

# Database Development and Design

# Database Development and Design

Developing and Design of databases  
using PostgreSQL - Powerful, open-  
source object-relational database



# Agenda



- Session 4
  - Homework discussion
  - Quiz
  - Built-in functions and operators
    - Mathematical functions and operators
    - String functions and operators
    - Temporary tables
    - SELECT INTO
    - Workshop
  - Introduction to PL/pgSQL
    - Overview of PL/pgSQL
    - Advantages and disadvantages of PL/pgSQL
    - Dollar-quoted string constant
    - Variables in PL/pgSQL
    - Workshop
  - User-defined functions
    - Creating functions
    - Calling functions
    - Workshop
  - Knowledge check (Workshop, Homework)



# Built-In functions and operators

# Built-In functions – Mathematical operators



Operator	Description	Example	Result
+	Addition	$2 + 3$	5
-	Subtraction	$3 - 2$	1
*	Multiplication	$2 * 3$	6
/	Division (integer division truncates the result)	$4 / 2$	2
^	Exponentiation	$2 ^ 3$	8
@	Absolute value	@ -5	5

# Built-In functions – Mathematical functions



Function	Description	Example	Result
<code>abs(x)</code>	Absolute value	<code>abs(-13.2)</code>	13.2
<code>ceil(x)</code>	Nearest integer greater than or equal to argument	<code>ceil(13.2)</code>	14
<code>floor(x)</code>	Nearest integer less than or equal to argument	<code>floor(13.8)</code>	13
<code>round(x)</code>	Round to nearest integer	<code>round(14.8)</code>	15
<code>random(x)</code>	Random value in the range $0.0 \leq x < 1.0$	<code>random()</code>	0.12514...

# Built-In functions – String functions and operators 1/2



Function	Description	Example	Result
string    string	String concatenation	'Post'    'greSQL'	PostgreSQL
string    non-string <b>or</b> non-string    string	String concatenation with one non-string input	'Value: '    42	Value: 42
char_length(string) <b>or</b> character_length(string) <b>or</b> length(string)	Number of characters in a string	char_length('bojan')	5
lower(string)	Convert string to lower case	lower('BOJAN')	bojan
position(substring in string)	Location in string of specified substring	position('an' in 'Bojan')	4
substring(string text <b>from</b> N int <b>for</b> M int)	Extract substring <b>from</b> n-th position <b>for</b> m number of characters	substring('Seavus' from 2 for 3)	eav
upper(string)	Convert string to upper case	upper('bojan')	BOJAN

# Built-In functions – String functions and operators 2/2



Function	Description	Example	Result
<code>concat(string1, string2, ...)</code>	Concatenate all arguments. Null arguments are ignored.	<code>concat('Bojan', 2, NULL, 'Zdravkovski')</code>	Bojan2Zdravkovski
<code>left(str text, n int)</code>	Return first n characters in the string.	<code>left('Bojan', 2)</code>	Bo
<code>replace(string text, from text, to text)</code>	Replace all occurrences in string of substring <b>from</b> with substring <b>to</b> .	<code>replace('123ab123cd123ef', '123', 'XX')</code>	XXabXXcdXXef
<code>reverse(str)</code>	Return reversed string.	<code>reverse('Bojan')</code>	najoB
<code>right(str text, n int)</code>	Return last n characters in the string.	<code>right('Bojan', 2)</code>	an



# Temporary tables



- A temporary table is a short-lived table that exists for the duration of a database session. PostgreSQL automatically drops the temporary tables at the end of a session or a transaction.
- To create a temporary table, you use the CREATE TEMPORARY TABLE statement.
- The TEMP and TEMPORARY keywords are equivalent so you can use them interchangeably:

```
CREATE TEMPORARY TABLE temp_table_name(  
    column_list  
);
```

# SELECT INTO



- The PostgreSQL SELECT INTO statement can create a new table and insert data returned from a query into the table.

```
SELECT
    select_list
INTO TEMPORARY TABLE new_table_name
FROM
    table_name
WHERE
    search_condition;
```

- More commonly, SELECT INTO is used to assign the values from the select statement into a variable. We will get on this later.

# Built-in functions – workshop



- Concatenate director's first name and last name with movie title
- Convert all genre names to uppercase
- Convert all movie titles to lowercase
- Extract the first 10 characters of movie titles
- Extract the last 5 characters of actor names
- Extract the last 5 characters of actor names
- Get the length of movie titles
- Extract a substring from movie plot summaries (characters 10-40)
- Remove leading and trailing spaces from movie locations
- Replace null values with 'Not Available' for movie languages
- Replace 'a' with 'A' in movie titles



# Introduction to PL/pgSQL

# Overview of PostgreSQL PL/pgSQL



- PL/pgSQL is a procedural programming language for the PostgreSQL database system.
- PL/pgSQL allows you to extend the functionality of the PostgreSQL database server by creating server objects with complex logic.
- PL/pgSQL was designed to:
  - Create user-defined functions, stored procedures, and triggers.
  - Extend standard SQL by adding control structures such as if, case, and loop statements.
  - Inherit all user-defined functions, operators, and types.
- Since PostgreSQL 9.0, PL/pgSQL is installed by default.

# Advantages of using PL/pgSQL 1/2



- SQL is a query language that allows you to query data from the database easily. However, PostgreSQL can only execute SQL statements individually.
- It means that you have multiple statements, you need to execute them one by one like this:
- Send a query to the PostgreSQL database server.
- Wait for it to process.
- Process the result set.
- Do some calculations.
- Send another query to the PostgreSQL database server and repeat this process.
- This process creates unnecessary network overhead and is not efficient for working with complex queries containing multiple statements.
- To resolve this issue, PostgreSQL uses PL/pgSQL.

# Advantages of using PL/pgSQL 2/2



- PL/pgSQL wraps multiple statements in an object and stores it on the PostgreSQL database server.
- So instead of sending multiple statements to the server one by one, you can send one statement to execute the object stored in the server. This allows you to:
- Reduce the number of round trips between the application and the PostgreSQL database server.
- Avoid transferring the intermediate results between the application and the server.

# PostgreSQL PL/pgSQL disadvantages



- Slower in software development because PL/pgSQL requires specialized skills that many developers do not possess.
- Difficult to manage versions and hard to debug.
- May not be portable to other database management systems.



# Dollar-quoted string constant syntax



- In PostgreSQL, you use single quotes for a string constant like this:

```
SELECT 'String constant';
```

- When a string constant contains a single quote ('), you need to escape it by doubling up the single quote. For example:

```
SELECT 'I''m also a string constant';
```

- In older versions of PostgreSQL, backslashes need to be escaped too.
- The problem arises when the string constant contains many single quotes and backslashes. Doubling every single quote and backslash makes the string constant more difficult to read and maintain.
- The dollar quoting feature gets rid of this problem. Example:  
`$tag$<string_constant>$tag$`

# Using dollar-quoted string constant in anonymous blocks



- The following shows the anonymous block in PL/pgSQL:

```
do
  'declare
    totalquantity integer;
  begin
    select count(quantity) into totalquantity from orderdetails;
    raise notice 'The total number of quantities for all orders is: %',
totalquantity;
  end;';
```

- To avoid escaping every single quotes and backslashes, you can use the dollar-quoted string as follows:

```
do
$$
  declare
    totalquantity integer;
begin
  select count(quantity) into totalquantity from orderdetails;
  raise notice 'The total number of quantities for all orders is: %', totalquantity;
end; $$;
```

# Variables in PL/pgSQL



- A variable is a meaningful name of a memory location. A variable holds a value that can be changed through the block. A variable is always associated with a particular data type.
- Before using a variable, you must declare it in the declaration section of the PL/pgSQL block.

```
variable_name data_type [:= expression];
```

- Example:

```
do $$  
declare  
    first_name varchar(50) := 'John';  
    last_name  varchar(50) := 'Doe';  
    payment    numeric(11,2) := 20.5;  
begin  
    raise notice '% % has been paid % USD',  
                first_name,  
                last_name,  
                payment;  
end $$;
```



# User-defined functions

# CREATE FUNCTION 1/2



- The CREATE FUNCTION statement allows you to define a new user-defined function.
- The following illustrates the syntax of the create function statement:

```
create [or replace] function function_name(param_list)
    returns return_type
    language plpgsql
as
$$
declare
-- variable declaration
begin
-- logic
end;
$$
```

# CREATE FUNCTION 2/2



- First, specify the name of the function after the create function keywords. If you want to replace the existing function, you can use the or replace keywords.
- Then, specify the function parameter list surrounded by parentheses after the function name. A function can have zero or many parameters.
- Next, specify the datatype of the returned value after the returns keyword.
- After that, use the language plpgsql to specify the procedural language of the function. Note that PostgreSQL supports many procedural languages, not just plpgsql.
- Finally, place a block in the dollar-quoted string constant.

# Calling a user-defined function



- PostgreSQL provides you with three ways to call a user-defined function:
- Using positional notation
- Using named notation
- Using the mixed notation

```
create function get_product_count_by_weight(min_weight int, max_weight int)
    returns int
    language plpgsql
as
$$
declare
    product_count integer;
begin
    select count(*)
    into product_count
    from product
    where weight between min_weight and max_weight;

    return product_count;
end;
$$;
```

# Calling a user-defined function using positional notation



- To call a function using the positional notation, you need to specify the arguments in the same order as parameters. For example:

```
select get_product_count_by_weight(40, 90);
```



# Calling a user-defined function using named notation



- In the named notation, you use the `=>` to separate the argument's name and its value.

```
select get_product_count_by_weight(  
    min_weight => 40,  
    max_weight => 90  
);
```

# Calling a user-defined function using mixed notation



- The mixed notation is the combination of positional and named notations. For example:

```
select get_product_count_by_weight(40, max_weight => 90);
```

- Note that you cannot use the named arguments before positional arguments like this:

```
select get_product_count_by_weight(min_weight => 40, 90);
```

# User-defined functions – workshop



- Declare function (`calculate_movie_age`) to calculate movie age in years
- Declare function (`format_full_name`) to format full name (combines first and last name with proper spacing)
- Declare function to (`calculate_profit`) to calculate movie profit

# Questions?

Trainer Name

Trainer mail

Assistant Name

Assistant mail