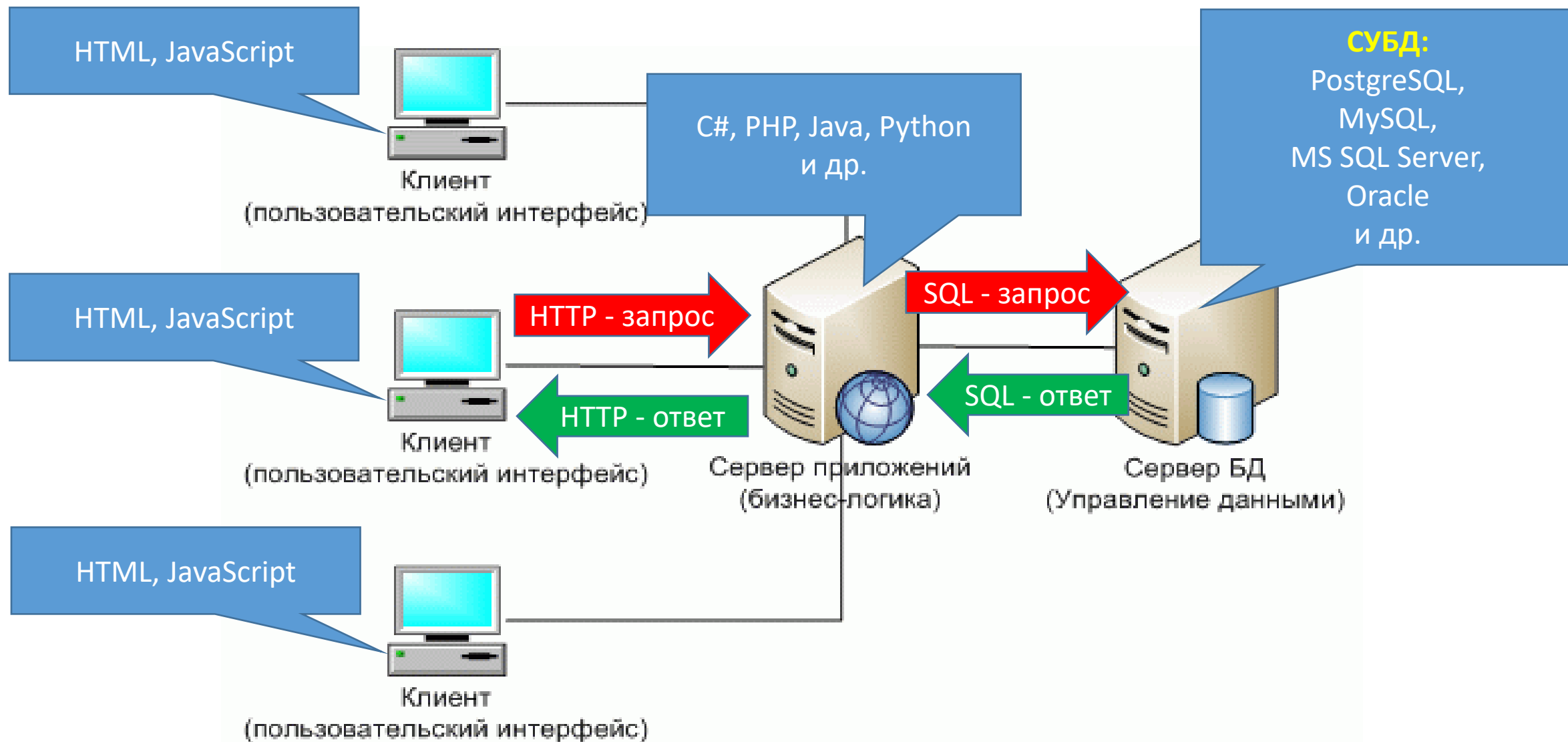
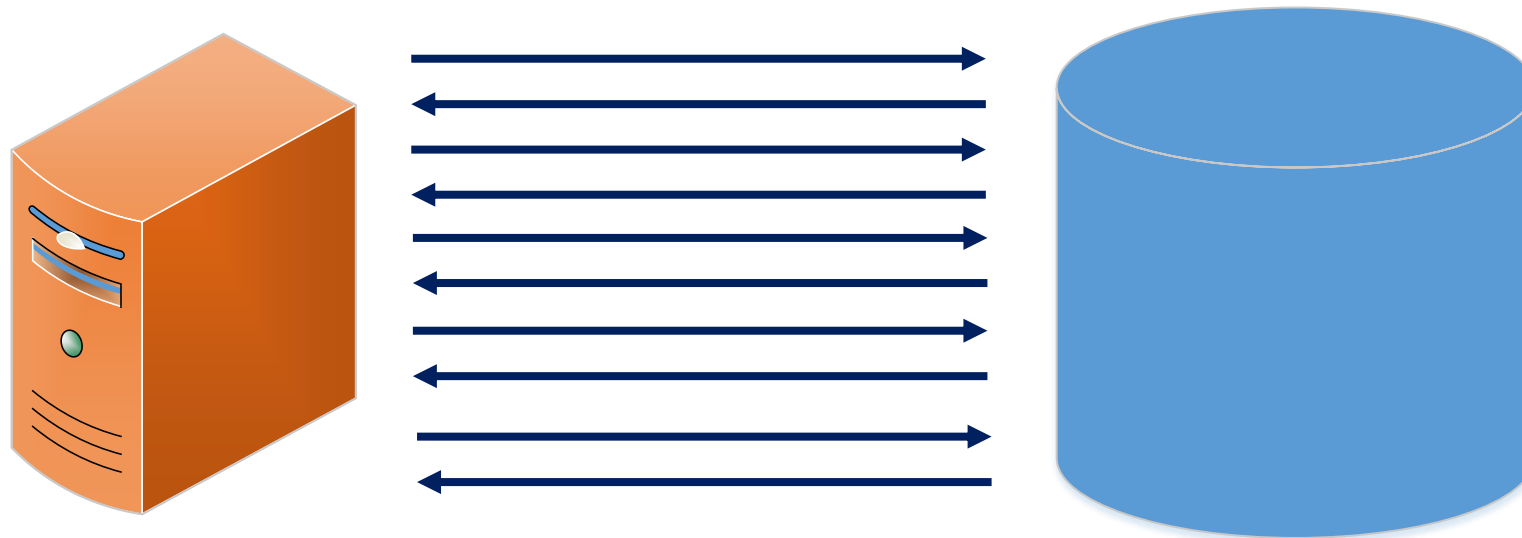


Трёхзвенная архитектура

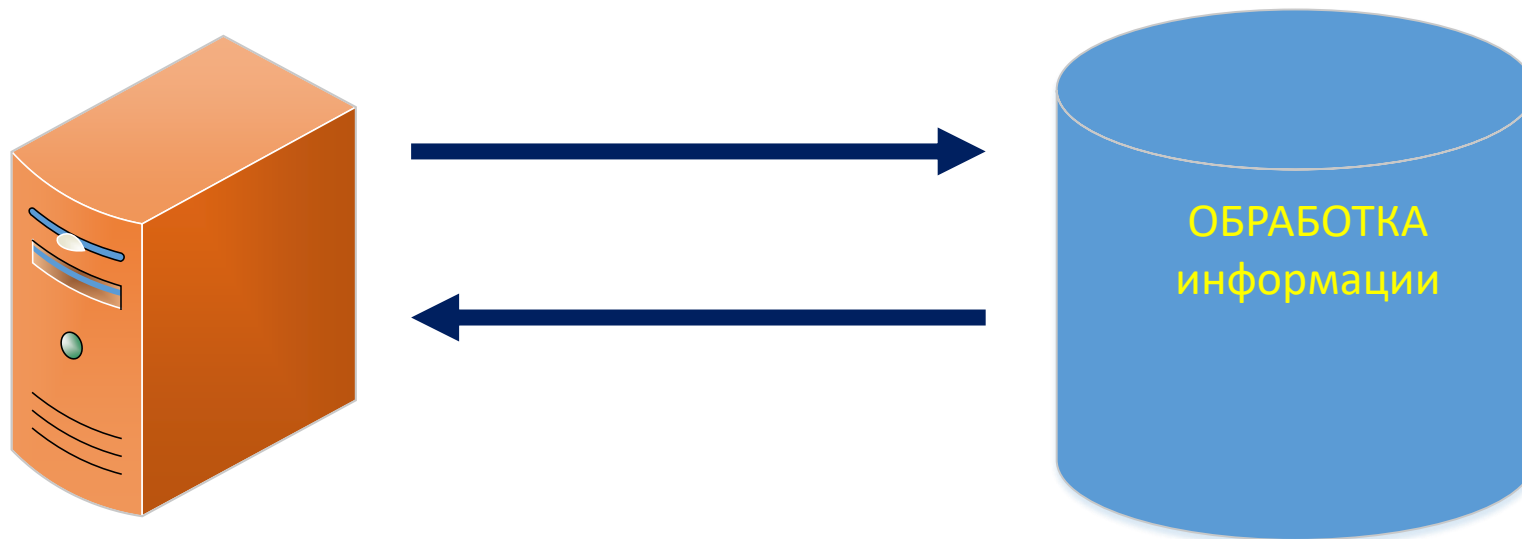


В процессе обработки HTTP-запроса, web-сервер может делать большое количество SQL-запросов к СУБД, при этом, между WEB-сервером и сервером СУБД может возникать большой трафик промежуточных данных.



Для уменьшения количества запросов и объёма информационного трафика между WEB-сервером и сервером СУБД, было бы целесообразно часть бизнес-логики перенести на сервер СУБД.

Для этих целей служит язык PL/pgSQL



Для комплексной обработки данных, в СУБД PostgreSQL предусмотрена возможность создания хранимых **ПРОЦЕДУР** и **ФУНКЦИЙ**

Основные отличия **процедур** и **функций**:

Функция	Процедура
Вызывается в запросе, как его часть	Вызывается ключевым словом CALL и не может являться частью запроса
Может возвращать значения	Не возвращает значений, но это можно обойти через "выходные" параметры
Внутри функции невозможно управлять транзакциями (начинать и откатывать)	Если процедура не вызвана внутри другой явной транзакции, то внутри процедуры можно управлять транзакциями (начинать и откатывать полностью или к точке восстановления)
Создаётся командой CREATE FUNCTION	Создаётся командой CREATE PROCEDURE

Общий синтаксис объявления функции:

CREATE FUNCTION название(**тип**, **тип**)

RETURNS integer

AS \$\$

DECLARE

Объявление переменных

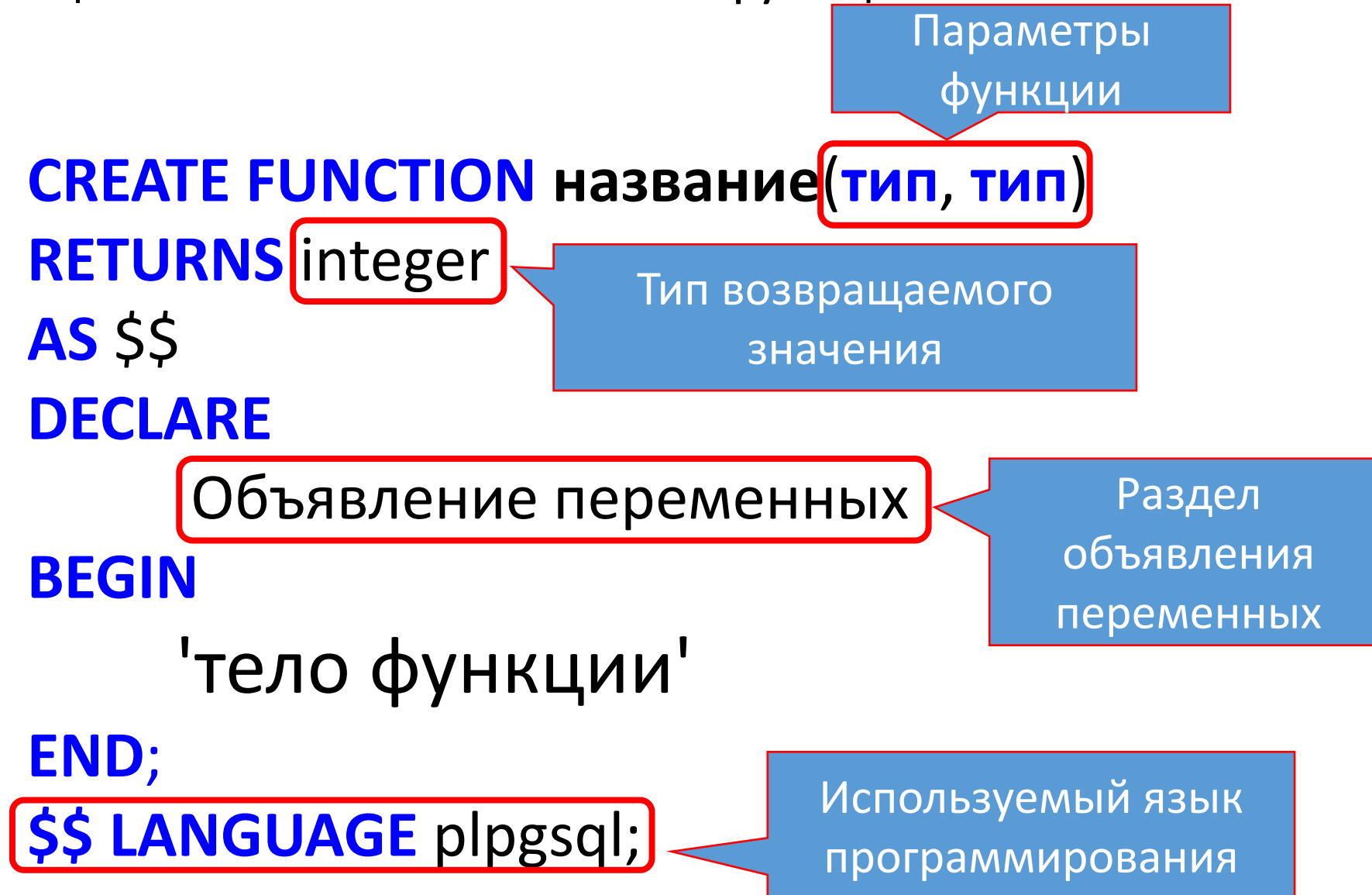
BEGIN

'тело функции'

END;

\$\$ LANGUAGE plpgsql;

Общий синтаксис объявления функции:



ПЕРЕМЕННЫЕ

Все переменные объявляются только в разделе объявления переменных

При объявлении переменных, обязательно, должен указываться тип

Пример объявления переменных:

DECLARE

```
user_id INTEGER;  
quantity NUMERIC(5);  
url VARCHAR(150);
```


ПЕРЕМЕННЫЕ

Переменные допустимо сразу инициализировать при объявлении.

Переменные можно защитить от изменения – объявить КОНСТАНТОЙ

Пример инициализации переменной и константы:

DECLARE

```
user_id INTEGER := 0;
```

```
number_pi CONSTANT INT := 3.14;
```

ПАРАМЕТРЫ ФУНКЦИИ

Параметры функции имеют идентификаторы вида \$1, \$2

Параметр \$1

Параметр \$2

```
CREATE FUNCTION get_sum(float, float)
```

```
RETURNS float
```

```
AS $$
```

```
DECLARE
```

```
    sum float;
```

```
BEGIN
```

```
    sum := $1 + $2;
```

```
    RETURN sum;
```

```
END;
```



```
$$ LANGUAGE plpgsql;
```

ВЫЗОВ ФУНКЦИИ

Функция может быть вызвана, почти, в любом месте SQL-запроса

Например так:

SELECT get_sum(5, 3.4)

Data Output			Explain	Messages	Notifications
		get_sum double precision			
1			8.4		

ПАРАМЕТРЫ ФУНКЦИИ

Параметры функции могут иметь псевдонимы, заданные при их объявлении

```
CREATE OR REPLACE FUNCTION get_sum(num_a float, num_b float)
RETURNS float
AS $$
DECLARE
    sum float;
BEGIN
    sum := num_a + $2;
    RETURN sum;
END;
$$ LANGUAGE plpgsql;
```

Обращаться к параметрам функции
можно как через псевдоним, так и
через идентификатор

В процедурах и функциях допустимо применять все арифметические и логические операции, допустимые в языке **SQL**

Управляющие конструкции PL/pgSQL

УСЛОВИЯ

Условные операторы:

IF условие (логическое выражение)

THEN

действия

END IF;

Действия после THEN выполняются,
только если условие **ИСТИННО**

Условные операторы:

IF условие (логическое выражение)

THEN

действия

Действия после THEN выполняются,
только если условие **ИСТИННО**

ELSE

действия

Действия после ELSE выполняются,
только если условие **ЛОЖНО**

END IF;

Условные операторы:

IF условие 1

THEN

действия 1

Эти действия после THEN выполняются,
только если условие 1 **ИСТИННО**

ELSEIF условие 2

действия 2

Эти действия выполняются только если
условие 2 **ИСТИННО**

ELSEIF условие 3

действия 3

Эти действия выполняются только если
условие 3 **ИСТИННО**

ELSE

действия 4

Действия 4, после ELSE выполняются,
только если все условия **ЛОЖНЫ**

END IF;

ЦИКЛЫ

Бесконечный цикл

LOOP

операторы

END LOOP;

LOOP организует безусловный цикл, который повторяется до бесконечности, пока не будет прекращён операторами **EXIT** или **RETURN**.

Цикл с предварительным условием

WHILE логическое-выражение
LOOP
 операторы
END LOOP;

WHILE организует условный цикл, который повторяется до тех пор, пока истинно заданное, после **WHILE**, логическое выражение или пока не будет прекращён операторами **EXIT** или **RETURN**.

Цикл со счётчиком

```
FOR i IN 1..10 LOOP
```

-- внутри цикла переменная i будет иметь значения 1,2,3,4,5,6,7,8,9,10

```
END LOOP;
```

В этом цикле значения переменной **i** будут перебираться в заданном диапазоне в заданном направлении, с заданным шагом

Цикл со счётчиком

```
FOR i IN REVERS 1..10 LOOP
```

-- внутри цикла переменная i будет иметь значения 10,9,8,7,6,5,4,3,2,1

```
END LOOP;
```

Если мы укажем направление REVERS, то проход будет идти в обратном направлении.

Цикл со счётчиком

```
FOR i IN REVERS 1..10 BY 2 LOOP
```

-- внутри цикла переменная i будет иметь значения 10,8,6,4,2

```
END LOOP;
```

Цифрой, после ключевого слова BY можно задать шаг прохода по диапазону

