Create A Function

standard from

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [,
...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | PARALLEL { UNSAFE | RESTRICTED | SAFE }
    | COST execution_cost
    | ROWS result_rows
    | SUPPORT support_function
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
```

`CREATE FUNCTION` defines a new function. `CREATE OR REPLACE FUNCTION` will either create a new function, or replace an existing definition.

If a schema name is included, then the function is created in the specified schema. Otherwise it is **created in the current schema**. The name of the new function must not match any existing function or procedure with the same input argument types in the same schema. However, functions and procedures of different argument types can share a name (this is called **overloading**).

To replace the current definition of an existing function, use `CREATE OR REPLACE FUNCTION`. It is **not possible to change the name or argument types of a function this way** (if you tried, you would actually be creating a new, distinct function). Also, `CREATE OR REPLACE FUNCTION` will **not let you change the return type of an existing function**. To do that, you must drop and recreate the function. (When using `OUT` parameters, that means you cannot change the types of any `OUT` parameters except by dropping the function.)

When `CREATE OR REPLACE FUNCTION` is used to replace an existing function, the ownership and permissions of the function do not change.

If you drop and then recreate a function, the new function is **not the same entity as the old**; you will have to drop existing rules, views, triggers, etc. that refer to the old function. What's more, you can use `ALTER FUNCTION` to change most of the auxiliary properties of an existing function.

The user that creates the function becomes the owner of the function.

To be able to create a function, you must have `USAGE` privilege on the argument types and the return type.

Four kinds of function

- Query language function (written in SQL)
- Procedure language function

  pgSQL/Tcl/Perl/Python
- Internal function
- C-language function

Function of SQL

SQL Functions on Base Types

Experiment 1:

```sql
--No params, and return a basic form
CREATE FUNCTION one() RETURNS integer AS $$
    SELECT 1 AS result;
$$ LANGUAGE SQL;

-- Alternative syntax for string literal:
CREATE OR REPLACE FUNCTION one() RETURNS integer AS '
    SELECT 1 AS result;
' LANGUAGE SQL;

SELECT one();

SELECT one() AS answer;
```

Experiment 2:

```sql
--Simple input params
CREATE FUNCTION add_em(x integer, y integer) RETURNS integer AS $$
    SELECT x + y;
$$ LANGUAGE SQL;

SELECT add_em(1, 2) AS answer;

CREATE FUNCTION add_em2(integer, integer) RETURNS integer AS $$
    SELECT $1 + $2;
$$ LANGUAGE SQL;

SELECT add_em2(1, 2) AS answer;
```

SQL Functions on Composite Types

Experiment 3:

```sql
CREATE TABLE emp_new (
    id          integer primary key ,
    name        varchar(30),
    age         integer,
    dept_id     integer,
    salary      numeric default 0.0
);

INSERT INTO emp_new values (001, 'Zhang San',23,1,5000);

CREATE FUNCTION double_salary(emp_new) RETURNS numeric AS $$
    SELECT $1.salary * 2 AS salary;
$$ LANGUAGE SQL;

SELECT name, double_salary(emp_new.*) AS dream
    FROM emp_new
    WHERE emp_new.id = 001;
```

SQL Functions with Output Parameters

Experiment 4:

```sql
--sub1
CREATE FUNCTION add_em (IN x int, IN y int, OUT sum int)
AS 'SELECT x + y'
LANGUAGE SQL;

SELECT add_em(3,7);

--sub2
CREATE FUNCTION sum_n_product (x int, y int, OUT sum int, OUT product int)
AS 'SELECT x + y, x * y'
LANGUAGE SQL;

SELECT * FROM sum_n_product(11,42);
```

SQL Functions with Default Values for Arguments

Experiment 5:

```sql
CREATE FUNCTION test_default(a int, b int DEFAULT 2, c int DEFAULT 3)
RETURNS int
LANGUAGE SQL
AS $$
    SELECT $1 + $2 + $3;
$$;

SELECT test_default(10, 20, 30);
SELECT test_default(10, 20);
SELECT test_default(10);
SELECT test_default();  error
```

Tips: Functions that have different argument type lists will **not be** considered to **conflict at creation time**, but if defaults are provided they might **conflict in use**.

PostgreSQL function

Basic Form

```sql
create or replace function function_name(param_name, param_type)
returns return_type
language plpgsql
as$$
declare
variable_name variable_type:=initial value
......
begin
end;
$$
language plpgsql;
```

Experiment 6:

```sql
--Create a function to calculate the sum of two integer numbers. After your
design, you can execute the following query.
create or replace function sum_func(a int, b int)
returns int
language plpgsql
as $function$
begin
return a + b;
end;
$function$;

select sum_func(3,4);
```

Experiment 7:

```
-- Create a function named "fullname", which has two variables called "firstname"
and "secondname" and return the combination of two variables. After your design,
you can execute following queries.
create or replace function fullname(firstname varchar, secondname varchar)
    returns varchar
    language plpgsql
as
$function$
declare
    name varchar := null;
begin
    name := firstname || ' ' || secondname;
    return name;
end;
$function$;

select fullname('Zhang','San');
```

Conditions in Function (using database film.db)

Basic Form

```
begin
  if condition1
  then
    ...
  elseif condition2
  then
    ...
  else
    ...
  end if;
end;
```

Experiment 8:

```
-- Create a function to combine firstname and surname of people according to the
people coming from eastern country or western country
create function full_name(p_fn varchar, p_sn varchar, style char)
    returns varchar
as
$$
begin
    if upper(style) = 'W'
    then
        return trim(coalesce(p_fn, '') || ' ' || p_sn);
    elseif upper(style) = 'E'
    then
        return trim(p_sn || ' ' || coalesce(p_fn, ''));
```

```
    else
        raise exception 'Style must be W or E';
    end if;
end;
$$
    language plpgsql;

--test
select full_name(p.first_name, p.surname, 'E')
from people p
        join credits c on p.peopleid = c.peopleid and c.credited_as = 'D'
        join movies m on m.movieid = c.movieid
where m.country = 'cn';
```

Loop in Function

Basic Form

```
for variable_value in start_value..end_value loop
   statements;
end loop;

while condition loop
   statements;
end loop;
```

Experiment 9:

```
-- Find the factorial of number
create or replace function factorial(number int)
    returns int
    language plpgsql
as
$function$
declare
    result int;
begin
    result = 1;
    for i in 1 .. number
        loop
            result = result * i;
        end loop;
    return result;
end;
$function$;

--or
create or replace function factorial2(number int)
    returns int
    language plpgsql
as
$function$
```

```
declare
    result int;
    i       int;
begin
    result = 1;
    i = 1;
    while i <= number
        loop
            result = result * i;
            i = i + 1;
        end loop;
    return result;
end;
$function$;

select factorial(5);
select factorial2(5);
```

Return A table from Function

```
create function fun_name(arg1 type1, ......)
    returns
        table
        (
            col_name1 col_type,
            col_name2 col_type, ..
            ..
            ..
        )
as
$$
begin
    return query select col1, col2 from......;
end;
$$
    language plpgsql;
```

The column type of result set should be same as the type of return table exactly, more specifically, the type of col1 should be same as the first col_type, and the type of col2 should be same as the second col_type.

Experiment 10:

```
--Design a function to return a table that contains all characters and their
ascii code from a pattern string in ascending order of ascii code.
create function character_table(pattern varchar)
    returns table
            (
                chr    char,
                ascii int
            )
as
```

```
    $body$
    begin
        return query
            select distinct cast(substr(t1.title, t2, 1) as char) chr,
                             ascii(substring(t1.title, t2, 1))     ascii
            from (select pattern as title) t1
                    cross join generate_series(1, length(pattern)) t2
            order by ascii;
    end;
    $body$
        LANGUAGE plpgsql;

    --test
    select * from character_table('I love database!');
```

Comprehensive Example (Provide by Stephane Faroult)

Writing a function that recognizes in which script a text is written (to be applied to column TITLE in table ALT_TITLES). We'll only consider the main writing systems. In particular the table at List of writing scripts by adoption, with the number of users.

If you execute the query:

```
select script (title ),title from (select title from alt_titles) x
```

The result would be

	script	title
37	Latin	All's Well, Ends Well 1997
38	Latin	99 Francs
39	Latin	Days of Being Wild
...	...	...

Hints:

Ranges to consider for the ascii() return value (approximate blocks but the result should be OK -

can be refined if needed)

Latin

<=740

[7424,8594]

[11360,11391]

[42786,43876]

Greek

[880, 1023]

[7462, 8446]Cyrillic

[1024, 1327]

[7296, 7544]

[42560, 42655]

Arabic

[1536, 2303]

[64336, 69246]

[124464, 126705]

Indian

[2304, 3572]

Thai

[3585, 3675]

Burmese

[4096, 4255]

Korean

[4352, 4607]

[12593, 12686]

[12800, 12926]

[43360, 55203]

[43360, 55291]

[65440, 65500]

Khmer

[6016, 6137]

Chinese

[11904, 12333]

[12344, 12347]

[13312, 42182]

Japanese

[12353, 12543]

[12784, 12799]

[13008, 13143]

Other

 everything else ...

Solution1:

```
--solution1
```

```
create function script(fm character varying)
    returns character varying
    language plpgsql
as
$$
declare
    ascimax int;
    ascimin int;
    value   varchar;
begin
    select max(x.a)
    into ascimax
    from (
            select distinct ascii(substr(t.title, n, 1)) a,
                            substr(t.title, n, 1)
            from (select fm as title) t
                    cross join generate_series(1, length(t.title)) n) x;
    select min(x.a)
    into ascimin
    from (
            select distinct ascii(substr(t.title, n, 1)) a,
                            substr(t.title, n, 1)
            from (select fm as title) t
                    cross join generate_series(1, length(t.title)) n) x
    where a > 127;
    if ((ascimax <= 740 and ascimax >= 0) or (ascimax >= 7424 and ascimax <=
                                                                    8594) or
        (ascimax >= 11360 and ascimax <= 11391) or (ascimax >= 42786 and ascimax
            <= 43876))
    then
        value = 'Latin';
    elseif ((ascimin >= 880 and ascimin <= 1023) or (ascimin >= 7462 and ascimin
        <= 8446))
    then
        value = 'Greek';
    elseif ((ascimin >= 1024 and ascimin <= 1327) or (ascimin >= 7296 and
ascimin
        <= 7544) or
            (ascimin >= 42560 and ascimin <= 42655))
    then
        value = 'Cyrillic';
    elseif ((ascimin >= 1536 and ascimin <= 2303) or (ascimin >= 64336 and
                                                        ascimin <= 69246) or
            (ascimin >= 124464 and ascimin <= 126705))
    then
        value = 'Arabic';
    elseif (ascimin >= 2304 and ascimin <= 3572)
    then
        value = 'Indian';
    elseif (ascimin >= 3585 and ascimin <= 3675)
    then
        value = 'Thai';
    elseif (ascimin >= 4096 and ascimin <= 4255)
    then
        value = 'Burmese';
    elseif ((ascimin >= 4352 and ascimin <= 4607) or (ascimin >= 12593 and
                                                        ascimin <= 12686) or
            (ascimin >= 12800 and ascimin <= 12926) or (ascimin >= 43360 and
```

```
                                                    ascimin <= 55203) or
            (ascimin >= 43360 and ascimin <= 55291) or (ascimin >= 65440 and
                                                    ascimin <= 65500))
    then
        value = 'Korean';
    elseif (ascimin >= 6016 and ascimin <= 6137)
    then
        value = 'Khmer';
    elseif ((ascimin >= 11904 and ascimin <= 12333) or (ascimin >= 12344 and
                                                    ascimin <= 12347) or
            (ascimin >= 13312 and ascimin <= 42182))
    then
        value = 'Chinese';
    elseif ((ascimin >= 12353 and ascimin <= 12543) or (ascimin >= 12784 and
                                                    ascimin <= 12799) or
            (ascimin >= 13008 and ascimin <= 13143))
    then
        value = 'Japanese';
    else
        value = 'Other';
    end if;
    return value;
end;
$$;


-- or
--Solution2:
create or replace function script2(title varchar)
    returns varchar as
$script$
declare
    max_ascii               int;
    declare min_ascii_gt127 int;
begin
    -- get the greatest code point in title
    select max(char_ascii)
    into max_ascii
    from (select ascii(chars) char_ascii
          from unnest(string_to_array(title, null)) chars) char_asciis;
    -- get the smallest code point that over 127 in title
    select min(char_ascii)
    into min_ascii_gt127
    from (select ascii(chars) char_ascii
          from unnest(string_to_array(title, null)) chars) char_asciis
    where char_ascii > 127;
    -- if the greatest code point is Latin, then the string uses the Latin
script
    if max_ascii <= 740
        or max_ascii between 7424 and 8594
        or max_ascii between 11360 and 11391
        or max_ascii between 42786 and 43876
    then
        return 'Latin';
    else
        -- otherwise that the smallest code point over 127 in the string
probably defines the script
        return case
```

```
                   when min_ascii_gt127 <= 740
                       or min_ascii_gt127 between 7424 and 8594
                       or min_ascii_gt127 between 11360 and 11391
                       or min_ascii_gt127 between 42786 and 43876
                       then 'Latin'
                   when min_ascii_gt127 between 880 and 1023
                       or min_ascii_gt127 between 7462 and 8446
                       then 'Greek'
                   when min_ascii_gt127 between 1024 and 1327
                       or min_ascii_gt127 between 7296 and 7544
                       or min_ascii_gt127 between 42560 and 42655
                       then 'Cyrillic'
                   when min_ascii_gt127 between 1536 and 2303
                       or min_ascii_gt127 between 64336 and 69246
                       or min_ascii_gt127 between 124464 and 126705
                       then 'Arabic'
                   when min_ascii_gt127 between 2304 and 3572
                       then 'Indian'
                   when min_ascii_gt127 between 3585 and 3675
                       then 'Thai'
                   when min_ascii_gt127 between 4096 and 4255
                       then 'Burmese'
                   when min_ascii_gt127 between 4352 and 4607
                       or min_ascii_gt127 between 12593 and 12686
                       or min_ascii_gt127 between 12800 and 12926
                       or min_ascii_gt127 between 43360 and 55203
                       or min_ascii_gt127 between 43360 and 55291
                       or min_ascii_gt127 between 65440 and 65500
                       then 'Korean'
                   when min_ascii_gt127 between 6016 and 6137
                       then 'Khmer'
                   when min_ascii_gt127 between 11904 and 12333
                       or min_ascii_gt127 between 12344 and 12347
                       or min_ascii_gt127 between 13312 and 42182
                       then 'Chinese'
                   when min_ascii_gt127 between 12353 and 12543
                       or min_ascii_gt127 between 12784 and 12799
                       or min_ascii_gt127 between 13008 and 13143
                       then 'Japanese'
                   else 'Other'
               end;
        end if;
end;
$script$
    language plpgsql;
```

Function Overload

```
--function overload
create function test_overload(int, int) RETURNS int
LANGUAGE SQL
AS $$
    SELECT $1 + $2 ;
$$;
select test_overload(1,2);
```

```sql
create function test_overload(x int, y int default 3) RETURNS int
LANGUAGE SQL
AS $$
    SELECT $1 + $2 ;
$$;

create function test_overload(int, int) RETURNS text
LANGUAGE SQL
AS $$
    SELECT $1 + $2 ;
$$;

create function test_overload(x int, y int, z varchar(10)) RETURNS text
LANGUAGE SQL
AS $$
    SELECT $1 + $2||z ;
$$;
select test_overload(1,2,' kids');
```

Tips: What funcs created by myself?

```sql
SELECT pg_proc.proname     AS "函数名称",
       pg_proc.prorettype AS "返回值数据类型",
       pg_proc.pronargs   AS "参数个数"
FROM pg_proc
where pg_proc.pronamespace = (SELECT pg_namespace.oid FROM pg_namespace WHERE
nspname = 'public');
```