

КОМИТЕТ ПО ОБРАЗОВАНИЮ ПРАВИТЕЛЬСТВА САНКТ-
ПЕТЕРБУРГА

Санкт-Петербургское государственное
бюджетное профессиональное образовательное учреждение
«КОЛЛЕДЖ ЭЛЕКТРОНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

«МДК.07.01 Управление и автоматизация баз данных»

ОТЧЁТ

по лабораторной работе №5

«Программирование базы данных»

Работу выполнил студент

325гр.:

Шлычков И. Д.

Преподаватель: Фомин А.В.

Санкт-Петербург 2025

Цель работы:

Разработка функций для управления и автоматизации базы данных, включая обработку запросов для получения информации о регионах, странах, городах, береговых линиях и температурных измерениях.

Здесь мы создаем схему «api» и реализуем функции из указанного ниже набора, чтобы сделать следующие функции. Рисунок 1:

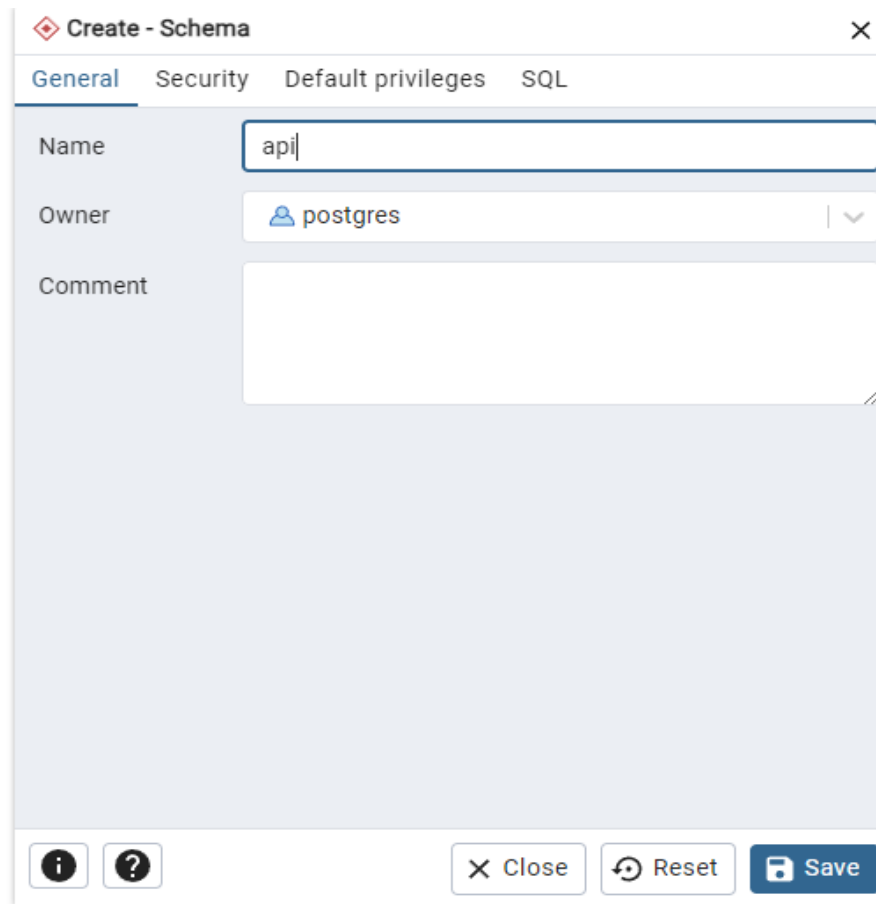


Рисунок 1 – создание схемы

- get_region_countries_count - количество стран в регионах
- get_country_cities_count - количество городов в странах
- get_regions - список регионов
- get_countries - список стран
- get_cities - список городов
- get_city_locations - список местоположений городов
- get_coastline_shapes - список береговых линий
- get_shape_points - координаты точек ломаной береговой линии
- get_measurement_time_range - начальная и конечная дата измерений по городу
- get_daily_temperatures - список измерений температуры по городу за определенное время

- get_daily_temperatures_reduce - усредненные значения температуры по городу за определенное время в указанном количестве

На данном рисунке представлен фрагмент кода функций. Рисунок 2



```
postgres/postgres@PostgreSQL 17
Query History
1 CREATE OR REPLACE FUNCTION api.get_region_countries_count()
2 RETURNS TABLE(region_id integer, country_count bigint) AS $$
3 BEGIN
4     RETURN QUERY
5     SELECT r.identifier, COUNT(c.identifier)::bigint
6     FROM data.region r
7     LEFT JOIN data.country c ON r.identifier = c.region
8     GROUP BY r.identifier;
9 END;
10 $$ LANGUAGE plpgsql;
11
12 CREATE OR REPLACE FUNCTION api.get_country_cities_count()
13 RETURNS TABLE(country_id integer, city_count bigint) AS $$
14 BEGIN
15     RETURN QUERY
16     SELECT c.identifier, COUNT(ci.identifier)::bigint
17     FROM data.country c
18     LEFT JOIN data.city ci ON c.identifier = ci.country
19     GROUP BY c.identifier;
20 END;
21 $$ LANGUAGE plpgsql;
22
23 CREATE OR REPLACE FUNCTION api.get_regions()
24 RETURNS TABLE(id integer, name text) AS $$
25 BEGIN
26     RETURN QUERY
27     SELECT r.identifier, r.description
28     FROM data.region r;
29 END;
30 $$ LANGUAGE plpgsql;
31
32 CREATE OR REPLACE FUNCTION api.get_countries(region_id_param integer)
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 119 msec.

Рисунок 2 – фрагмент кода

Убеждаемся, что наши функции были созданы. Рисунок 3

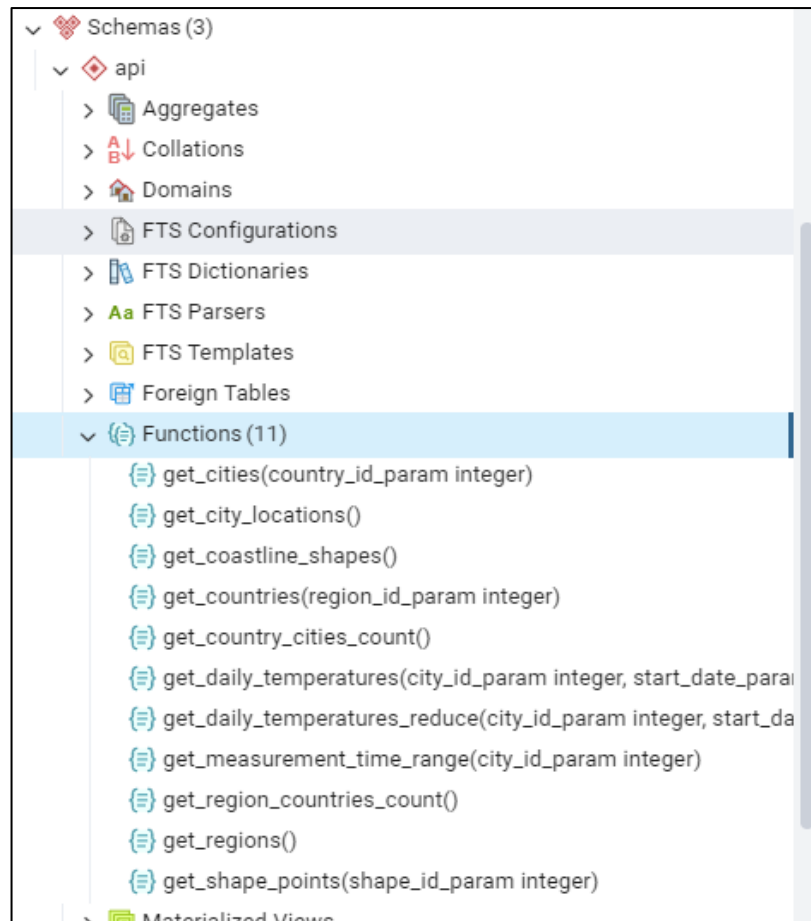
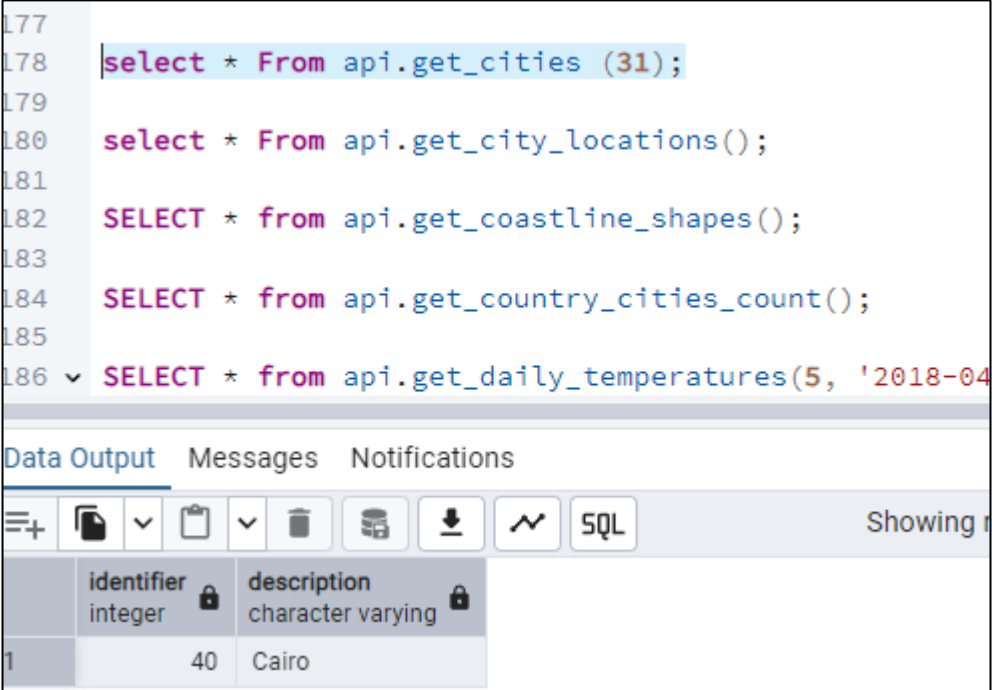


Рисунок 3 – функции

Описание функции get_cities

Функция выводит список городов по указанному идентификатору страны, результат включает в себя следующие поля: идентификатор города и название города. На рисунке 4 показан пример работы функции.



The screenshot shows a SQL IDE interface. The top pane contains a SQL query with line numbers 177 to 186. The query is:

```
177  
178 select * From api.get_cities (31);  
179  
180 select * From api.get_city_locations();  
181  
182 SELECT * from api.get_coastline_shapes();  
183  
184 SELECT * from api.get_country_cities_count();  
185  
186 SELECT * from api.get_daily_temperatures(5, '2018-04
```

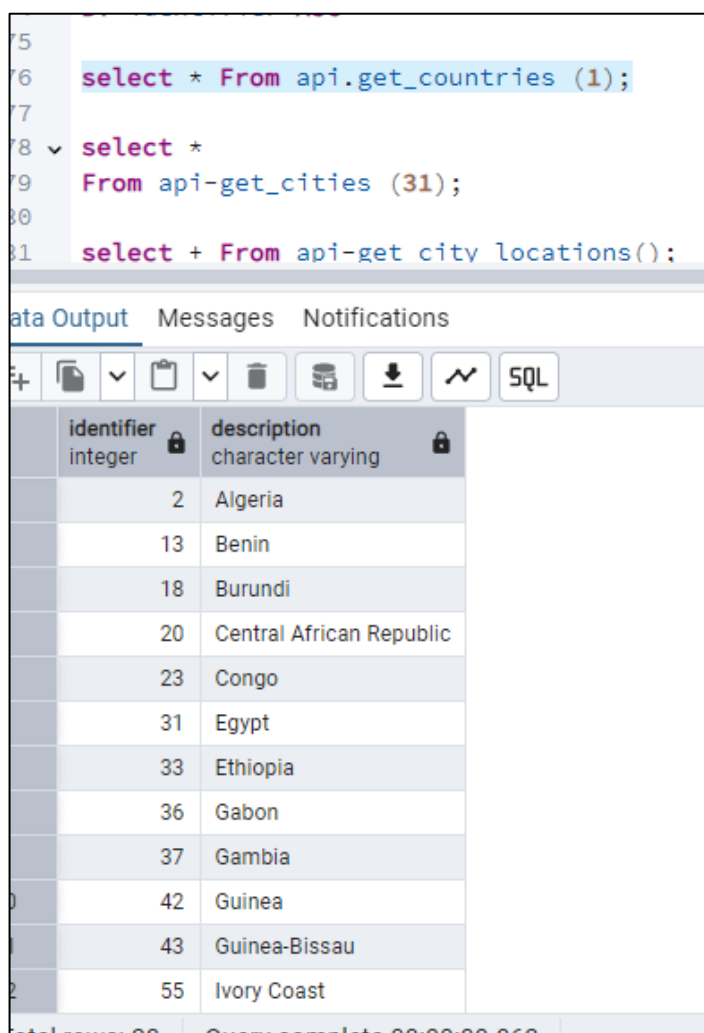
The bottom pane is titled "Data Output" and shows the result of the query. It has a toolbar with icons for expand, copy, paste, delete, save, download, and a "SQL" button. The result is displayed in a table with two columns: "Identifier" (integer) and "description" (character varying). The first row shows the value 40 for the Identifier and Cairo for the description.

	Identifier integer	description character varying
1	40	Cairo

Рисунок 4 – результат

Описание функции get_countries

Возвращает результат выполнения запроса на выборку стран в указанном по идентификатору регионе. в результирующей таблице содержится информация об идентификаторе страны и ее названии. На рисунке 5 показан пример работы



The screenshot shows a database query interface. The top part displays SQL code in a text editor. The bottom part shows the 'Data Output' tab with a table of results. The table has two columns: 'identifier' (integer) and 'description' (character varying). The results list countries with their identifiers: 2 (Algeria), 13 (Benin), 18 (Burundi), 20 (Central African Republic), 23 (Congo), 31 (Egypt), 33 (Ethiopia), 36 (Gabon), 37 (Gambia), 42 (Guinea), 43 (Guinea-Bissau), and 55 (Ivory Coast). The status bar at the bottom indicates 'total rows: 22' and 'Query complete 00:00:00.060'.

```
75  
76 select * From api.get_countries (1);  
77  
78 select *  
79 From api-get_cities (31);  
80  
81 select + From api-get city locations();
```

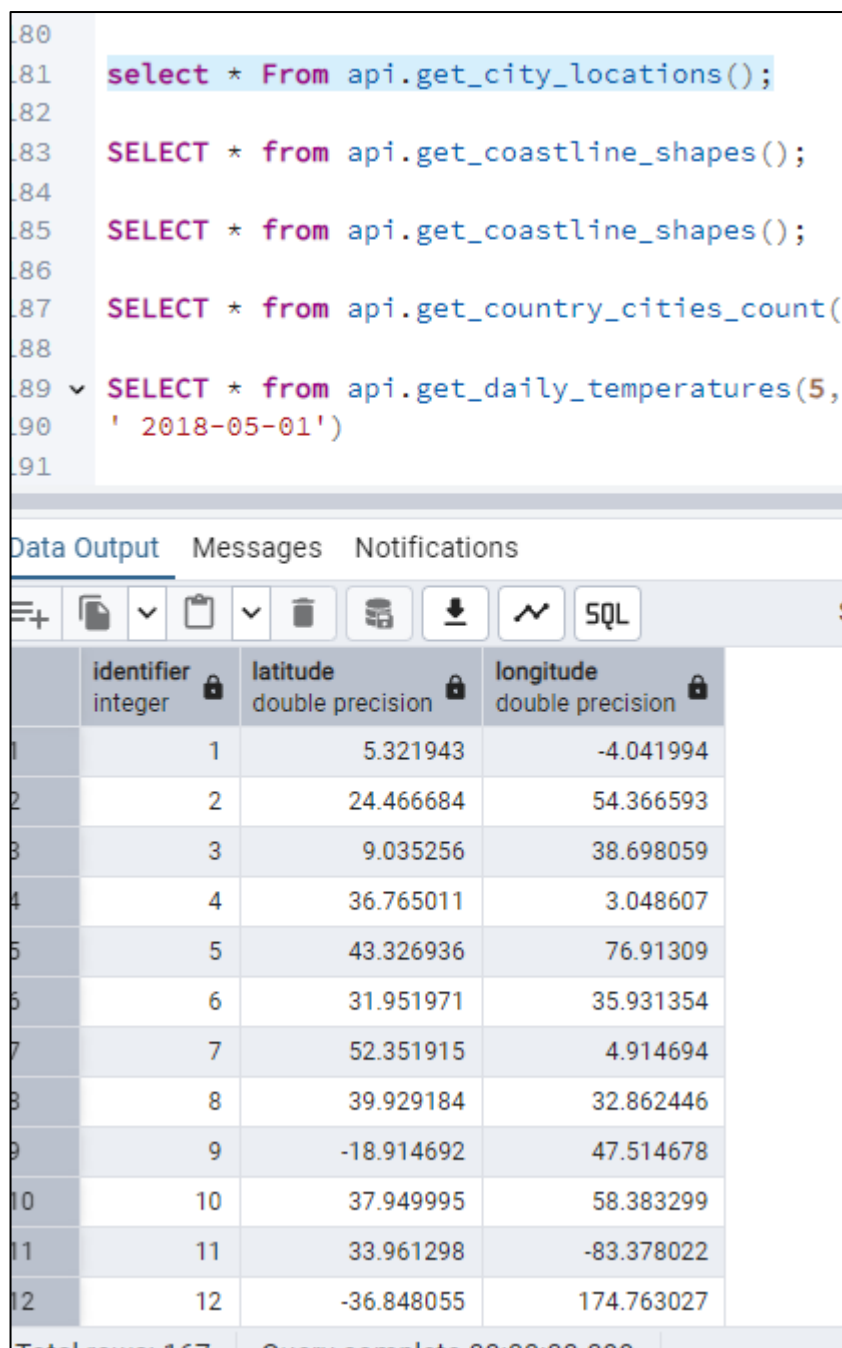
identifier integer	description character varying
2	Algeria
13	Benin
18	Burundi
20	Central African Republic
23	Congo
31	Egypt
33	Ethiopia
36	Gabon
37	Gambia
42	Guinea
43	Guinea-Bissau
55	Ivory Coast

total rows: 22 Query complete 00:00:00.060

Рисунок 5 – результат

Описание функции get_city_locations

Функция возвращает список идентификаторов всех городов и их географические координаты. На рисунке 6 показан пример работы функции.



The screenshot displays a database management tool interface. At the top, a SQL editor shows several queries. The first query, highlighted in blue, is `select * From api.get_city_locations();`. Below it are queries for `api.get_coastline_shapes()`, `api.get_country_cities_count()`, and `api.get_daily_temperatures(5, '2018-05-01')`. The 'Data Output' tab is active, showing a table with 12 rows of data. The table has three columns: 'Identifier' (integer), 'latitude' (double precision), and 'longitude' (double precision). The data represents the first 12 cities from the `get_city_locations()` function. At the bottom, a status bar indicates 'Total rows: 167' and 'Query complete 00:00:00.000'.

	Identifier integer	latitude double precision	longitude double precision
1	1	5.321943	-4.041994
2	2	24.466684	54.366593
3	3	9.035256	38.698059
4	4	36.765011	3.048607
5	5	43.326936	76.91309
6	6	31.951971	35.931354
7	7	52.351915	4.914694
8	8	39.929184	32.862446
9	9	-18.914692	47.514678
10	10	37.949995	58.383299
11	11	33.961298	-83.378022
12	12	-36.848055	174.763027

Total rows: 167 Query complete 00:00:00.000

Рисунок 6 – результат

Описание функции get_coastline_shapes

Выводит список всех замкнутых фигур, состоящих из соединенных друг с другом отрезков, необходимых для отрисовки береговой линии. Для каждой фигуры дополнительно выводится количество точек ломаной линии. На рисунке 7 показан пример работы функции.

The screenshot shows a SQL IDE interface. The top pane contains a list of SQL queries. The bottom pane shows the 'Data Output' tab with a table of results for the query `SELECT * from api.get_shape_points (0);`.

```
6
7 SELECT * from api.get_coastline_shapes();
8
9 SELECT * from api.get_country_cities_count()
10
11 SELECT * from api.get_daily_temperatures(5,
12
13 SELECT * from api.get_measurement_time_range
14
15 SELECT * from api.get_daily_temperatures_red
16
17 SELECT * from api.get_region_countries_count
18
19 SELECT * from api.get_regions();
20
21 SELECT * from api.get_shape_points (0);
```

	shape integer
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11

Total rows: 134 Query complete 00:00:00.105

Рисунок 7 – результат

Описание функции get_country_cities_count

Подсчитывает количество городов в каждой стране и выводит идентификаторы этих стран с указанием количества городов. На рисунке 8 показан пример работы функции.

The screenshot shows a SQL IDE interface. The top panel displays a list of SQL queries. The bottom panel shows the 'Data Output' tab with a table of results.

```
8
9 SELECT * from api.get_country_cities_count();
10
11 SELECT * from api.get_daily_temperatures(5, '2018-04-01'
12
13 SELECT * from api.get_measurement_time_range(5);
14
15 SELECT * from api.get_daily_temperatures_reduce(10, '202
16
17 SELECT * from api.get_region_countries_count();
18
19 SELECT * from api.get_regions();
20
21 SELECT * from api.get_shape_points (0);
```

Data Output Messages Notifications

Showing rows: 1

	identifier integer	countries integer
1	19	10
2	4	5
3	21	5
4	39	4
5	50	4
6	104	3
7	70	3
8	16	3
9	56	3
10	100	3
11	83	2
12	113	2

Рисунок 8 – результат

Описание функции get_daily_temperatures

Возвращает полный список показаний температуры в определенном городе по его идентификатору за выбранный период времени. В таблице с результатами присутствуют столбцы с временной отметкой в указанном диапазоне и температура в градусах Цельсия. На рисунке 9 показан пример работы функции.

The screenshot shows a SQL query editor with the following queries:

```
10
11 SELECT * from api.get_daily_temperatures(5, '2018-04-01', '2018-05-01')
12
13 SELECT * from api.get_measurement_time_range(5);
14
15 SELECT * from api.get_daily_temperatures_reduce(10, '2020-04-01', '2020-
16
17 SELECT * from api.get_region_countries_count();
18
19 SELECT * from api.get_regions();
20
21 SELECT * from api.get_shape_points (0);
```

Below the queries is a 'Data Output' section with a table showing the results of the first query. The table has two columns: 'ts' (timestamp without time zone) and 'temperature' (double precision). The results show daily temperature readings for April 2018.

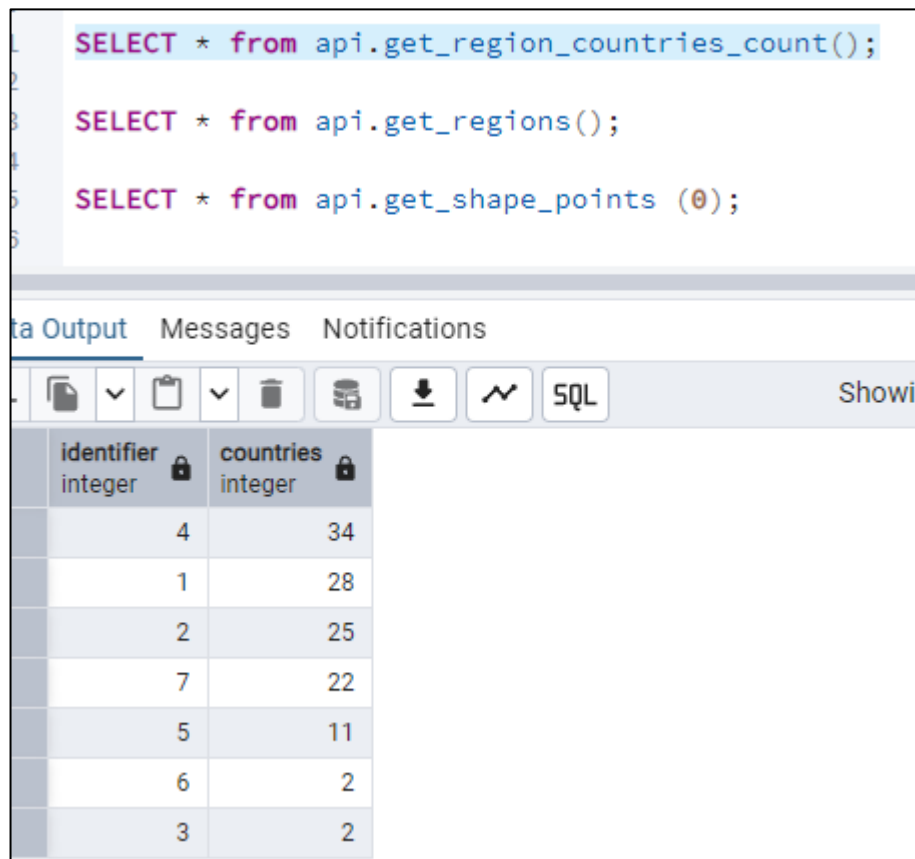
	ts timestamp without time zone	temperature double precision
1	2018-04-01 00:00:00	0.5555555555555556
2	2018-04-02 00:00:00	5.1666666666666665
3	2018-04-03 00:00:00	5.9444444444444446
4	2018-04-04 00:00:00	8.2222222222222221
5	2018-04-05 00:00:00	14.000000000000002
6	2018-04-06 00:00:00	17.777777777777778
7	2018-04-07 00:00:00	12.555555555555557
8	2018-04-08 00:00:00	8.777777777777777
9	2018-04-09 00:00:00	11.555555555555555
10	2018-04-10 00:00:00	7.8333333333333335
11	2018-04-11 00:00:00	5.1666666666666665
12	2018-04-12 00:00:00	9.000000000000002

A warning message is displayed in the bottom right corner: "You are currently however the cur Please click [here](#)".

Рисунок 9 – результат

Описание функции get_region_countries_count

Подсчитывает количество стран в каждом из регионов и выводит таблицу с двумя столбцами: идентификатор региона и количество стран в нем. На рисунке 10 показан пример работы функции.



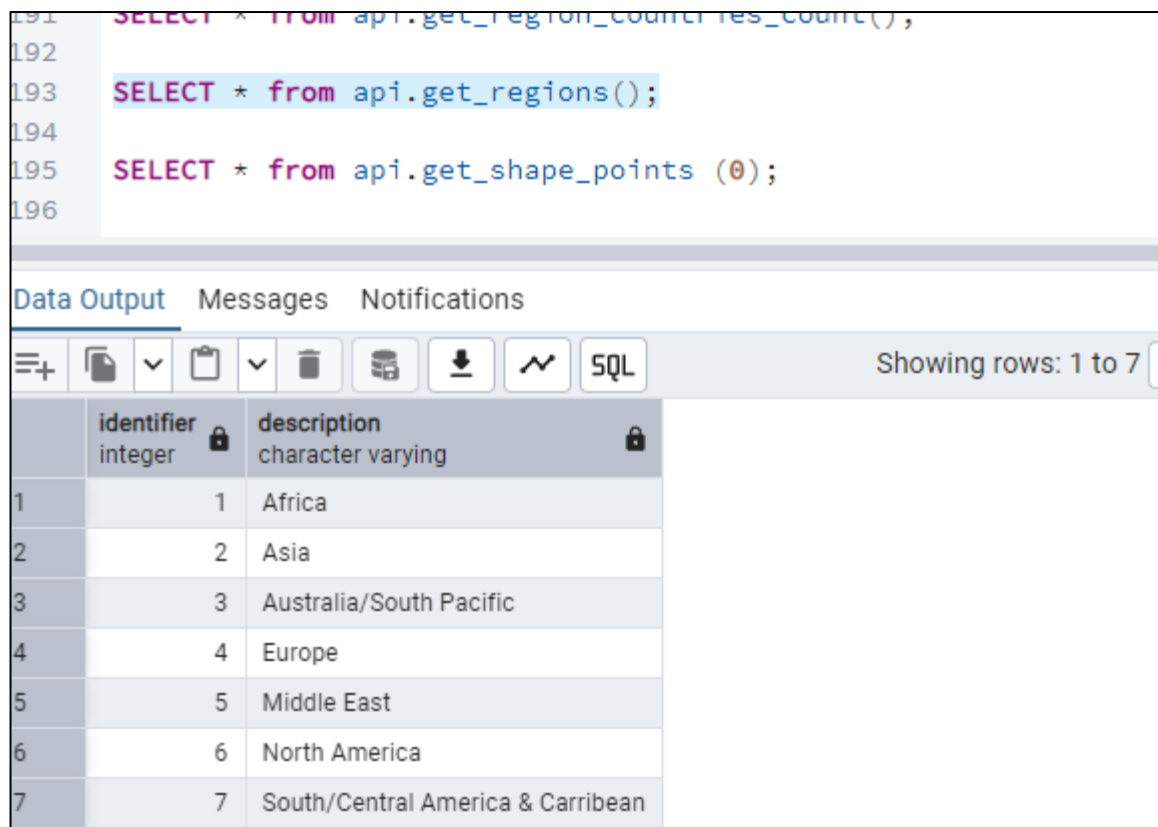
```
1 SELECT * from api.get_region_countries_count();
2
3 SELECT * from api.get_regions();
4
5 SELECT * from api.get_shape_points (0);
6
```

identifier integer	countries integer
4	34
1	28
2	25
7	22
5	11
6	2
3	2

Рисунок 10 – результат

Описание функции get_regions

Функция выводит список всех регионов с указанием их идентификатора и названия. На рисунке 11 показан пример работы функции.



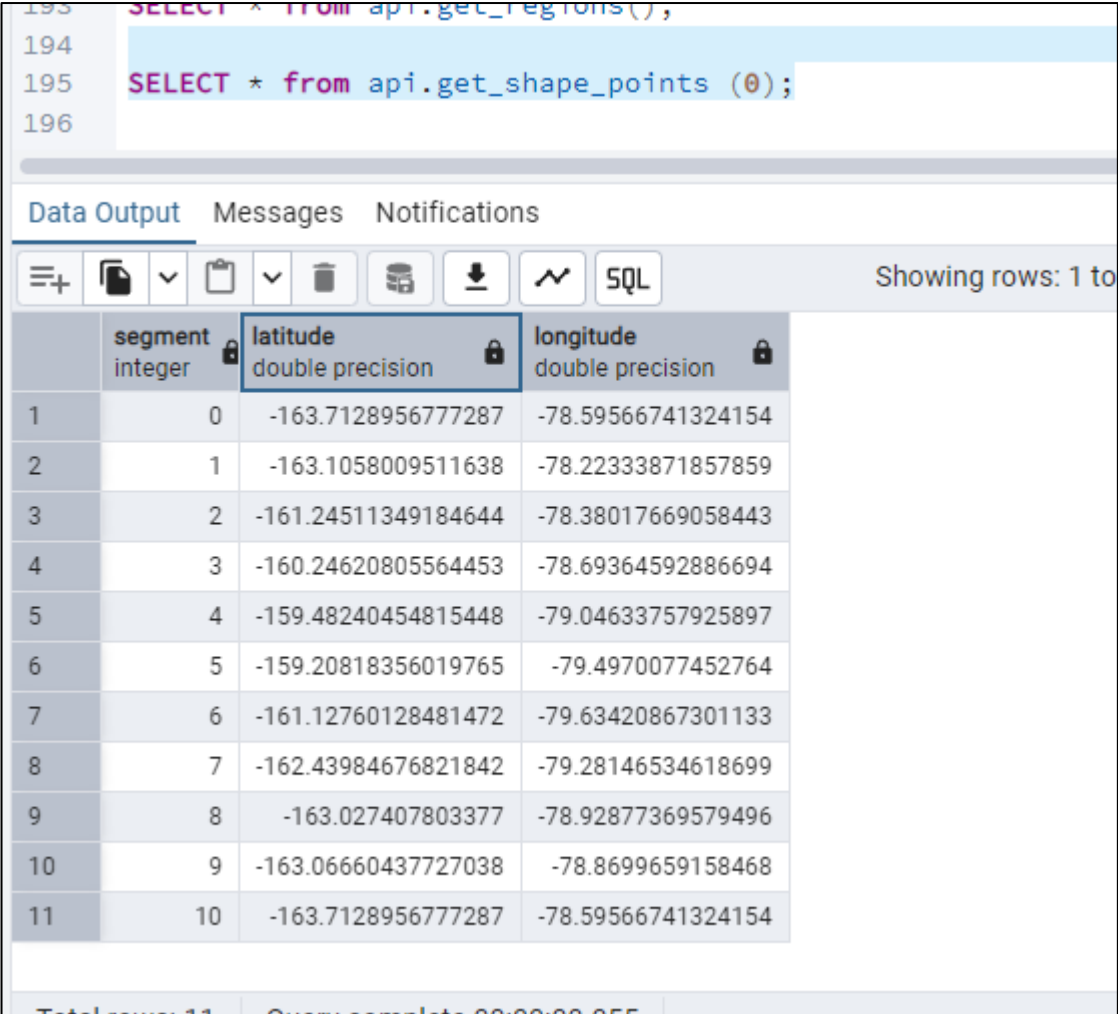
The screenshot displays a SQL editor with three queries. The second query, `SELECT * from api.get_regions();`, is highlighted. Below the editor, the 'Data Output' tab shows the results of this query as a table with two columns: 'identifier' (integer) and 'description' (character varying). The table contains seven rows representing different regions.

	identifier integer	description character varying
1	1	Africa
2	2	Asia
3	3	Australia/South Pacific
4	4	Europe
5	5	Middle East
6	6	North America
7	7	South/Central America & Carribean

Рисунок 11 – результат

Описание функции get_shape_points

В качестве результата своей работы, функция выдает список точек ломаной линии для вывода на экран одного из запрошенных фрагментов береговой линии. Каждая точка описывается своим порядковым номером и географическими координатами. На рисунке 12 показан пример работы функции.



The screenshot shows a database query interface. At the top, a SQL query is entered in a text area: `SELECT * FROM api.get_regions();` on line 193, followed by a blank line on 194, and `SELECT * from api.get_shape_points (0);` on line 195. Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, displaying a table of results. The table has four columns: an index column, 'segment integer', 'latitude double precision', and 'longitude double precision'. It contains 11 rows of data. At the bottom of the interface, a status bar indicates 'Total rows: 11' and 'Query complete 00:00:00.055'.

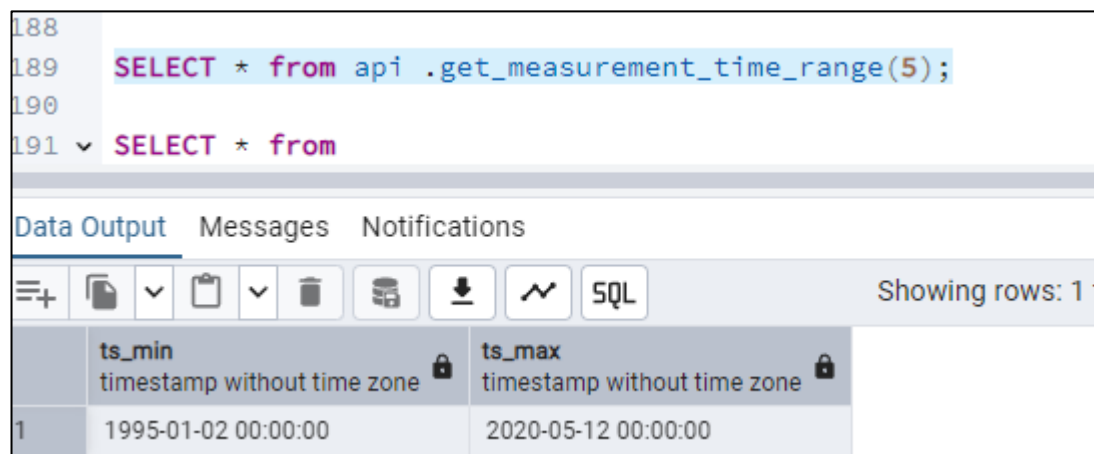
	segment integer	latitude double precision	longitude double precision
1	0	-163.7128956777287	-78.59566741324154
2	1	-163.1058009511638	-78.22333871857859
3	2	-161.24511349184644	-78.38017669058443
4	3	-160.24620805564453	-78.69364592886694
5	4	-159.48240454815448	-79.04633757925897
6	5	-159.20818356019765	-79.4970077452764
7	6	-161.12760128481472	-79.63420867301133
8	7	-162.43984676821842	-79.28146534618699
9	8	-163.027407803377	-78.92877369579496
10	9	-163.06660437727038	-78.8699659158468
11	10	-163.7128956777287	-78.59566741324154

Total rows: 11 Query complete 00:00:00.055

Рисунок 12 – результат

Описание функции get_measurement_time_range

Выводит всего одну строку для указанного по идентификатору города с двумя столбцами: датой самого раннего измерения температуры и самого позднего. На рисунке 13 показан пример работы функции



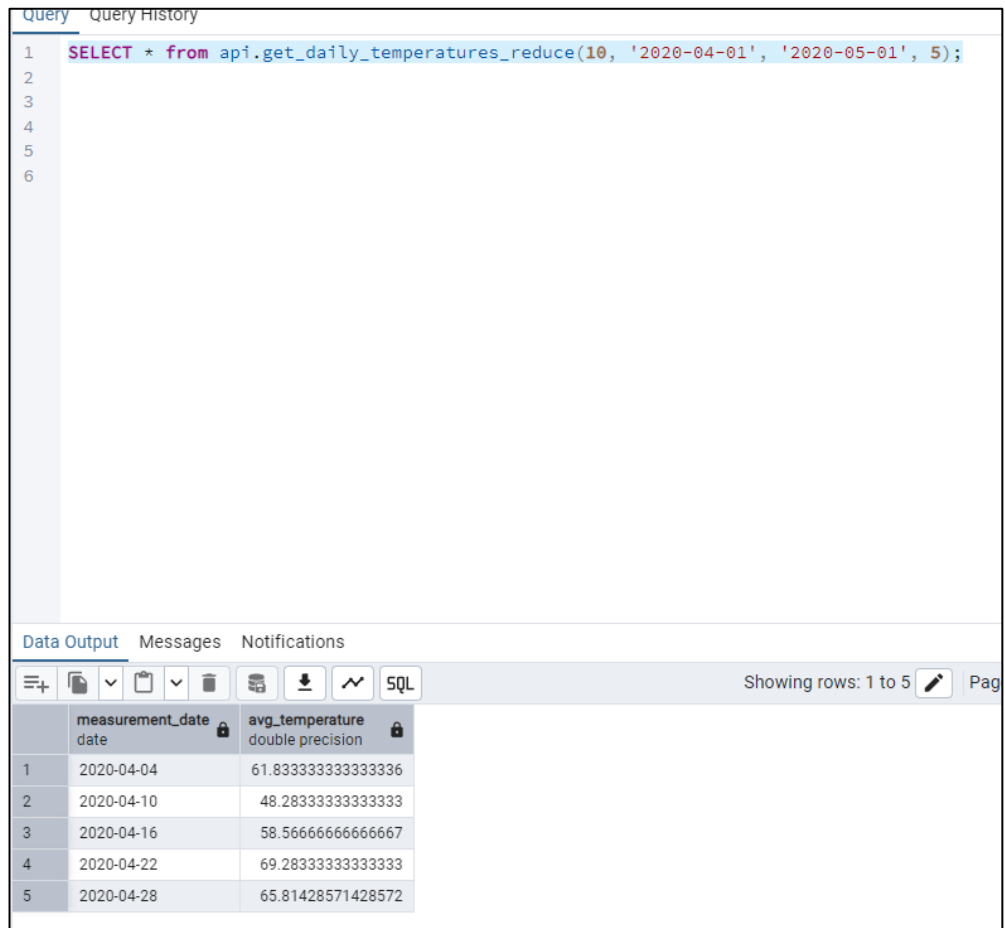
```
188
189 SELECT * from api.get_measurement_time_range(5);
190
191 SELECT * from
```

	ts_min timestamp without time zone	ts_max timestamp without time zone
1	1995-01-02 00:00:00	2020-05-12 00:00:00

Рисунок 13 – результат

Описание функции get_daily_temperatures_reduce

Вычисляет указанное количество значений температуры за заданный временной период путем усреднения всех найденных за этот период значений. Выборка производится по идентификатору города. На рисунке 14 показан пример работы функции.



The screenshot shows a database query interface. At the top, there are tabs for 'Query' and 'Query History'. Below them is a text area containing a SQL query: `SELECT * from api.get_daily_temperatures_reduce(10, '2020-04-01', '2020-05-01', 5);`. Below the query area are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with two columns: 'measurement_date' (date) and 'avg_temperature' (double precision). The table contains five rows of data.

	measurement_date date	avg_temperature double precision
1	2020-04-04	61.833333333333336
2	2020-04-10	48.283333333333333
3	2020-04-16	58.566666666666667
4	2020-04-22	69.283333333333333
5	2020-04-28	65.81428571428572

Рисунок 14 – результат

Вывод:

Все функции были успешно реализованы и протестированы.

Результаты выполнения функций представлены в виде таблиц с соответствующими данными (идентификаторы, названия, координаты, температуры и т.д.).

Примеры работы функций приведены на рисунках в отчете.

Реализованные функции позволяют эффективно извлекать и анализировать информацию, что является важным этапом в управлении базами данных.


```
CREATE OR REPLACE FUNCTION api.get_region_countries_count()
RETURNS TABLE(region_id integer, country_count bigint) AS $$
BEGIN

    RETURN QUERY

    SELECT r.identifier, COUNT(c.identifier)::bigint
    FROM data.region r
    LEFT JOIN data.country c ON r.identifier = c.region
    GROUP BY r.identifier;

END;

$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION api.get_country_cities_count()
RETURNS TABLE(country_id integer, city_count bigint) AS $$
BEGIN

    RETURN QUERY

    SELECT c.identifier, COUNT(ci.identifier)::bigint
    FROM data.country c
    LEFT JOIN data.city ci ON c.identifier = ci.country
    GROUP BY c.identifier;

END;

$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION api.get_regions()
RETURNS TABLE(id integer, name text) AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
    SELECT r.identifier, r.description
```

```
    FROM data.region r;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION api.get_countries(region_id_param  
integer)
```

```
RETURNS TABLE(id integer, name text) AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
    SELECT c.identifier, c.description
```

```
    FROM data.country c
```

```
    WHERE c.region = region_id_param;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION api.get_cities(country_id_param  
integer)
```

```
RETURNS TABLE(id integer, name text) AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
    SELECT ci.identifier, ci.description
```

```
    FROM data.city ci
```

```
        WHERE ci.country = country_id_param;

END;

$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION api.get_city_locations()

RETURNS TABLE(city_id integer, latitude double precision,
longitude double precision) AS $$

BEGIN

    RETURN QUERY

    SELECT c.identifrier, c.latitude, c.longitude

    FROM data.city c;

END;

$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION api.get_coastline_shapes()

RETURNS TABLE(shape_id integer, point_count bigint) AS $$

BEGIN

    RETURN QUERY

    SELECT cl.shape, COUNT(*)::bigint

    FROM data.coastline cl

    GROUP BY cl.shape;

END;

$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION api.get_shape_points(shape_id_param
integer)
```

```
RETURNS TABLE(point_num integer, latitude double precision,
longitude double precision) AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
    SELECT cl.segment, cl.latitude, cl.longitude
```

```
    FROM data.coastline cl
```

```
    WHERE cl.shape = shape_id_param
```

```
    ORDER BY cl.segment;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION
api.get_measurement_time_range(city_id_param integer)
```

```
RETURNS TABLE(min_date date, max_date date) AS $$
```

```
BEGIN
```

```
    RETURN QUERY
```

```
    SELECT MIN(m.mark::date), MAX(m.mark::date)
```

```
    FROM data.measurement m
```

```
    WHERE m.city = city_id_param AND m.temperature > -99;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION api.get_daily_temperatures(
```

```

        city_id_param integer,
        start_date_param date,
        end_date_param date
    )

    RETURNS TABLE(measurement_date date, temperature_celsius double
precision) AS $$
BEGIN
    RETURN QUERY
    SELECT m.mark::date, m.temperature
    FROM data.measurement m
    WHERE m.city = city_id_param
        AND m.mark::date BETWEEN start_date_param AND
end_date_param
        AND m.temperature > -99
    ORDER BY m.mark::date;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION api.get_daily_temperatures_reduce(
    city_id_param integer,
    start_date_param date,
    end_date_param date,
    point_count integer
)

```

```
RETURNS TABLE(measurement_date date, avg_temperature double
precision) AS $$
```

```
DECLARE
```

```
    total_days integer;
```

```
    days_per_point integer;
```

```
    current_start date;
```

```
    current_end date;
```

```
BEGIN
```

```
    -- Вычисляем общее количество дней в диапазоне
```

```
    total_days := end_date_param - start_date_param + 1;
```

```
    days_per_point := total_days / point_count;
```

```
    IF days_per_point < 1 THEN
```

```
        RETURN QUERY
```

```
        SELECT m.mark::date, AVG(m.temperature)
```

```
        FROM data.measurement m
```

```
        WHERE m.city = city_id_param
```

```
            AND m.mark::date BETWEEN start_date_param AND
end_date_param
```

```
            AND m.temperature > -99
```

```
        GROUP BY m.mark::date
```

```
        ORDER BY m.mark::date;
```

```
    ELSE
```

```
        FOR i IN 0..point_count-1 LOOP
```

```
            current_start := start_date_param + (i *
days_per_point);
```

```

        IF i = point_count-1 THEN
            current_end := end_date_param;
        ELSE
            current_end := start_date_param + ((i+1) *
days_per_point) - 1;
        END IF;

        RETURN QUERY

        SELECT
            current_start + (days_per_point/2),
            AVG(m.temperature)

        FROM data.measurement m

        WHERE m.city = city_id_param

            AND m.mark::date BETWEEN current_start AND
current_end

            AND m.temperature > -99;

    END LOOP;

END IF;


RETURN;

END;

$$ LANGUAGE plpgsql;

```