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:

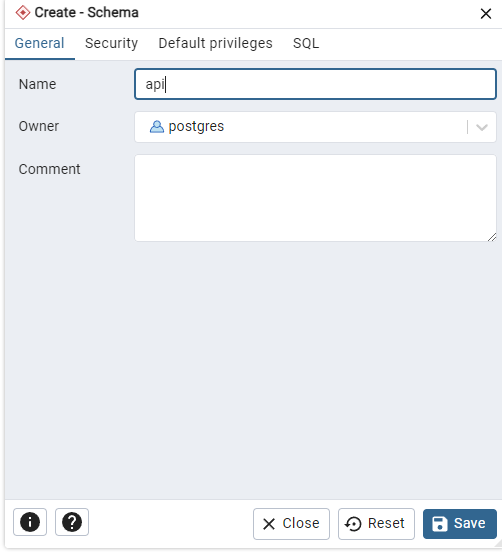


Рисунок 1 – создание схемы

* 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 темперануты 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



Рисунок 2 – фрагмент кода

Убеждаемся, что наши функции были созданы. Рисунок 3

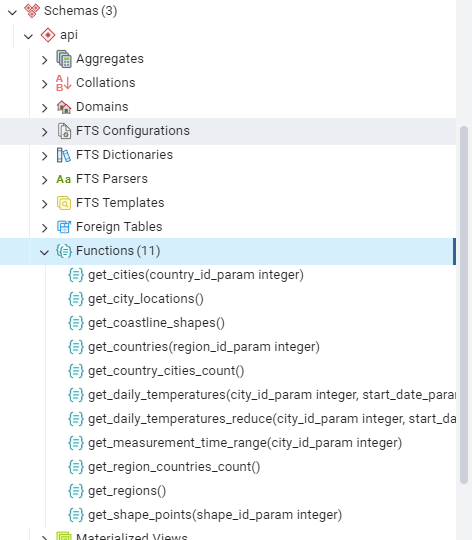


Рисунок 3 – функции

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.

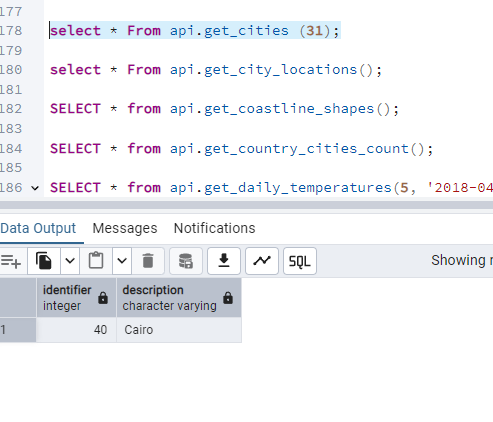


Рисунок 4 – результат

Описание функции get\_countries

Возвращает результат выполнения запрос 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

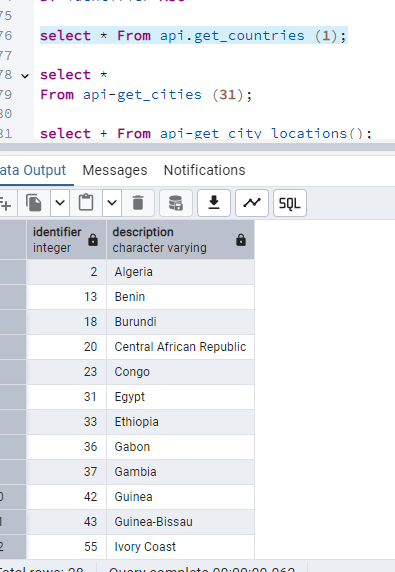


Рисунок 5 – результат

Описание функции get\_city\_locations

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

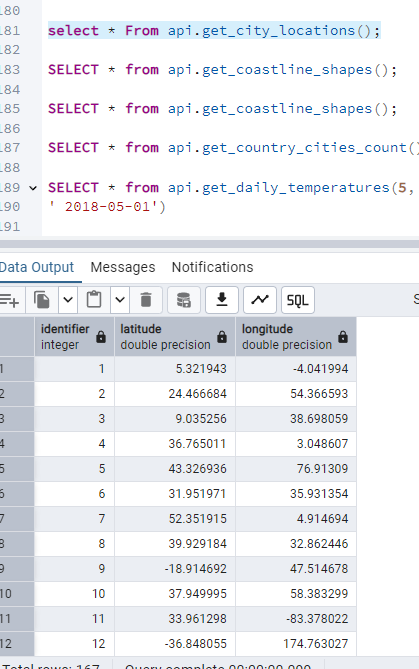


Рисунок 6 – результат

Описание функции get\_coastline\_shapes

Выводит список всех замкнутых фигур, состоящих из соединеных 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.

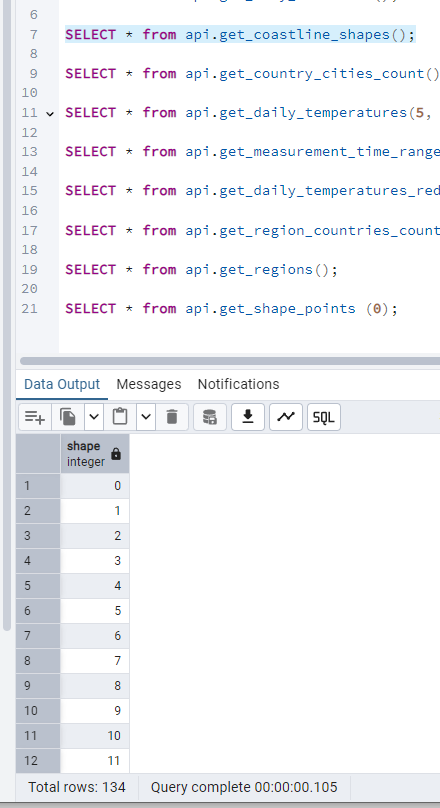


Рисунок 7 – результат

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.

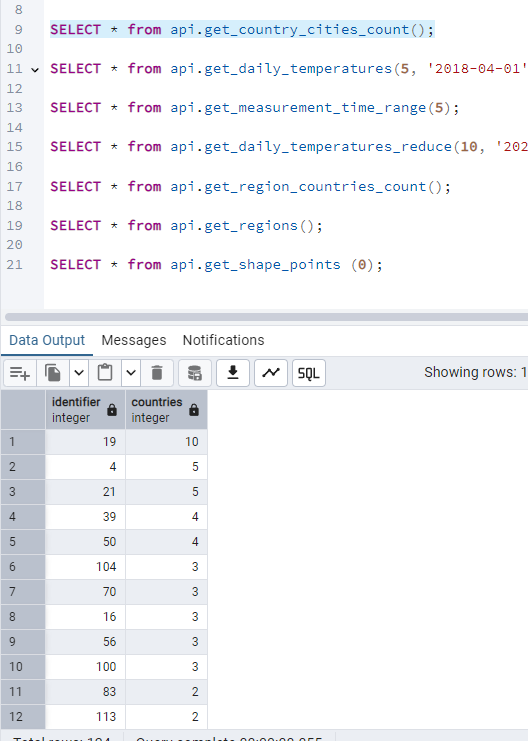


Рисунок 8 – результат

Описание функции get\_daily\_temperatures

Возвращает полный список показаний температуры в определенном городе по его идентификатору за выбраый 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.

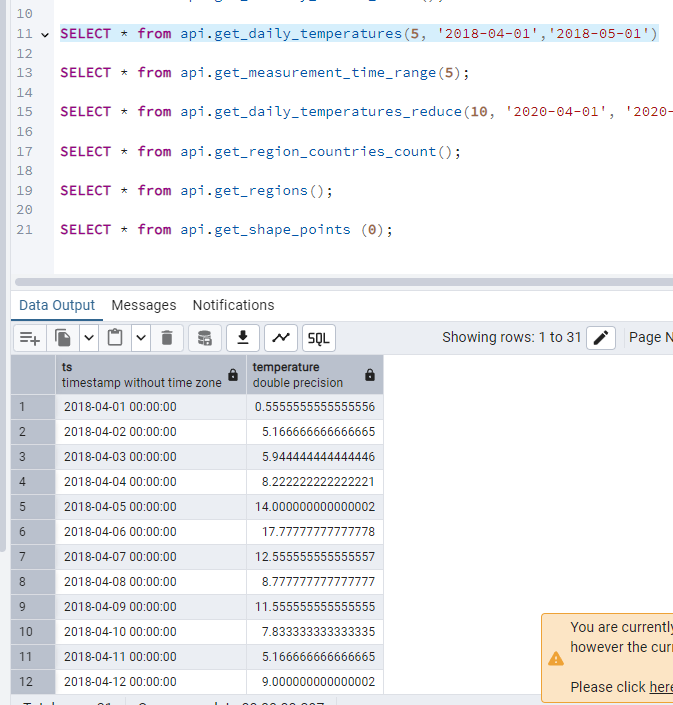


Рисунок 9 – результат

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.

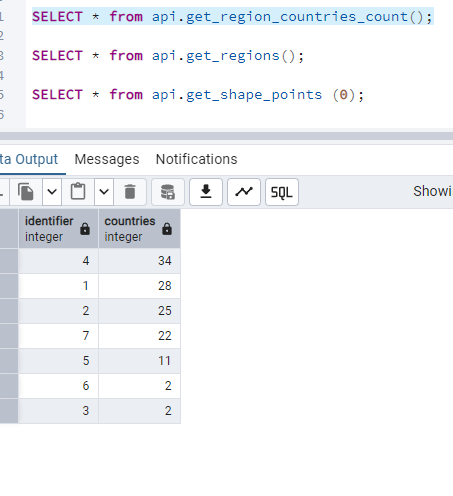


Рисунок 10 – результат

Описание функции get\_regions

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

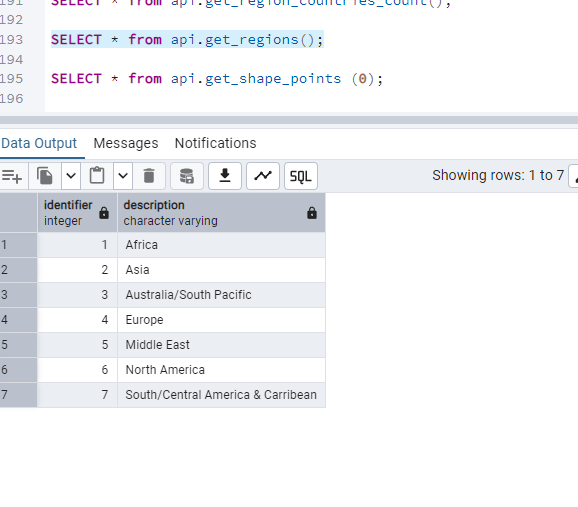


Рисунок 11 – результат

Описание функции get\_shape\_points

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

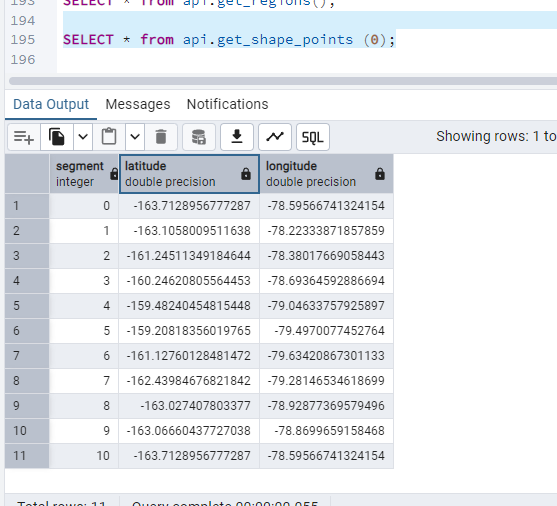


Рисунок 12 – результат

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

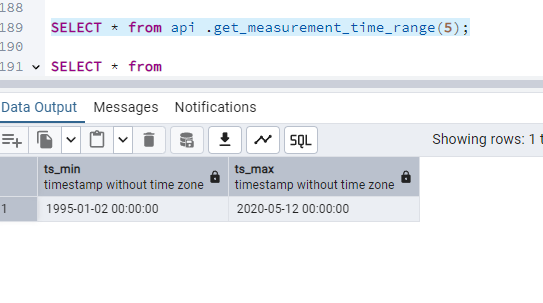


Рисунок 13 – результат

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.

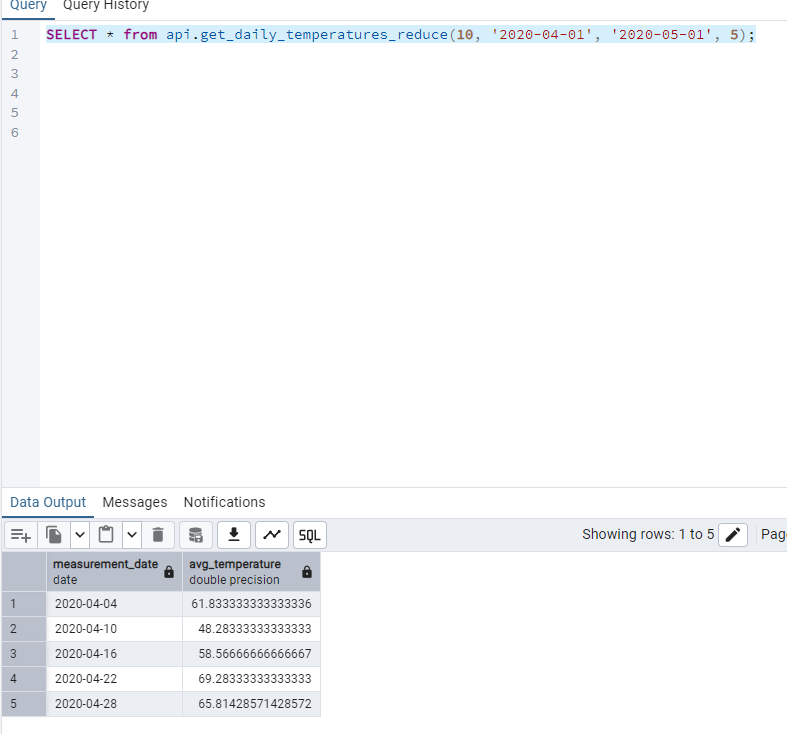


Рисунок 14 – результат

**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.identifier, 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;