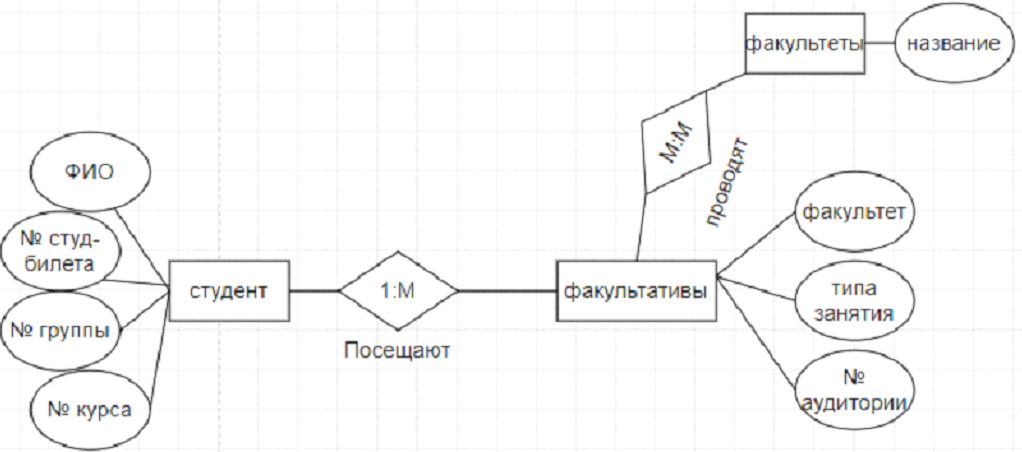
Определение факультативов для студентов.

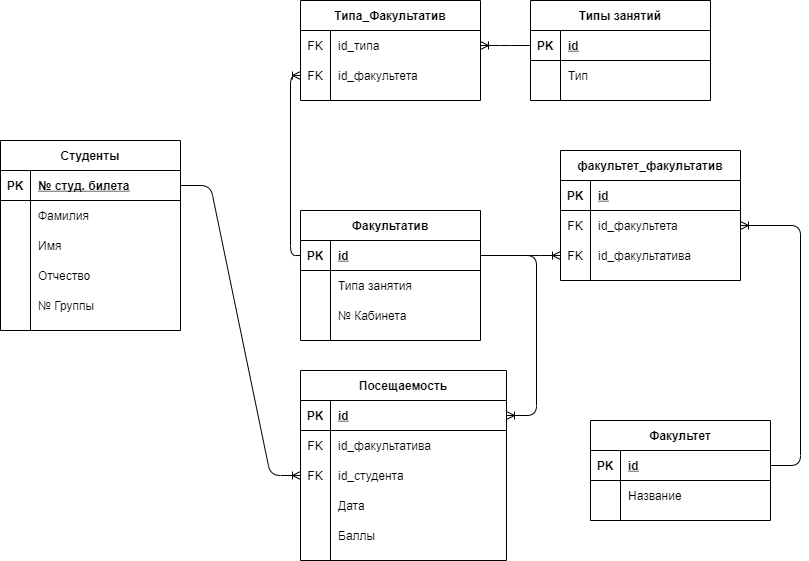
У нас есть доступ к данным студентов (№студ. билета, ФИО, группа, № курса) и

перечню факультативов. Каждый факультатив проводиться только одним из факультетов.

Данные о факультативе хранятся в базе данных (факультет, тип занятия, номер аудитории и т.д.). Также в БД будут записываться данные о

посещении факультативов студентами для выставления баллов.





### Физическое проектирование БД

При создании физической модели используется СУБД PostgreSQL. База данных состоит из 6-ти таблиц.

* Таблицы создаются командой CREATE TABLE. После этого указываются названия столбцов и тип данных для каждого из них.
* После перечисления столбцов и типов данных указывается внешний ключ в формате:

PRIMARY KEY (\*название\_столбца\*).

* Для создания связей используется команда:

\*столбец\* \*тип\_данных\* REFERENCES \* таблица\*(столбец\_таблицы).

* Для внесения данных в таблицу используется команда:

INSERT INTO public.\*название\_таблицы\* (атрибут1, …, атрибут n) VALUES ('значение атрибута 1', ..., 'значение атрибута n').

* Если значение столбца не должно быть пустым, то после типа данных дописывается "NOT NULL".

\*название\_столбца\* \*тип\_данных\* NOT NULL.

* Если значение столбца не должно быть пустым, то после типа данных дописывается "CHECK", который ограничивает значения вносимых данных.

\*название\_столбца\* \*тип\_данных\* CHECK (\*условие\*)

* Если столбец является первичным ключом (PRIMARY KEY), то к нему автоматически применяется ограничитель "NOT NULL".

Используемые типы данных

1. Тип данных "serial" - счетчик (увеличивается на единицу каждый раз). В основном используется для столбцов id
2. Тип данных "character varying" - символьная строка переменной длины. Используется во всех полях с названиями\именами
3. Тип данных "integer" - целочисленное значение. Например номер аудитории или студенческого билета
4. Тип данных "date" - календарная дата, хранящая год, месяц и день. Например дата поступления студента, или дата проведения занятия.
5. Тип данных "timestamp" - календарная дата, хранящая год, месяц, день и время без учета часового пояса.

Запросы и представления

Процесс или команда получения данных из базы данных называется запросом. В SQL запросы формулируются с помощью команды SELECT.

Предложение FROM образует таблицу из одной или нескольких ссылок на таблицы, разделенных запятыми.

Соединения любых типов могут вкладываются друг в друга или объединяться. Если скобки отсутствуют, предложения JOIN обрабатываются слева направо.

Таблицам и ссылкам на сложные таблицы в запросе можно дать временное имя, по которому к ним можно будет обращаться в рамках запроса. Такое имя называется псевдонимом таблицы. Определить псевдоним таблицы можно, написав

FROM *табличная\_ссылка* AS *псевдоним*

Псевдонимы часто применяются для назначения коротких идентификаторов длинным именам таблиц с целью улучшения читаемости запросов.

Предложение WHERE - проверка по условию, где условие ограничения — любое выражение значения, выдающее результат типа boolean.

Предложение GROUP BY группирует строки таблицы, объединяя их в одну группу при совпадении значений во всех перечисленных столбцах. Порядок, в котором указаны столбцы, не имеет значения. В результате наборы строк с одинаковыми значениями преобразуются в отдельные строки, представляющие все строки группы

Строки порожденной входной таблицы, прошедшие фильтр WHERE, можно сгруппировать с помощью предложения GROUP BY, а затем оставить в результате только нужные группы строк, используя предложение HAVING.

Порядок сортировки определяет предложение ORDER BY. Выражениями сортировки могут быть любые выражения, допустимые в списке выборки запроса.

Когда указывается несколько выражений, последующие значения позволяют отсортировать строки, в которых совпали все предыдущие значения. Каждое выражение можно дополнить ключевыми словами ASC или DESC, которые выбирают сортировку соответственно по возрастанию или убыванию. По умолчанию принят порядок по возрастанию.

Указание LIMIT позволяет получить только часть строк из тех, что выдал остальной запрос.

Если указывается число LIMIT, в результате возвращается не больше заданного числа строк (меньше может быть, если сам запрос выдал меньшее количество строк). LIMIT ALL равносильно отсутствию указания LIMIT.

Выражение LIKE возвращает true, если строка соответствует заданному шаблону.

'abc' LIKE 'abc' true

'abc' LIKE 'a%' true

'abc' LIKE '\_b\_' true

'abc' LIKE 'c' false

CREATE VIEW создать представление запроса. Создаваемое представление лишено физической материализации, поэтому указанный запрос будет выполняться при каждом обращении к представлению. (сам запрос пишется после CREATE VIEW имя AS, как обычный запрос)

Агрегатные функции:

* count(\*) - количество входных строк.
* max(выражение) - максимальное значение выражения среди всех входных данных, отличных от NULL.
* min(выражение) - минимальное значение выражения среди всех входных данных, отличных от NULL.
* avg(выражение) - арифметическое среднее для всех входных значений, отличных от NULL.
* sum(выражение) - сумма значений выражения по всем входным данным, отличным от NULL.

Представление 1:

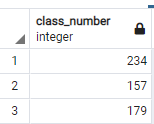
Предназначено для вывода всех неиспользуемых кабинетов.

create view empty\_cabinets as

select class\_number from elective

left join attendance on attendance.id\_elective = elective.id

where id\_elective is null;



Представление 2:

Предназначено для вывода всех студентов (№ билета, имя, фамилия, баллы) с суммой баллов больше ил равным 5

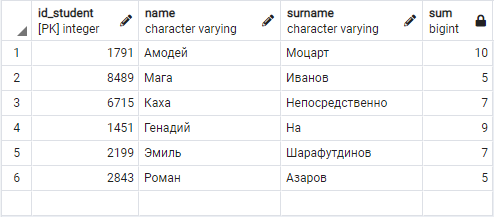
create view smart\_students as

select students.id\_student, students.name, students.surname, sum(attendance.score) from attendance

left join students on students.id\_student = attendance.id\_student

where score > 4

group by students.id\_student;



Представление 3:

Предназначено для вывода полного списка студентов (Фамилия, имя, № билета), отсортированных по фамилии.

create view all\_students as

select id\_student, surname, name from students

order by surname asc;



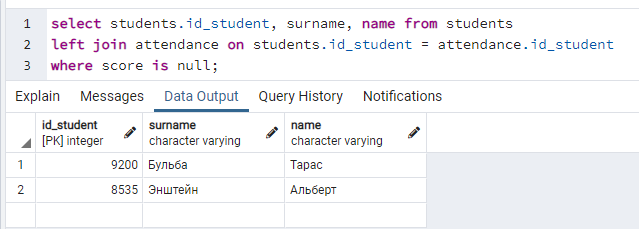
Запрос 1:

Выводит всех студентов, не посетивших ни одного занятия.

select students.id\_student, surname, name from students

left join attendance on students.id\_student = attendance.id\_student

where score is null;



Запрос 2:

Выводит все занятия за 2020 год

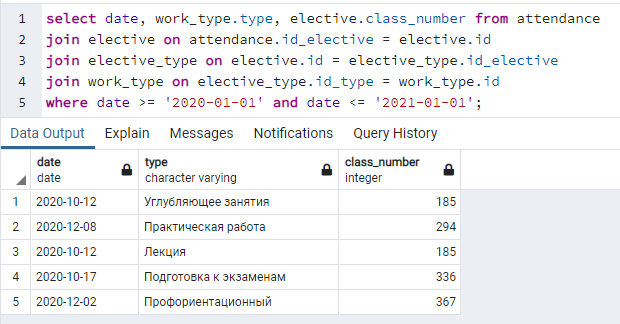
select date, work\_type.type, elective.class\_number from attendance

join elective on attendance.id\_elective = elective.id

join elective\_type on elective.id = elective\_type.id\_elective

join work\_type on elective\_type.id\_type = work\_type.id

where date >= '2020-01-01' and date <= '2021-01-01';

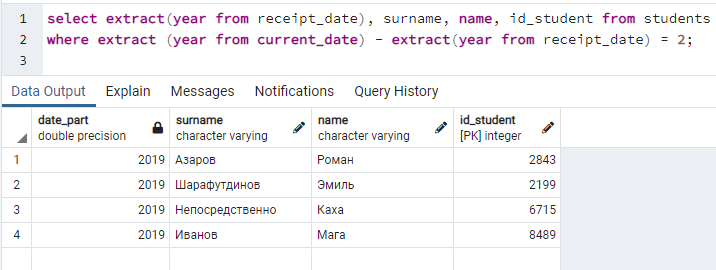


Запрос 3:

Выводит всех студентов второго курса.

select extract(year from receipt\_date), surname, name, id\_student from students

where extract (year from current\_date) - extract(year from receipt\_date) = 2;

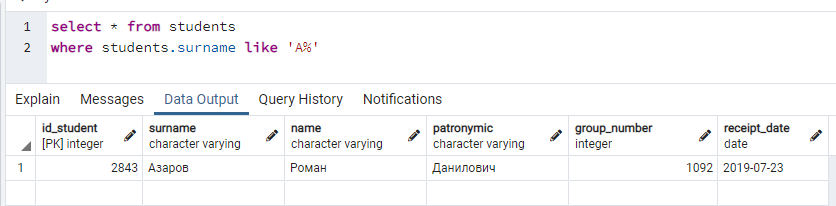


Запрос 4:

Выводит всех студентов с фамилией, начинающейся на букву “А”.

select \* from students

where students.surname like 'А%'



Запрос 5:

Выводит три самых часто используемых кабинета и кол-во проведенных в них занятий.

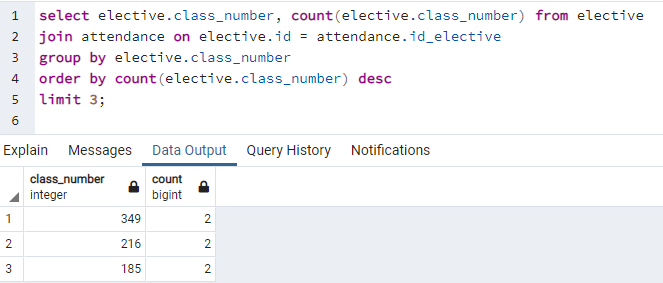
select elective.class\_number, count(elective.class\_number) from elective

join attendance on elective.id = attendance.id\_elective

group by elective.class\_number

order by count(elective.class\_number) desc

limit 3;

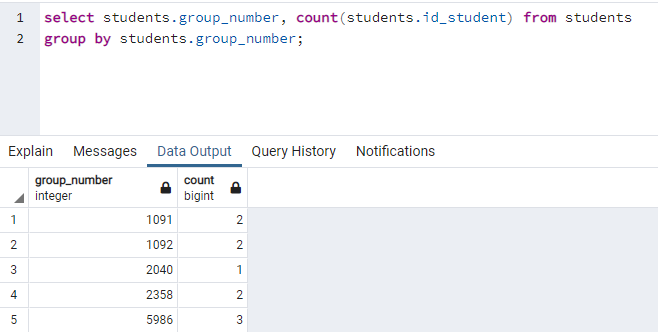


Запрос 6:

Выводит количество студентов в каждой группе.

select students.group\_number, count(students.id\_student) from students

group by students.group\_number;



Запрос 7:

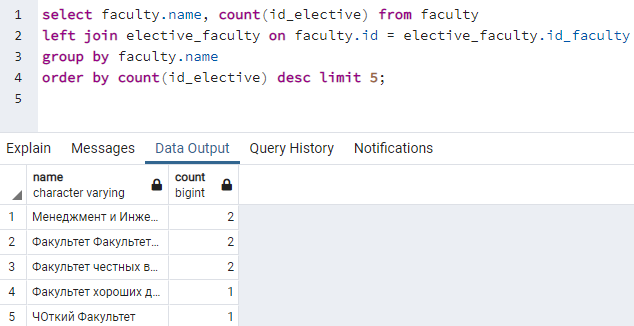
Выводит топ-5 самых активных факультетов (наибольшее число проведенных факультативов).

select faculty.name, count(id\_elective) from faculty

left join elective\_faculty on faculty.id = elective\_faculty.id\_faculty

group by faculty.name

order by count(id\_elective) desc limit 5;



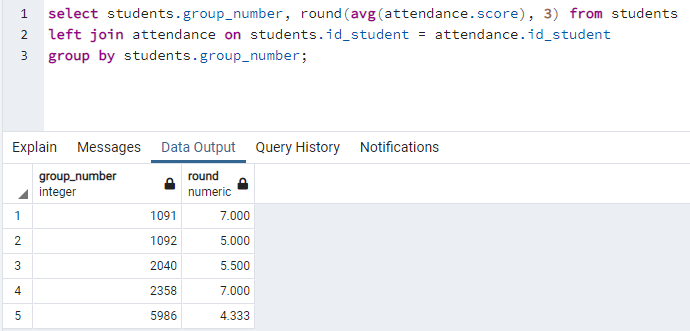
Запрос 8:

Выводит средние баллы студентов по группам. Результат округлен до тысячных.

select students.group\_number, round(avg(attendance.score), 3) from students

left join attendance on students.id\_student = attendance.id\_student

group by students.group\_number;



Запрос 9:

Выводит топ-3 студента по баллам: № билета, фамилию, и сумму баллов.

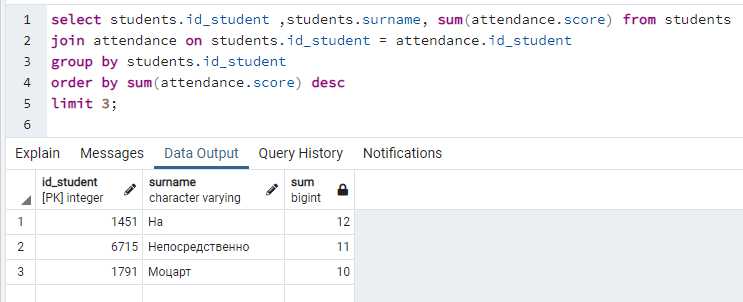
select students.id\_student ,students.surname, sum(attendance.score) from students

join attendance on students.id\_student = attendance.id\_student

group by students.id\_student

order by sum(attendance.score) desc

limit 3;



Запрос 10:

Выводит всех студентов (билет, фамилию, сумму баллов), с общими баллами > 5, которые не посетили ни одного занятия в 2021 год.

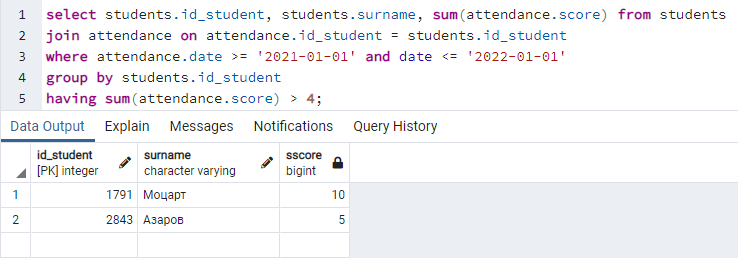
select students.id\_student, students.surname, sum(attendance.score) from students

join attendance on attendance.id\_student = students.id\_student

where attendance.date >= '2021-01-01' and date <= '2022-01-01'

group by students.id\_student

having sum(attendance.score) > 4;



Запрос 11:

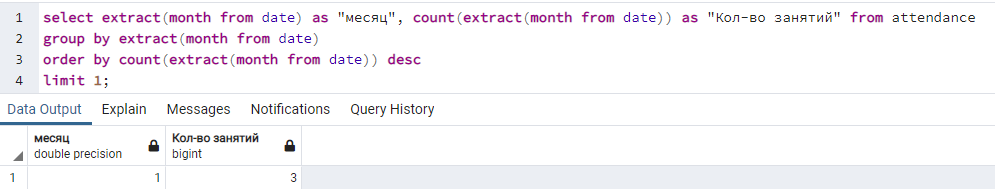
Вывести месяц, в котором было больше всего занятий (за всё время)

select extract(month from date) as "месяц", count(extract(month from date)) as "Кол-во занятий" from attendance

group by extract(month from date)

order by count(extract(month from date)) desc

limit 1;



Запрос 12:

Вывести типы занятий, и их популярность (Тип занятие и общее число посещений).

select work\_type.type, count(elective\_type.id\_type) from attendance

left join elective on attendance.id\_elective = elective.id

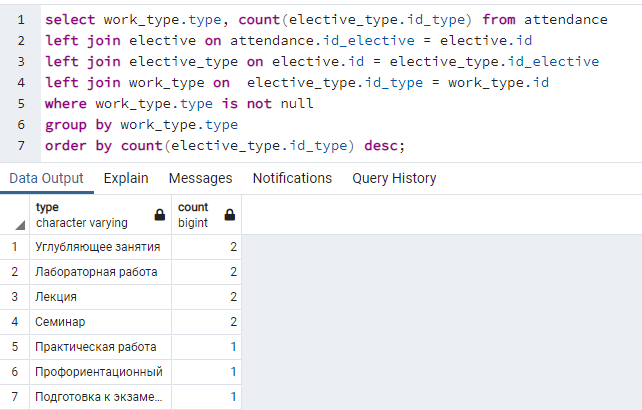
left join elective\_type on elective.id = elective\_type.id\_elective

left join work\_type on elective\_type.id\_type = work\_type.id

where work\_type.type is not null

group by work\_type.type

order by count(elective\_type.id\_type) desc;



Запрос 13:

Вывести всех студентов, которые посещали занятия одного типа более 1 раза.

select students.surname,students.name, students.name, students.id\_student, count(students.id\_student) from attendance

left join students on attendance.id\_student = students.id\_student

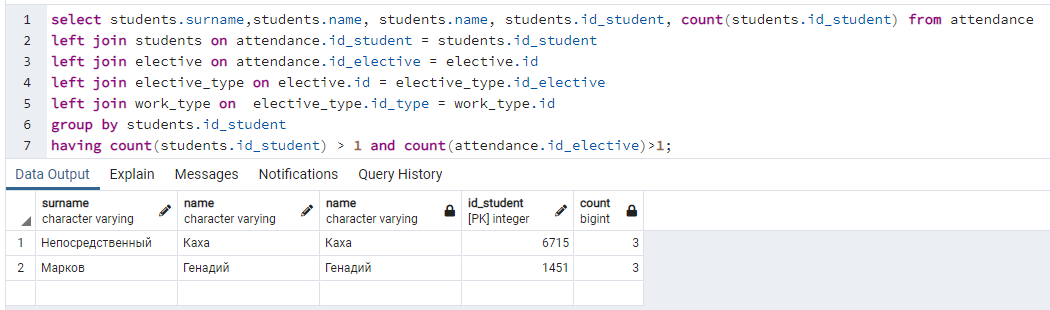
left join elective on attendance.id\_elective = elective.id

left join elective\_type on elective.id = elective\_type.id\_elective

left join work\_type on elective\_type.id\_type = work\_type.id

group by students.id\_student

having count(students.id\_student) > 1 and count(attendance.id\_elective)>1;



Запрос 14:

Выводит топ-5 типов занятий, на которых студенты получают больше всего баллов.

select work\_type.type, sum(score) from attendance

join students on attendance.id\_student = students.id\_student

left join elective on attendance.id\_elective = elective.id

left join elective\_type on elective.id = elective\_type.id\_elective

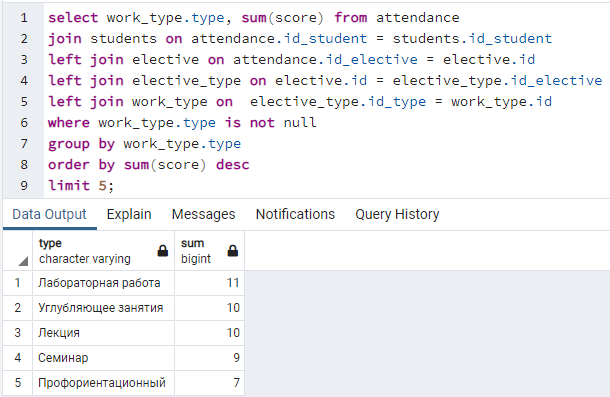
left join work\_type on elective\_type.id\_type = work\_type.id

where work\_type.type is not null

group by work\_type.type

order by sum(score) desc

limit 5;



Запрос 15:

Вывести всех студентов, посетивших факультатив 2021-01-24.

select students.id\_student, students.surname from attendance

left join students on attendance.id\_student = students.id\_student

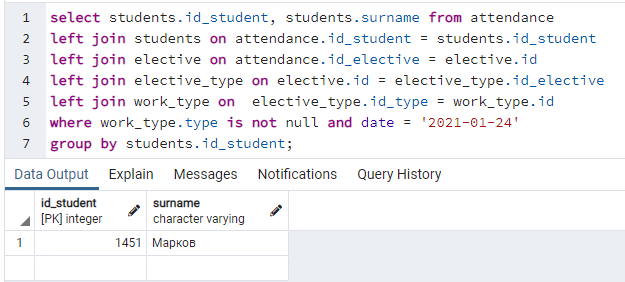
left join elective on attendance.id\_elective = elective.id

left join elective\_type on elective.id = elective\_type.id\_elective

left join work\_type on elective\_type.id\_type = work\_type.id

where work\_type.type is not null and date = '2021-01-24'

group by students.id\_student;



Функции

CREATE [OR REPLACE] FUNCTION <имя функции> (<входные или выходные параметры прописанные через запятую и через пробел тип данных; для входных данных перед именем переменной пишется IN, для выходных OUT>)

RETURNS <тип выходных данных; если они прописаны выше с помощью ключевого слова OUT, RETURNS можно не писать; если функция ничего не возвращает типом данных будет void>

AS

$$

тело функции;

$$

LANGUAGE <язык>;

Команда CREATE FUNCTION определяет новую функцию. CREATE OR REPLACE FUNCTION создаёт новую функцию, либо заменяет определение уже существующей.

Чтобы определить функцию, необходимо иметь право USAGE для соответствующего языка. Если указано имя схемы, функция создаётся в заданной схеме, в противном случае — в текущей. Имя новой функции должно отличаться от имtн существующих функций с такими же типами аргументов в этой схеме. Однако функции с аргументами разных типов могут иметь одно имя (это называется перегрузкой).

Язык, на котором пишется функция указывать нужно обязательно (sql, plpgsql)

1) Создать функцию, которая будет заменять номер студенческого билета на '\*\*\*\*'

create or replace function hide\_student\_id(student\_id int)

returns text

as

$$

select overlay(student\_id::varchar placing '\*\*\*\*' from 1 for 4)

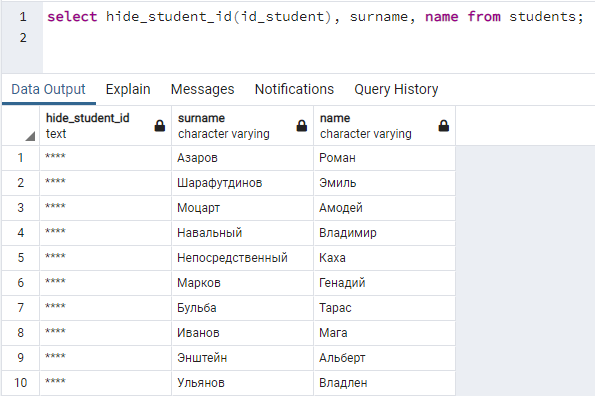
$$

language sql;

--Вызов

select hide\_student\_id(id\_student), surname, name from students;

--



2) Создать функцию, которая будет из полного имени делать инициалы.

create or replace function get\_full\_name(surname text, name text, patronymic text)

returns text

as

$$

select

case

when patronymic is not null then

surname || ' ' || overlay(name placing '.' from 2 for char\_length(name)) || overlay(patronymic placing '.' from 2 for char\_length(patronymic))

when patronymic is null then

surname || ' ' || overlay(name placing '.' from 2 for char\_length(name))

end

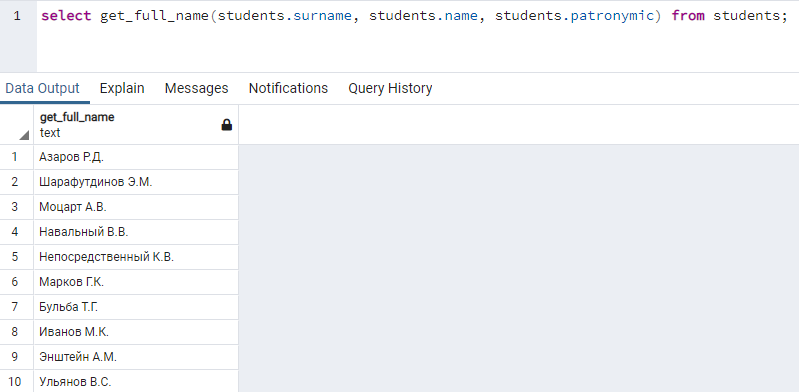
$$

language sql;

--Вызов

select get\_full\_name(students.surname, students.name, students.patronymic) from students;

--



3) Написать функцию, которая будет выводит в одну строкоу фамилию студента и его успеваемость.

create or replace function get\_marks()

returns setof text

as

$$

select

(case

when sum(attendance.score) between 1 and 3 then

students.surname || ': удовлетворительно'

when sum(attendance.score) between 3 and 5 then

students.surname || ': хорошо'

when sum(attendance.score) > 5 then

students.surname || ': отлично'

end)

from students

join attendance on students.id\_student = attendance.id\_student

group by students.id\_student

order by sum(attendance.score) desc;

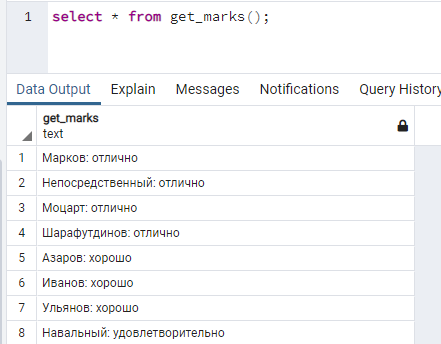
$$

language sql;

--Вызов

select \* from get\_marks();

--



Триггеры

Команда CREATE TRIGGER определяет новый триггер. CREATE OR REPLACE FUNCTION … returns trigger создаёт новую триггерную функцию, либо заменяет определение уже существующей, которая будет выполнятся при срабатывании триггера. Триггерная функция не должна принимать аргументы, а возвращать должна либо new (ключевое слово, означающее новое значение для таблицы) либо null (ничего). синтаксис следующий:

create or replace function <имя>() returns trigger as

$$

begin

return <null\new>;

end;

$$

language plpgsql;

create trigger <имя> before insert on <таблица, для которой создавался триггер>

for each row execute function <имя триггерной функции>();

before - слово, означающее что триггерная функция будет выполнена ДО действия insert (вместо before может быть after - после, а вместо insert могут быть delete и update - удалить или обновить соответственно)

--1) Проверяет номер студенческого билета на четырехзначность.

create or replace function check\_student\_id() returns trigger as

$$

begin

if new.id\_student < 1000 or new.id\_student > 9999 then

raise exception 'id студента должен быть четырехзначным';

end if;

return new;

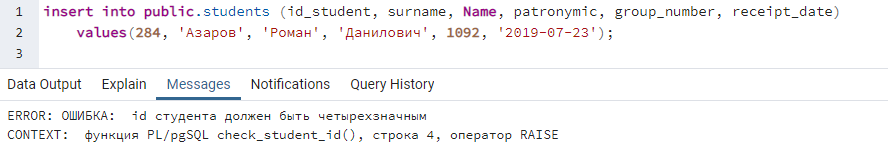
end;

$$

language plpgsql;

create trigger check\_student\_id before insert on students

for each row execute function check\_student\_id();



--2) создать триггер, выводящий сообщение о зачислении студента

create or replace function student\_out() returns trigger as

$$

declare

res varchar;

begin

select into res new.surname || ' ' || overlay(new.name placing '.' from 2 for char\_length(new.name)) ||

overlay(new.patronymic placing '.' from 2 for char\_length(new.patronymic));

raise notice 'Добавлен студент: %', res;

return new;

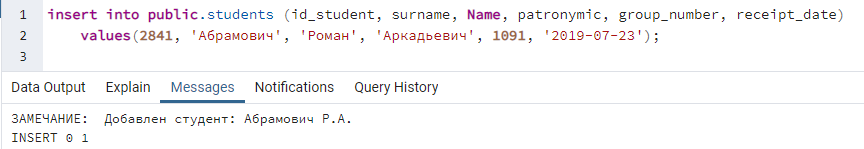
end;

$$

language plpgsql;

create trigger student\_out after insert on students

for each row execute function student\_out();



--3) Создать триггер, проверяющий наличие братьев и сестер в вузе (если студенты из одной семьи, то им полагаются скидки)

create or replace function check\_famaly() returns trigger

as

$$

begin

if (select count(\*) from students where surname = new.surname and patronymic = new.patronymic) > 0

then raise notice 'Возможно, у % есть брат\\сестра', new.name;

end if;

return new;

end;

$$

language 'plpgsql';

create trigger checks before insert on students

for each row execute procedure check\_famaly();

insert into public.students (id\_student, surname, Name, patronymic, group\_number, receipt\_date)

values

(1111, 'Азаров', 'Роман', 'Данилович', 1092, '2019-07-25');

