

Principles of Database Systems (CS307)

Lab Session: Function

Yuxin Ma

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

Function

Built-in Functions

- Most DBMS provides a series of built-in functions
 - E.g., Scalar function, aggregation function, window function



```
round(3.141592, 3)  -- 3.142
trunc(3.141592, 3)  -- 3.141
```



```
upper('Citizen Kane')
lower('Citizen Kane')
substr('Citizen Kane', 5, 3)  -- 'zen'
trim('  Oops  ')  -- 'Oops'
replace('Sheep', 'ee', 'i')  -- 'Ship'
```

```
count(*)/count(col), min(col), max(col), stddev(col), avg(col)
```

```
<function> over (partition by <col_p> order by <col_o1, col_o2, ...>)
```

- <function>: we can apply (1) ranking window functions, or (2) aggregation functions
- **partition by**: specify the column for grouping
- **order by**: specify the column(s) for ordering in each group

Self-defined Function

- Sometimes the built-in functions cannot fulfill our requirements
 - And the power of declarative language (SQL) is not enough
- Most DBMS implement a built-in, SQL-based programming language
 - A procedural extension to SQL

Procedural vs. Declarative

- Two different programming paradigms
 - **Imperative**: Describe the algorithms step-by-step (How to do)
 - **Procedural**: C (and many other legacy languages)
 - Object-oriented: Java
 - **Declarative**: Describe the result without specifying the detailed steps (What to do)
 - **(Pure) declarative**: SQL, Regular Expressions, Markup (HTML, XML), CSS
 - Functional: Scheme, Haskell, Scala, Erlang
 - Logic programming: Prolog

Procedural vs. Declarative

- E.g., How can we get a cup of tea?
 - In a procedural way:

-
- In a declarative way:

Procedural vs. Declarative

- E.g., How can we get a cup of tea?

- In a procedural way:

1. Get a cup
2. Get some tea
3. Get some hot water
4. Put tea into the cup
5. Pour hot water into the cup
6. return tea;



- In a declarative way:

Procedural vs. Declarative

- E.g., How can we get a cup of tea?

- In a procedural way:

1. Get a cup
2. Get some tea
3. Get some hot water
4. Put tea into the cup
5. Pour hot water into the cup
6. return tea;



- In a declarative way:

<a cup of tea/>

- You don't really need to know how to make a cup of tea
 - The system can do it in a black-box manner



大佬喝茶

Procedural vs. Declarative

- E.g., Find all Chinese movies before 1990 in the movies table?

- In a procedural way:

1. Read the movies table into the memory
2. For each row *i* in the table, repeat:
 - 2.1 In row *i*, read the value of the column “country”
 - 2.2 if ...

-
- In a declarative way:






```
select * from movies where country = 'cn' and year_released < 1990
```

- You don't really need to know how to filter the table
 - The DBMS system can do it in a black-box manner

Procedural vs. Declarative

- Benefits in declarative languages
 - No need to understand the details
 - The systems take in charge of all the details
 - Easier to use than imperative programming
 - More user-friendly
- Problem in declarative languages
 - Cannot specify the control flow of a program
 - “If there is no such command as <a cup of tea/>, you need to create it by yourself”

Procedural Extension to SQL

- Many DBMS products provide a **proprietary procedural extension** to the standard SQL
 - Transact-SQL (T-SQL)  Microsoft SQL Server
 - PL/SQL  ORACLE
 - PL/PGSQL  PostgreSQL
 - (No specific name)  MySQL
 - (Not supported)  SQLite

... well, sometimes SQLite is even not considered a DBMS

Function in (Postgre)SQL

- Example: Display the full name for people with “von”
 - When introducing **update**, we have modified the names starting with “von” into “... (von)” for ordering

	peopleid	first_name	surname	born	died	gender
1	16439	Axel	Ambesser (von)	1910	1988	M
2	16440	Daniel	Bargen (von)	1950	2015	M
3	16441	Eduard	Borsody (von)	1898	1970	M
4	16442	Suzanne	Borsody (von)	1957	<null>	F
5	16443	Tomas	Brömssen (von)	1943	<null>	M
6	16444	Erik	Detten (von)	1982	<null>	M
7	16445	Theodore	Eltz (von)	1893	1964	M
8	16446	Gunther	Fritsch (von)	1906	1988	M
9	16447	Katja	Garnier (von)	1966	<null>	F
10	16448	Harry	Meter (von)	1871	1956	M
11	16449	Jenna	Oy (von)	1977	<null>	F
12	16450	Alicia	Rittberg (von)	1993	<null>	F
13	16451	Daisy	Scherler Mayer (von)	1966	<null>	F
14	16452	Gustav	Seyffertitz (von)	1862	1943	M

Function in (Postgre)SQL

- If we simply concatenate the first name and the last name, it looks like this:
 - A little bit weird format (a trailing “von”)



```
select first_name || ' ' || surname
from people
where surname like '%(von)';
```

	?column?
1	Axel Ambesser (von)
2	Daniel Bargaen (von)
3	Eduard Borsody (von)
4	Suzanne Borsody (von)
5	Tomas Brömssen (von)
6	Erik Detten (von)
7	Theodore Eltz (von)
8	Gunther Fritsch (von)
9	Katja Garnier (von)
10	Harry Meter (von)
11	Jenna Oÿ (von)
12	Alicia Rittberg (von)
13	Daisy Scherler Mayer (von)
14	Gustav Seyffertitz (von)

Function in (Postgre)SQL

- Question: How can we restore the format into “first_name von surname”?
 - String operations

Function in (Postgre)SQL

- Question: How can we restore the format into “first_name von surname”?
 - String operations

```
select case
  when first_name is null then ''
  else first_name || ' '
end || case position('(' in surname)
  when 0 then surname
  else trim(')' from substr(surname, position('(' in surname) + 1))
      || ' '
      || trim(substr(surname, 1, position('(' in surname) - 1))
end
from people
where surname not like '%(von)';
```

Function in (Postgre)SQL

- Question: How can we restore the format into “first_name von surname”?
 - String operations

Then, how can we store this part to reuse it in the future?

```
case
  when first_name is null then ''
  else first_name || ' '
end || case position('(' in surname)
  when 0 then surname
  else trim(')' from substr(surname, position('(' in surname) + 1))
      || ' '
      || trim(substr(surname, 1, position('(' in surname) - 1))
end
from people
where surname not like '%(von)';
```


Function in (Postgre)SQL

- “Copy and paste” is not a good habit
 - Whenever you have painfully written something as complicated, which is pretty generic, you'd rather not copy and paste the code every time you need it

```
case
  when first_name is null then ''
  else first_name || ' '
end || case position('(' in surname)
  when 0 then surname
  else trim(')' from substr(surname, position('(' in surname) + 1))
      || ' '
      || trim(substr(surname, 1, position('(' in surname) - 1))
end
```



Function in (Postgre)SQL

- Store for Reuse
 - In PostgreSQL, we can store the expression and reuse it in another context
- Self-defined Function
 - **create function**



```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype AS $variable_name$
    DECLARE
        declaration;
    [...]
    BEGIN
        < function_body >
    [...]
    RETURN { variable_name | value }
END; LANGUAGE plpgsql;
```



```
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'
  | sql_body
} ...
```

...or, a simpler version

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

Function name and the parameter list

- Format for variables and parameters: [name] [type]

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar Return type
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' (' || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function in (Postgre)SQL

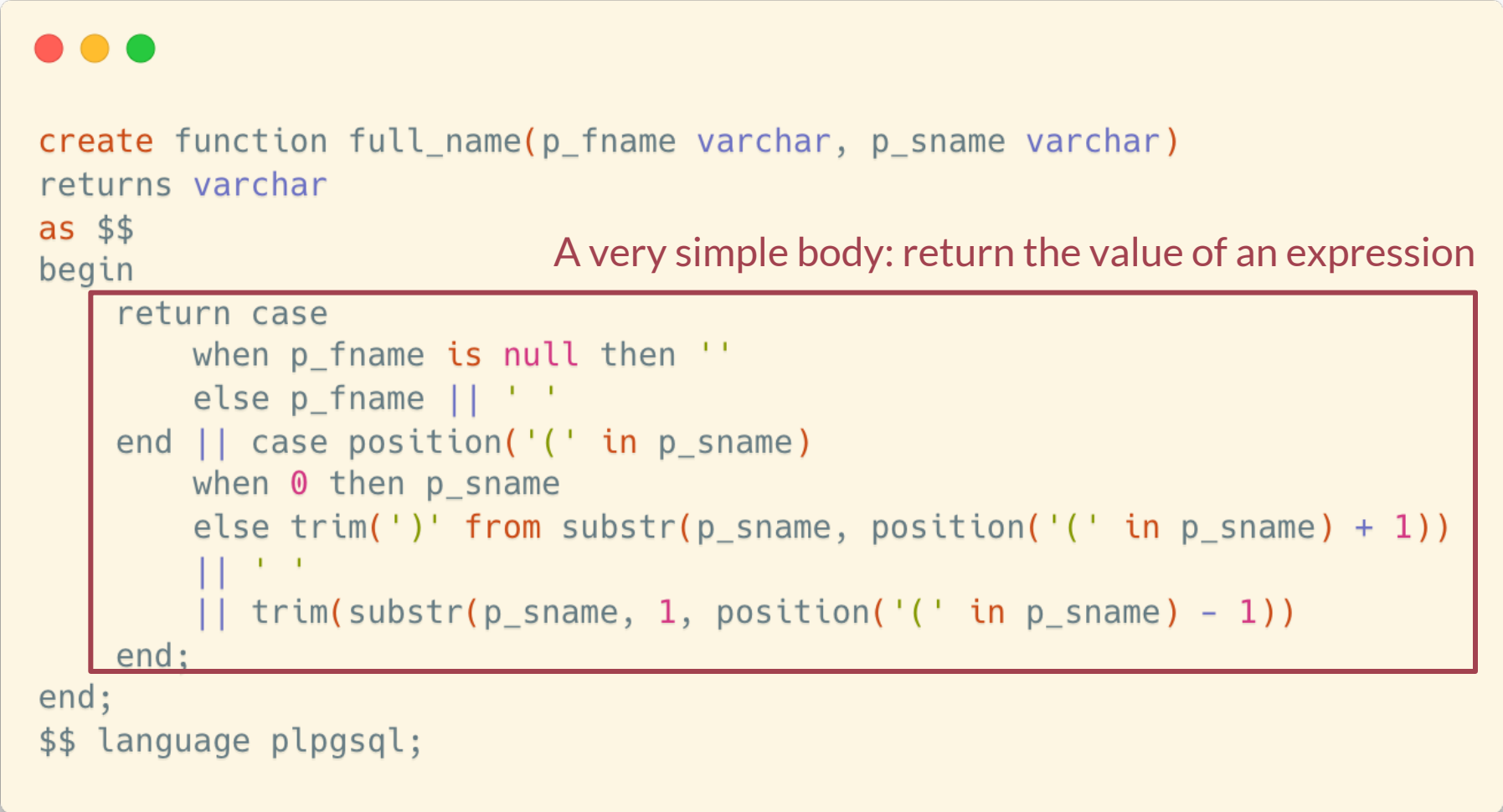
- How do we rewrite the name conversion expression into a function?

Body

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' (' || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?



```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' (' || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

A very simple body: return the value of an expression

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
```

A very simple body: return the value of an expression

```
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position
            || ' '
            || trim(substr(p_sname, 1, position('(' in
        end;
end;
$$ language plpgsql;
```

Procedural extensions provide all the bells and whistles in a true (procedural) programming languages, such as:

- Variables
- Conditions
- Loops
- Arrays
- Error management
- ...

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' (' || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Language Type

PostgreSQL supports 4 procedural languages:

PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python

- Tcl, Perl, and Python are famous scripting languages in case you don't know

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_
returns varchar
as $$
begin
    return case
        when p_fname is null
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim('(' from substr(sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Yes, we can even use Python to write functions

```
create function append_test(p_code varchar)
returns varchar
as $$
    if p_code == 'cn':
        return 'China'
    else:
        return 'not China'
$$ language plpython3u;
```

Language Type

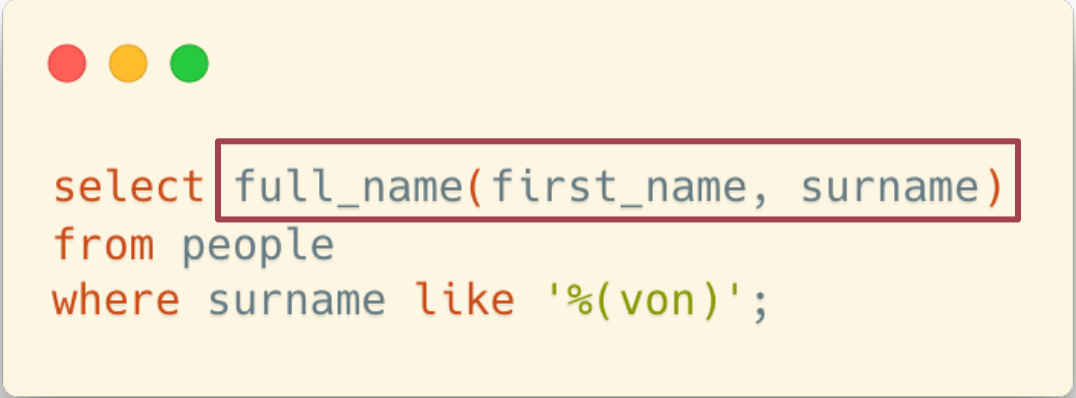
PostgreSQL supports 4 procedural languages:

PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python

- Tcl, Perl, and Python are famous scripting languages in case you don't know

Function in (Postgre)SQL

- Once your function is created, you can use it as if it were any built-in function.



```
select full_name(first_name, surname)
from people
where surname like '%(von)';
```

Function in (Postgre)SQL

- We can run `select` queries in functions
 - Example: design a function “get_country_name” to transform the country codes into country names based on the `countries` table

Function in (Postgre)SQL

- We can run **select** queries in functions
 - Example: design a function “get_country_name” to transform the country codes into country names based on the **countries** table

```
● ● ●  
  
create function get_country_name(p_code varchar)  
returns countries.country_name%type  
as $$  
declare  
    v_name countries.country_name%type;  
begin  
    select country_name  
    into v_name  
    from countries  
    where country_code = p_code;  
    return v_name;  
end;  
$$ language plpgsql;
```

```
● ● ●  
  
select get_country_name(country) from movies;
```

Function in (Postgre)SQL

- We can run **select** queries in functions
 - Example: design a function “get_country_name” to transform the country codes into country names based on the **countries** table

```
● ● ●  
  
create function get_country_name(p_code varchar)  
returns countries.country_name%type  
as $$  
declare  
    v_name countries.country_name%type;  
begin  
    select country_name  
    into v_name  
    from countries  
    where country_code = p_code;  
    return v_name;  
end;  
$$ language plpgsql;
```

```
● ● ●  
  
select get_country_name(country) from movies;
```

... seems to be an easy way to get rid of join operations?

```
● ● ●  
  
select c.country_name  
from countries c join movies m  
on c.country_code = m.country;
```

Function in (Postgre)SQL

- “Cultural Mismatch”
 - Here we have a problem, because there is a **big cultural gap** between the **relational mindset** and **procedural processing**.
 - A “look-up function” forces a “one row at a time” join which in most cases will be dreadful



```
select get_country_name(country) from movies;
```

For each row in movies, the select query in `get_country_name()` is executed once

Comment on Procedural SQL (PL/SQL)

- **Tom Kyte**, who is a Senior Technology Architect at Oracle, says that his mantra is:
 - You should do it in a single SQL statement if at all possible.
 - If you cannot do it in a single SQL statement, then do it in PL/SQL (as little PL/SQL as possible!)
- And some other suggestions (from your lecturer):
 - You should ask for help from someone more experienced than you
 - Stackoverflow, forums, Google, etc.



More to Read

- We may not cover all the details in functions in the theoretical session, so here are some more materials on procedural programming in PostgreSQL:
 - Lab tutorial on Functions
 - Please read it before your next lab sessions
 - Chapter 5.2 “Functions and Procedures,” Database System Concepts (7th Edition)
 - Chapter 43 “PL/pgSQL,” PostgreSQL Documentation
 - <https://www.postgresql.org/docs/current/plpgsql.html>