МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВПО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Методические указания

к самостоятельным и лабораторным работам

по дисциплине «Базы данных»

# «Основы DML-запросов в PostgreSQL»

ВятГУ, 2019

## 1. Цели лабораторной работы

- освоить основные варианты DML-запросов в PostgreSQL;
- научиться создавать SQL-скрипты для заполнения таблиц данными;
- научиться работать с представлениями.

## 2. Задание на лабораторную работу

При выполнении работы следует использовать базу данных, созданную в лабораторной работе №1. Ниже приведены задания, которые необходимо выполнить в ходе лабораторной работы:

- Создать и выполнить SQL-скрипт, который будет заполнять таблицы данными. Нужно добавить не менее 3-5 строк в каждую таблицу.
- Создать представления для нескольких таблиц, в которых собираются данные из самой таблицы и других, на которые она ссылается. Среди представлений обязательно должно быть представление для таблицы, которая используется для связи «многие-ко-многим». Выборка из любого представления должна давать полную и осмысленную информацию по сущностям. Хотя бы одно из представлений должно быть сделано с использованием соединений (join) в запросе.
- Для любой таблицы, содержащей столбец с числовыми данными, создать представление следующего вида, отражающее информацию по этому столбцу (в представлении ровно 3 столбца и 4 строки):

| | | |
|---|---|---|
| Минимальное значение | <значение> | <id записи с минимальным значением> |
| Максимальное значение | <значение> | <id записи с максимальным значением> |
| Среднее значение | <значение> | null |
| Сумма значений | <значение> | null |

## 3. Требования к отчету

Отчет по лабораторной работе должен включать в себя следующие разделы:

- Титульный лист;
- Цели и задания на лабораторную работу;
- Выполненные задания;
- Листинг созданных скриптов (скрипт для заполнения таблиц данными и скрипты, создающие представления) с комментариями;
- Примеры результатов выборок из созданных представлений;
- Вывод по лабораторной работе.

## 4. Методические указания к выполнению

### 4.1.Создание учебной структуры

Для того, чтобы рассмотреть создание скриптов создадим свою собственную БД. Для примера возьмём следующую структуру (рисунок 1).

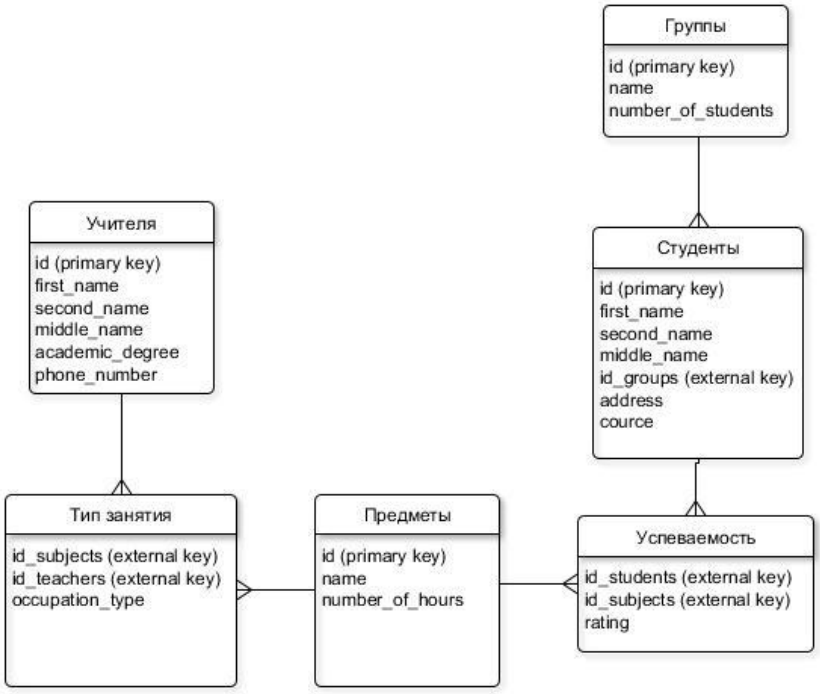


Рисунок 1 - Структура учебной БД

В университете есть студенты и преподаватели. В базе данных все студенты распределены по группам, у каждого есть свой табель успеваемости. Преподаватели проводят определенные занятия по предметам.

У данной БД будет 6 таблиц: «Студенты», «Группы», «Успеваемость», «Предметы», «Преподаватели», промежуточная таблица «Тип проводимого занятия».

Ниже приведем скрипт, создающий учебную БД. Все таблицы размещаются в схеме public.

```
create table groups (
        id                      bigserial primary key,
        name                    varchar(30) not null,
        number_of_students      int not null default 0 check(number_of_students >= 0 and
                                number_of_students <= 30)
);

create table students (
        id                      bigserial primary key,
        first_name              varchar(30) not null,
        second_name             varchar(30) not null,
        middle_name             varchar(30),
        id_groups               bigint not null references groups(id),
```

```
    address                 varchar(100),
    cource                  int check (cource >= 1 and cource <=5)
);

create table subjects (
    id                      bigserial primary key,
    name                    varchar(50) not null,
    number_of_hours         int not null check(number_of_hours > 0 and
                            number_of_hours <= 10)
);

create table progress (
    id_students             bigint not null references students(id),
    id_subjects             bigint not null references subjects(id),
    rating                  int check (rating >= 2 and rating <=5)
);

create table teachers (
    id                      bigserial primary key,
    first_name              varchar(30) not null,
    second_name             varchar(30) not null,
    middle_name             varchar(30),
    academic_degree         varchar(50),
    phone_number            varchar(20)
);

create type occupation_type_enum as enum (
    'lecture', 'laboratory work', 'practical lesson', 'seminar'
);

create table type_of_occupation (
    id_subjects             bigint references subjects(id),
    id_teachers             bigint references teachers(id),
    occupation_type         occupation_type_enum not null default 'lecture'
);
```

## 4.2.Создание SQL-скрипта, заполняющий таблицу данными.

### 4.2.1.Команда INSERT

Для добавления данных применяется команда INSERT, которая имеет следующий формальный синтаксис:

```
INSERT INTO имя_таблицы (столбец1, столбец2, ... столбецN)
VALUES (значение1, значение2, ... значениеN)
```

После INSERT INTO идет имя таблицы, затем в скобках указываются все столбцы через запятую, в которые надо добавлять данные. И в конце после слова VALUES в скобках перечисляются добавляемые значения.

Попробуем добавить в базу данных запись:

```
insert into students(first_name, second_name, middle_name,id_groups,
address, cource)
values
('Николай','Иванов','Олегович', 11, NULL, NULL),
```

*('Станислав','Сидоров','Ивандович', 12, NULL, NULL),*
*('Иван','Солженицин','Алексеевич', 12, NULL, NULL)*

### 4.2.2. Создание SQL-скрипта

При вставке большого количества данных со сложными зависимостями с помощью команды INSERT, изученной в прошлом пункте, могут возникнуть проблемы и неудобства. Для решения такой проблемы на помощь приходят SQL-скрипты на языке PL/pgSQL. Ниже приведена общая структура SQL-скрипта на языке PL/pgSQL:

```
do language plpgsql $$
declare
   -- Переменные, хранящие id новых записей
begin
   --Добавляем вставку в таблицу
   insert into …
         values …
         returning … into …
end;
$$;
```

RETURNING:

Иногда бывает полезно получать данные из модифицируемых строк в процессе их обработки. Это возможно с использованием предложения RETURNING, которое можно задать для команды INSERT.

В команде INSERT данные, выдаваемые в RETURNING, образуются из строки в том виде, в каком она была вставлена. Это не очень полезно при простом добавлении, так как в результате будут получены те же данные. Но это может быть очень удобно при использовании вычисляемых значений по умолчанию.

Например, если в таблице есть ключевой столбец типа serial, в котором генерируются уникальные идентификаторы, команда RETURNING может возвратить идентификатор, назначенный новой строке, и с помощью INTO присвоить его некоторой перменной:

```
insert into groups(name, number_of_students)
        values ('Овечкин',5)
        returning id into _id_groups;
```

При создании простых SQL-скриптов для заполнения таблиц данными главной проблемой является поддержание ссылочной целостности.

В примере это можно наблюдать между таблицами «учителя» и «студенты». Между ними есть связочная таблица «тип занятия». Для того чтобы заполнить ее необходимо знать ID двух основных таблиц.

Вручную делать это неправильно! Конечно можно сделать и вручную, но читать и модифицировать такой скрипт тяжело.

Для решения проблемы напишем небольшой скрипт с командой INSERT:

```
do language plpgsql $$
declare
   -- Переменные, хранящие id новых записей
   --Хранит ID студентов
   _id_students bigint;
   --Хранит ID предметов
   _id_subjects bigint;
   --Хранит ID учителей
   _id_teachers bigint;
   --Хранит ID групп
   _id_groups bigint;
begin
   --Добавляем группу
   insert into groups(name, number_of_students)
         values ('Овечкин',5)
         returning id into _id_groups;

   --Добавляем студента
   insert into students(first_name, second_name, middle_name, id_groups, address, cource)
         VALUES ('Иванов', 'Иван', 'Ивынович', _id_groups, 'Киров, ул. Московская, д. 36', 1)
         returning id into _id_students;

   --Добавляем предмет
   insert into subjects(name, number_of_hours)
         VALUES ('Астрономия', 2)
         returning id into _id_subjects;

   --Добавляем учителей
   insert into teachers(first_name, second_name, middle_name,academic_degree,phone_number)
            VALUES ('Логинов', 'Андрей', 'Павлович', 'Професор', NULL)
            returning id into _id_teachers;

   --Добавляем данные в связующие таблицы
   insert into progress (id_students, id_subjects, rating)
         values (_id_students, _id_subjects, 4);

   insert into type_of_occupation(id_subjects, id_teachers, occupation_type)
         values(_id_subjects, _id_teachers, 'Лекция');

end;
$$;
```

## 4.3. Создание представления (view)

### 4.3.1. Команда SELECT

Для извлечения данных из БД применяется команда SELECT. В упрощенном виде она имеет следующий синтаксис:

```
SELECT список_столбцов FROM имя_таблицы
```

Приведем несколько основных пример, которые показывают основные возможности SELECT. Попробуем получить данные из таблиц:

```
--Выбираем все данные из таблицы students
SELECT * FROM students
```

| | id bigint | first_name character varying (30) | second_name character varying (30) | middle_name character varying (30) | id_groups bigint | address character varying (100) | cource integer |
|---|---|---|---|---|---|---|---|
| 1 | 11 | Николай | Иванов | Олегович | 11 | [null] | [null] |
| 2 | 12 | Станислав | Сидоров | Ивандович | 12 | [null] | [null] |
| 3 | 13 | Иван | Солженицин | Алексеевич | 12 | [null] | [null] |
| 4 | 14 | Петр | Криволапов | Александрович | 13 | [null] | [null] |
| 5 | 15 | Сергей | Кривошеев | Алексеевич | 15 | [null] | [null] |

```
--Выбираем только first_name из таблицы students
SELECT first_name FROM students
```

| | first_name character varying (30) |
|---|---|
| 1 | Николай |
| 2 | Станислав |
| 3 | Иван |
| 4 | Петр |
| 5 | Сергей |

```
--Выбираем только first_name и называем колонку my_name  из таблицы students
SELECT first_name as my_name FROM students
```

| | my_name character varying (30) |
|---|---|
| 1 | Николай |
| 2 | Станислав |
| 3 | Иван |
| 4 | Петр |
| 5 | Сергей |

Если добавить WHERE после FROM, то можно отфильтровать запрос по булевому выражению. Любая строка, не удовлетворяющая этому условию, исключается из результата. Приведем примеры использования предложения:

```
--Выбираем только first_name и называем колонку my_name  из таблицы students
SELECT first_name as my_name FROM students s WHERE s.id >17
```

| | my_name<br>character varying (30) |
|---|---|
| 1 | Николай |
| 2 | Павел |
| 3 | Павел |
| 4 | Иванов |

Кроме WHERE есть другие команды для организации выборки данных в таблице – order by, limit.

ORDER BY: используется для сортировки строк в указанном порядке – порядок сортировки asc – по возрастанию, desc – по убыванию. Если порядок не будет указан, то будет принято по умолчанию asc.

LIMIT: ограничивает максимальное количество строк, которое будет возвращено одним запросом.

### 4.3.2. Команда WITH

WITH предоставляет способ написания вспомогательных операторов для использования в более больших запросах. Эти операторы, которые часто называются как «Общие Табличные Выражения», могут быть задуманы как определяющие временные таблицы, которые существуют только для данного запроса.

WITH можно использовать с SELECT или INSERT.

```
with
   temp_table as (select * from subjects where number_of_hours > 5)
select * from temp_table;
```

| | id<br>bigint | name<br>character varying (50) | number_of_hours<br>integer |
|---|---|---|---|
| 1 | 2 | Математика | 8 |
| 2 | 4 | Информатика | 6 |
| 3 | 6 | Программирование | 6 |
| 4 | 7 | Базы данных | 6 |
| 5 | 8 | Компьютерная графика | 6 |

### 4.3.3. Команда JOIN

Чтобы соединить две таблицы используется оператор JOIN или INNER JOIN. Он представляет так называемое внутреннее соединение.

```
SELECT столбцы
FROM таблица1
[INNER] JOIN таблица2
ON условие1
[[INNER] JOIN таблица3
```

ON условие2]

После оператора JOIN идет название второй таблицы, данные которой надо добавить в выборку. Перед JOIN можно указывать необязательный оператор INNER. Его наличие или отсутствие ни на что не влияет. Далее после ключевого слова ON указывается условие соединения. Это условие устанавливает, как две таблицы будут сравниваться. Как правило, для соединения применяется первичный ключ главной таблицы и внешний ключ зависимой таблицы.

```
select  s.first_name, s.second_name, s.middle_name,
      s.cource, g.name
from students s join groups g on g.id = s.id_groups;
```

| | first_name character varying (30) | second_name character varying (30) | middle_name character varying (30) | cource integer | name character varying (30) |
|---|---|---|---|---|---|
| 1 | Николай | Иванов | Олегович | [null] | ИВТ |
| 2 | Иван | Солженицин | Алексеевич | [null] | ИКТ |
| 3 | Станислав | Сидоров | Ивандович | [null] | ИКТ |
| 4 | Петр | Криволапов | Александрович | [null] | ИСТ |
| 5 | Александр | Столбов | Григорович | [null] | ИБ |

### 4.3.4.Представление (view)

Представление (view) — это виртуальная таблица, содержимое которой определяется запросом SELECT. Запрос будет выполняться при каждом обращении к представлению. Тем не менее, представления очень удобны для работы с большими и часто повторяющимися запросами.

При разработке приложений делаются представления для многих таблиц, которые имеют внешние ключи. Информация из связывающих таблиц бесполезна для отображения в приложении, т.к. в ней нет зачастую нет существенной информации.

Исходя из информации, полученной в предыдущих подпунктах, можно написать своей представление по следующему образцу:

```
create or replace view «имя представления» as
  select ...
    from ...
      where ...;
```

Например, сделаем представление по данным таблицам:

```
create or replace view teacher_subject_v as
  select s.name, t.first_name
from type_of_occupation st, subjects s, teachers t
where st.id_subjects = s.id and st.id_teachers = t.id;
```

| | name<br>character varying (50) | first_name<br>character varying (30) |
|---|---|---|
| 1 | Математика | Александр |
| 2 | Программирование | Станислав |
| 3 | Информатика | Александр |
| 4 | Базы данных | Иван |
| 5 | Английский язык | Иван |

```
create or replace view students_subjects_v as
  select  s.first_name, s.second_name, s.middle_name,
      s.cource, g.name
from students s join groups g on g.id = s.id_groups;
```

| | first_name<br>character varying (30) | second_name<br>character varying (30) | middle_name<br>character varying (30) | cource<br>integer | name<br>character varying (30) |
|---|---|---|---|---|---|
| 1 | Николай | Иванов | Олегович | [null] | ИВТ |
| 2 | Иван | Солженицин | Алексеевич | [null] | ИКТ |
| 3 | Станислав | Сидоров | Ивандович | [null] | ИКТ |
| 4 | Петр | Криволапов | Александрович | [null] | ИСТ |
| 5 | Александр | Столбов | Григорович | [null] | ИБ |

## 4.4. Создать представление, отражающее информацию при помощи агрегатных функций

### 4.4.1. Агрегатные функции

Агрегатные функции используются, когда нужно получить результат выполнения какой-либо функции (сумма, среднее значение, подсчет количества и т.д.).

В PostgreSQL существует большое количество различных агрегатных функций. Если необходимо получить результат выполнения агрегатной функции над разными группами строк из выборки, необходимо использовать предложение GROUP BY. В нем перечисляются все столбцы, которые перечисляются в SELECT, но не участвуют в какой-либо агрегатной функции. На основе совпадения значений в указанных столбцах строки объединяются в группы.

| Функция | Типы аргумента | Тип результата | Описание |
|---|---|---|---|
| avg(выражение) | smallint, int, bigint, real, double<br>precision, numericили int<br>erval | numeric для любых целочисленных аргументов, double precision для аргументов с плавающей точкой, в противном случае тип | арифметическое среднее для всех входных значений |

| | | | |
|---|---|---|---|
| | | данных аргумента | |
| count(*) | | bigint | количество входных строк |
| count(выражение) | any | bigint | количество входных строк, для которых значение выражения не равно NULL |
| max(выражение) | любой числовой, строковый, сетевой тип или тип даты/времени, либо массив этих типов | тот же, что и тип аргумента | максимальное значение выражения среди всех входных данных |
| min(выражение) | любой числовой, строковый, сетевой тип или тип даты/времени, либо массив этих типов | тот же, что и тип аргумента | минимальное значение выражения среди всех входных данных |
| string_agg(выражение, разделитель) | (text, text) или (bytea, bytea) | тот же, что и типы аргументов | входные данные складываются в строку через заданный разделитель |
| sum(выражение) | smallint, int, bigint, real, double precision, numeric, interval или money | bigint для аргументов smallint или int, numeric для аргументов bigint, и тип аргумента в остальных случаях | сумма значений выражения по всем входным данным |

Следует заметить, что за исключением count, все эти функции возвращают NULL, если для них не была выбрана ни одна строка. В частности, функция sum, не получив строк, возвращает NULL, а не 0, как можно было бы ожидать.

### 4.4.2. Создание представления с агрегатными функциями

Итогом всей лабораторной работы будет создание представления с использованием внутри агрегатной функции.

Ниже будет приведен пример кода для данной структуры, описанной вначале лабораторной:

```
create or replace view groups_v as
    (select 'Минимальное значение', number_of_students, id from groups
ORDER BY number_of_students desc LIMIT 1)
    union all
    (select 'Максимальное значение', number_of_students, id from groups
ORDER BY number_of_students LIMIT 1)
```

```
    union all
select 'Среднее значение', avg(g.number_of_students), null from groups g
    union all
select 'Сумма значений', sum(number_of_students), null from groups;
```

После того, как представление создано, можно проверить правильность его выполнения с помощью команды SELECT:

```
select * from groups_v
```

| | ?column? text | number_of_students numeric | id bigint |
|---|---|---|---|
| 1 | Минимальное значение | 30 | 17 |
| 2 | Максимальное значение | 5 | 18 |
| 3 | Среднее значение | 18.2500000000000000 | [null] |
| 4 | Сумма значений | 146 | [null] |