WEEK 06
ADVANCED SQL

Instructor: Yanan Wu
TA: Vanchy Li

Spring 2025

WEEK 06

LECTURE SESSION

Instructor: Yanan Wu
TA: Vanchy Li

Spring 2025

6.1
FUNCTION

FUNCTION

- **What is a Function in SQL?**
  - A **function** in SQL is a reusable block of code that performs a specific task.
  - Functions **accept input parameters**, process data, and **return a value or a table**.
  - Helps **simplify queries, automate calculations, and improve maintainability**

- **Key Components of an SQL Function**
  - CREATE FUNCTION **function_name(parameters)** → Defines the function.
  - RETURNS data_type / **RETURNS TABLE** → Specifies the function's return type.
  - AS **$$ ... $$** → Defines the function body inside dollar quotes.
  - **BEGIN ... END**; → Contains function logic.
  - **RETURN** → Specifies what the function should return.
  - **LANGUAGE plpgsql** → Indicates the procedural SQL language used (PostgreSQL).

plpgsql (Procedure Language / PostgreSQL)

- plpgsql (Procedure Language / PostgreSQL)
  - Procedural Language: Extends SQL with control structures like loops, conditions, and variables.
  - Use Case: Used for writing functions, triggers, and procedural logic inside PostgreSQL.
  - Features:
    a) Variables and control flow (IF, LOOP, FOR, WHILE)
    b) Exception handling
    c) Stored procedures and triggers

Key Difference between plpgsql and SQL

Feature	SQL	PL/pgSQL
Type	Declarative	Procedural
Purpose	Query and manipulate data	Create functions, procedures, triggers
Control Flow (IF, LOOP)	Not available	Available
Variables	Not available	Available
Error Handling	Minimal	Exception handling supported
Use Case	Simple queries, DDL/DML operations	Complex business logic, stored procedures

FUNCTION-EXAMPLE 01 – SUBSTRING()

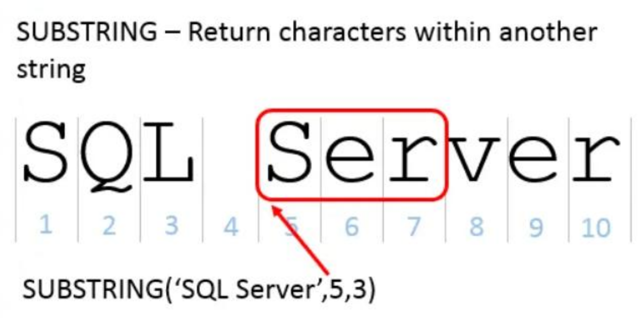
Extract a substring from a given string, starting at a specified position and spanning a specified length.

```
CREATE OR REPLACE FUNCTION func_name()
returns <return_datatype>
as
$$
begin
        <type in function body here>
end;
$$
language plpgsql;


select fn_mid()
```

```
CREATE OR REPLACE FUNCTION fn_mid()
returns varchar
as
$$
begin
        return substring();
end;
$$
language plpgsql;


select fn_mid()
```



SUBSTRING – Return characters within another string

SUBSTRING('SQL Server',5,3)

FUNCTION - ALIAS

- The DECLARE ... ALIAS FOR $n syntax is used in PL/pgSQL (Procedural Language for PostgreSQL) to assign meaningful names to function parameters.

- This is particularly useful when working with positional parameters ($1, $2, $3), as it makes the code more readable and maintainable.

FUNCTION EXAMPLE 02 - ALIAS

Using Alias to create a temporary name that assigned to a **parameter, column, or table** within a function to improve readability and prevent conflicts.

```
CREATE OR REPLACE FUNCTION fn_mid(varchar, integer, integer)
returns varchar
as
$$

begin
  return substring(word, startPos, endPoss);
end;
$$
language plpgsql;
```

FUNCTION – PARAMETER NAMES AND DATA TYPES

- Function parameters can be declared with **explicit names and data types**
  - Explicit Data Types
    a) VARCHAR → Defines word as a text-based input.
    b) INTEGER → Defines startPos and endPos as integer inputs.
    c) Ensuring proper data types helps prevent unexpected errors.

FUNCTION EXAMPLE 03 - Parameter name and Datatype

Extract a substring from a given string (word), starting at the position specified by **startPos** and spanning the length specified by **endPos**.

```
CREATE OR REPLACE FUNCTION fn_mid()
returns varchar
as
$$
begin
        return substring();
end;
$$
language plpgsql;

select fn_mid('software', 5,3)
```

FUNCTION – USE INOUT PARAMETER TYPE

- In PostgreSQL PL/pgSQL, function parameters can be defined with different modes:
  - IN (Default) → Input-only parameter
  - OUT → Output-only parameter
  - INOUT → Acts as both an input and an output parameter
- What is INOUT?
  - INOUT parameters serve as both inputs and outputs in a function.
  - The function modifies the input value and returns it.
  - This allows you to return multiple values without needing a RETURNS clause.

FUNCTION EXAMPLE 04 – Inout Parameter Type

Create a function to swap the two points

```
-- parameter type{in*|out|inout|VARIADIC**}  *DEFAULT **variable number of arguments
create or replace function fnSwap()
as
$$
begin
  end;
$$
language plpgsql;

select fnSwap(10,20)
```

FUNCTION – ARRAY INPUT PARAMETER

- Function Parameters
  - Accepts an array of numeric values (numeric[]).
  - Looping Through an Array

- FOREACH val IN ARRAY n_array LOOP
  - Iterates through each value in the array.
  - Adds each value to total and increases the counter cnt.

FUNCTION EXAMPLE 05 – Array Input Parameter

Create a **fnMean** function calculates the mean (average) of a numeric array in PostgreSQL.

```
CREATE OR REPLACE FUNCTION fnMean()
returns numeric
as
$$
declare
begin

end;
$$
language plpgsql;

select fnMean()
```

FUNCTION – Table

- Perform query on existing Table
- Specify name, column and query for Table
- RETURN QUERY – Used in PL/pgSQL to return the result of a SELECT statement.

FUNCTION EXAMPLE 06 – Table

- Create a function to filter the subway based on borough’s name
  - Right click ‘subway’, Scripts – Create Script to check the whole column names

```
CREATE OR REPLACE FUNCTION subway_filter()
returns table(
)
as
$$
begin
-- table alias is mandatory, or use tablename as alias
end;
$$
language plpgsql;
select * from subway_filter('Brooklyn')
```

FUNCTION – Dynamic SQL for Table

- **What is Dynamic SQL?**
  - Allows you to construct **SQL queries dynamically** at runtime.
  - Enables execution of **queries with variable table names, column names, and conditions**.
  - Used in **stored functions, procedures, and triggers** for flexibility.
- **Syntax for Dynamic SQL in PL/pgSQL**
  - Use EXECUTE to run dynamically built queries.
  - Use format() to safely construct queries.
  - Use %I for identifiers (tables, columns) and %L for literal values.

FUNCTION EXAMPLE 07 – Dynamic SQL for Table

- Create a function to dynamically filters subway station data based on a user-specified schema, table, column, and filter value.
    - Filter subway station based on color

```
CREATE OR REPLACE FUNCTION dynamic_subway_filter()
RETURNS TABLE ()
AS
$$
DECLARE
   sql_query TEXT;  -- Stores the dynamic SQL query
BEGIN
sql_query := format();
RETURN QUERY EXECUTE sql_query;
END;
$$
LANGUAGE plpgsql;
```

- format() Function is used to dynamically construct SQL queries.
- Identifiers (%I):
    - %I.%I → Ensures proper schema and table name formatting (e.g., ch05.subway).
    - %I → Ensures proper column name formatting.
- Literals (%L):
    - %L → Ensures that filter_value is correctly formatted as a string to prevent SQL injection.

FUNCTION – Dynamic SQL for Spatial Relationship

- **Why Use Dynamic SQL for Spatial Relationship?**
  - Enables flexible queries based on user-defined inputs (e.g., schema, table, column names).
  - Allows complex spatial operations, like ST_Intersects() and ST_Buffer(), to be dynamically generated.
  - Helps avoid hardcoding table and column names, making functions more reusable.

FUNCTION EXAMPLE 08 – Dynamic SQL for Spatial Relationship

Find streets that intersects with subway stations with specified color

```
CREATE OR REPLACE FUNCTION dynamic_subway_filter()
RETURNS TABLE ()
AS
$$
DECLARE
    sql_query TEXT;  -- Stores the dynamic SQL query
BEGIN
sql_query := format();
RETURN QUERY EXECUTE sql_query;
END;
$$
LANGUAGE plpgsql;
select * from dynamic_subway_filter('ch05','streets', 'ch05', 'subway', 'color', 'GREEN', 100)
```