 years 9 mons 27 days
age(timestamp)	interval	从current_date减去得到的数值	age(timestamp '1957-06-13')	43 years 8 mons 3 days
current_date	date	今天的日期		
current_time	time	现在的时间		
current_timestamp	timestamp	日期和时间		
date_part(text, timestamp)	double	获取子域(等效于extract)	date_part('hour', timestamp '2001-02-16 20:38:40')	20
date_part(text, interval)	double	获取子域(等效于extract)	date_part('month', interval '2 years 3 months')	3
date_trunc(text, timestamp)	timestamp	截断成指定的精度	date_trunc('hour', timestamp '2001-02-16 20:38:40')	2001-02-16 20:00:00+00
extract(field from timestamp)	double	获取子域	extract(hour from timestamp '2001-02-16 20:38:40')	20
extract(field from interval)	double	获取子域	extract(month from interval '2 years 3 months')	3
localtime	time	今日的时间		
localtimesta	timest			

mp	amp	日期和时间		
now()	timestamp	当前的日期和时间(等效于 current_timestamp)		
timeofday()	text	当前日期和时间		

```
3. EXTRACT，date_part函数支持的field：
```

来源： http://blog.csdn.net/snn1410/article/details/7741283

域	描述	例子	结果
CENTURY	世纪	EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');	20
DAY	(月分)里的日期域(1-31)	EXTRACT(DAY from TIMESTAMP '2001-02-16 20:38:40');	16
DECADE	年份域除以10	EXTRACT(DECADE from TIMESTAMP '2001-02-16 20:38:40');	200
DOW	每周的星期号(0-6；星期天是0) (仅用于timestamp)	EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');	5
DOY	一年的第几天(1 -365/366) (仅用于 timestamp)	EXTRACT(DOY from TIMESTAMP '2001-02-16 20:38:40');	47
HOUR	小时域(0-23)	EXTRACT(HOUR from TIMESTAMP '2001-02-16 20:38:40');	20
MICROSECONDS	秒域，包括小数部分，乘以 1,000,000。	EXTRACT(MICROSECONDS from TIME '17:12:28.5');	28500000
MILLENNIUM	千年	EXTRACT(MILLENNIUM from TIMESTAMP '2001-02-16 20:38:40');	3
MILLISECONDS	秒域，包括小数部分，乘以 1000。	EXTRACT(MILLISECONDS from TIME '17:12:28.5');	28500
MINUTE	分钟域(0-59)	EXTRACT(MINUTE from TIMESTAMP '2001-02-16 20:38:40');	38

MONTH	对于timestamp数值，它是一年里的月份数(1-12)；对于interval数值，它是月的数目，然后对12取模(0-11)	EXTRACT(MONTH from TIMESTAMP '2001-02-16 20:38:40');	2
QUARTER	该天所在的该年的季度(1-4)(仅用于 timestamp)	EXTRACT(QUARTER from TIMESTAMP '2001-02-16 20:38:40');	1
SECOND	秒域，包括小数部分(0-59[1])	EXTRACT(SECOND from TIMESTAMP '2001-02-16 20:38:40');	40
WEEK	该天在所在的年份里是第几周。	EXTRACT(WEEK from TIMESTAMP '2001-02-16 20:38:40');	7
YEAR	年份域	EXTRACT(YEAR from TIMESTAMP '2001-02-16 20:38:40');	2001