**Лабораторная работа №2.**

**Основы DML-запросов в PostgreSQL**

*Цели лабораторной работы:*

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

*Задание на лабораторную работу:*

При выполнении работы используется БД, созданная в лабораторной работе №1.

Нужно выполнить следующие шаги:

1. Создать и выполнить SQL-скрипт, который будет заполнять таблицы данными. Нужно добавить не менее 3-5 строк в каждую таблицу.
2. Создать представления для нескольких таблиц, в которых собираются данные из самой таблицы и других, на которые она ссылается. 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.
3. 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 |

Отчет по лабораторной работе должен содержать:

* созданные скрипты с комментариями (скрипт для заполнения таблиц данными и скрипты, создающие представления);
* примеры результатов выборок из созданных представлений.

*Выполнение лабораторной работы:*

Ниже показан скрипт, который был создан в процессе выполнения лабораторной работы 1. Выполнение данной лабораторной работы выполним с его использованием.

*--Создание таблицы ГРУППЫ***create table public**.group(  
 **id bigserial primary key**,  
 **name varchar**(30) **not null unique**,  
 **number\_of\_students int not null check** (**number\_of\_students** >= 1)  
);  
  
*--Создание таблицы СТУДЕНТЫ***create table public**.student(  
 **id bigserial primary key**,  
 **name varchar**(30) **not null**,  
 **surname varchar**(30) **not null**,  
 **middle\_name varchar**(30),  
 **id\_group bigint not null references public**.group(**id**),  
 **address varchar**(100),  
 **cource int check** (**cource** >= 1 **and cource** <=5)  
);  
  
*--Создание таблицы ПРЕДМЕТЫ***create table public**.subject(  
 **id bigserial primary key**,  
 **name varchar**(50) **not null unique**,  
 **number\_of\_hours int not null check** (**number\_of\_hours** > 0)  
);  
  
*--Создание таблицы УСПЕВАЕМОСТЬ***create table public**.progress(  
 **id\_student bigint not null references public**.student(**id**),  
 **id\_subject bigint not null references public**.subject(**id**),  
 **rating int check** (**rating** >= 2 **and rating** <=5)  
);  
  
*--Создание типа перечисления для обозначения должности преподавателя***create type public**.position\_type\_enum **as enum** (  
 **'docent'**, **'professor'**, **'senior lecturer'**, **'instructor'**);  
  
*--Создание таблицы ПРЕПОДОВАТЕЛИ***create table public**.teacher(  
 **id bigserial primary key**,  
 **name varchar**(30) **not null**,  
 **surname varchar**(30) **not null**,  
 **middle\_name varchar**(30),  
 **position public**.**position\_type\_enum not null default 'instructor'**,  
 **phone\_number varchar**(30)  
);  
  
*--Создание типа перечисления для обозначения типа проводимого занятия***create type public**.occupation\_type\_enum **as enum** (  
 **'lecture'**, **'laboratory work'**, **'practical lesson'**, **'seminar'**);  
  
*--Создание таблицы ТИП ЗАНЯТИЯ***create table public**.type\_of\_occupation(  
 **id\_subject bigint not null references public**.subject(**id**),  
 **id\_teacher bigint not null references public**.teacher(**id**),  
 **occupation\_type public**.**occupation\_type\_enum not null  
 default 'lecture'**);

1. Создать и выполнить SQL-скрипт, который будет заполнять таблицы данными. Нужно добавить не менее 3-5 строк в каждую таблицу.

***Команда insert***

Команда insert используется для вставки строк в таблицу. За один запрос можно вставить ноль и более строк. Она может как добавлять строки с указанными явно значениями, так и добавлять строки, которые были получены из дополнительного запроса (см. [пункт «Подзапросы и предложение with»](#Заголовок)).

Здесь будут рассмотрены базовые способы применения команды insert. *Подробнее о синтаксисе команды insert можно прочитать в документации.*

-- Пример

*-- Вставляет новую строку в customer, id проставляется автоматически.*

*-- Поле команды вставки insert into указывается имя таблицы, в скобках перечисляются заполняемые поля, затем после ключевого слова values указываются значения этих полей.*

**insert into** group(**name, number\_of\_students**)

**values** (**'ИВТб-3302-02-00'**, 20);

-- Пример

*-- Если указать id явно, как в примере ниже, он будет установлен в 15.  
-- При этом последовательность id никак не сдвинется!  
-- При вставке следующей строки без id случится конфликт, когда последовательность дойдет до 15.  
-- Поэтому никогда не надо ставить руками id, которые имеют тип bigserial!   
-- Если очень нужно так сделать, то после этого обязательно проверить состояние последовательности и сдвинуть её при необходимости.***insert into** group(**id, name, number\_of\_students**)

**values** (15, **'ИВТб-3302-02-00'**, 20);

-- Пример

*-- За один раз можно указать несколько строк для вставки*

**insert into** group(**name, number\_of\_students**)

**values** (**'ИВТб-3301-01-00'**, 15),

(**'ИВТб-3302-02-00'**, 20),

(**'ИВТб-4301-01-00'**, 12);

-- Пример 4

*-- Два запроса ниже аналогичны, т.к. у teacher есть значение по умолчанию.  
-- Если вместо default в первом запросе указать конкретное значение, то будет вставлено оно.***insert into** teacher(**first\_name**, **second\_name**, **middle\_name, position, phone\_number**)

**values** (**'Николай'**, **'Сергеевич'**, **'Котов'**, **default**, **'89530000000'**);  
**insert into** teacher(**first\_name**, **second\_name**, **middle\_name, phone\_number**) **values** (**'Николай'**, **'Сергеевич'**, **'Котов'**, **'89530000000'**);

При создании скриптов для заполнения таблиц данными главной проблемой является поддержание ссылочной целостности. Вручную проставлять числовые значения столбцов с внешними ссылками неправильно: практически никогда нельзя с уверенность сказать, какие первичные ключи будут сгенерированы в таблице, на которые делаются ссылки. Даже если это можно сделать, читать и модифицировать такой скрипт тяжело.

Один из вариантов решения проблемы – использование запросов insert с предложениями with (см. [пример 69](#Пример)).

Тем не менее, при вставке большого количества данных со сложными зависимостями это может быть неудобным. Другой вариант – использовать скрипт на PL/pgSQL. Работа с PL/pgSQL подробнее рассматривается в лабораторной работе №3. В рамках текущей лабораторной работы можно воспользоваться примером 5.

-- Пример

**do $$  
declare**

*-- Здесь описываются переменные, которые будут хранить id новых записей*groupID **bigint**;  
  
**begin**

*-- Используя слово into в предложении returning, можно записывать результат в переменную*

**insert into** group(**name**, **number\_of\_students**)  
**values** (**'ИВТб-3301-01-00'**, 15)  
**returning id into** groupID;

**insert into** student(**first\_name**, **second\_name**, **middle\_name**, **id\_group**, **address**, **cource**)

**values** (**'Котов'**, **'Сергеевич'**, **'Николай'**, groupID, **'г. Киров ул. Ломоносова 16а'**, 3)**;**

**insert into** student(**first\_name**, **second\_name**, **middle\_name**, **id\_group**, **address**, **cource**)

**values** (**'Олоев'**, **'Петрович'**, **'Василий'**, groupID, **'г. Киров ул. Свободы 4'**, 3);

**insert into** group(**name**, **number\_of\_students**)  
**values** (**'ИВТб-3302-02-00'**, 20)  
**returning id into** groupID;

**insert into** student(**first\_name**, **second\_name**, **middle\_name**, **id\_group**, **address**, **cource**)

**values** (**'Зотов'**, **'Геогриевич'**, **'Олег'**, groupID, **'г. Киров ул. Воровского 122'**, 3);

**insert into** group(**name**, **number\_of\_students**)  
**values** (**'ИВТб-4301-01-00'**, 12)  
**returning id into** groupID;

**insert into** student(**first\_name**, **second\_name**, **middle\_name**, **id\_group**, **address**, **cource**) **values** (**'Романов'**, **'Владимирович'**, **'Константин'**, groupID, **'г. Киров ул. Ленина 52'**, 4);

**end**;  
**$$ language** plpgsql;

1. Создать представления для нескольких таблиц, в которых собираются данные из самой таблицы и других, на которые она ссылается. 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.

***Команда select***

Команда select используется для получения строк из таблиц. Она обладает очень гибким синтаксисом и огромным количеством возможностей, позволяя получать данные из некоторого множества таблиц (возможно, пустого), фильтруя, сортируя, объединяя и предварительно обрабатывая их различными способами.

В рамках этого раздела будут рассмотрены базовые способы применения команды select. *Подробнее о синтаксисе команды select можно прочитать в документации.*

-- Пример

*-- Получить все строки из таблицы student*

*-- После ключевого слова select указываются столбцы, которые необходимо получить, после ключевого слова from указываются имена таблиц, из которых будут выбраны столбцы***select** *\** **from** student;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **id** | **name** | **surname** | **middle\_name** | **id\_group** | **address** | **cource** |
| *1* | *Николай* | *Котов* | *Сергеевич* | *4* | *г. Киров*  *ул. Ломоносова 16а* | *3* |
| *2* | *Василий* | *Олоев* | *Петрович* | *4* | *г. Киров*  *ул. Свободы 4* | *3* |
| *3* | *Олег* | *Зотов* | *Георгиевич* | *5* | *г. Киров*  *ул. Воровского 122* | *3* |
| *4* | *Константин* | *Романов* | *Владимирович* | *6* | *г. Киров*  *ул. Ленина 52* | *4* |

-- Пример

*-- Получить только значения столбца address из таблицы customer***select address from** student;

|  |
| --- |
| **address** |
| *г. Киров ул. Ломоносова 16а* |
| *г. Киров ул. Свободы 4* |
| *г. Киров ул. Воровского 122* |
| *г. Киров ул. Ленина 52* |

-- Пример

*-- С помощью ключевого слова as можно изменять имя столбца в результирующей выборке***select id**, **address as** adrs **from** student;

|  |  |
| --- | --- |
| **id** | **adrs** |
| *1* | *г. Киров ул. Ломоносова 16а* |
| *2* | *г. Киров ул. Свободы 4* |
| *3* | *г. Киров ул. Воровского 122* |
| *4* | *г. Киров ул. Ленина 52* |

Если выполнить select из нескольких таблиц (без каких-либо фильтров), то результатом запроса будет декартово произведение множеств их строк.

-- Пример

*-- Выборка всех данных из таблицы subject***select** *\** **from** subject;

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **number\_of\_hours** |
| 1 | *Технологии программирования* | *18* |
| 2 | *История* | *36* |

*-- Выборка всех данных из таблицы progress***select** *\** **from** progress;

|  |  |  |
| --- | --- | --- |
| **id\_student** | **id\_subject** | **rating** |
| *1* | *1* | *4* |
| *2* | *1* | *5* |
| 4 | 2 | 5 |
| 2 | 2 | 4 |
| 3 | 2 | 3 |

*-- Выборка всех данных из обеих таблиц - декартово произведение множеств их строк*

*-- Если одна из таблиц пуста, то результатом запроса будет пустое множество***select** *\** **from** subject, progress;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | **name** | **number\_of\_hours** | **id\_student** | **id\_subject** | **rating** |
| 1 | *Технологии программирования* | *18* | *1* | *1* | *4* |
| 2 | *История* | *36* | *1* | *1* | *4* |
| 1 | *Технологии программирования* | *18* | *2* | *1* | *5* |
| 2 | *История* | *36* | *2* | *1* | *5* |
| 1 | Технологии программирования | 18 | 4 | 2 | 5 |
| 2 | История | 36 | 4 | 2 | 5 |
| 1 | Технологии программирования | 18 | 2 | 2 | 4 |
| 2 | История | 36 | 2 | 2 | 4 |
| 1 | Технологии программирования | 18 | 3 | 2 | 3 |
| 2 | История | 36 | 3 | 2 | 3 |

***Предложение where***

Чтобы отфильтровать результат запроса, используется предложение where. В нем задается любое выражение, имеющее тип boolean. Любая строка, не удовлетворяющая этому условию, исключается из результата. Строка удовлетворяет условию, если оно возвращает true при подстановке вместо ссылок на переменные фактических значений из этой строки.

Ниже показаны примеры использования предложения where.

-- Пример

*-- Этот запрос вернет все строки из subject***select** *\** **from** subject **where true**;

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **number\_of\_hours** |
| 1 | *Технологии программирования* | *18* |
| 2 | *История* | *36* |

-- Пример

*-- Этот запрос вернет 0 строк***select** *\** **from** subject **where false**;

-- Пример

*-- Этот запрос вернет только одну строку, поскольку только одна строка отвечает условию id = 2***select** *\** **from** subject **where id** = 2;

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **number\_of\_hours** |
| 2 | *История* | *36* |

-- Пример

*-- Этот запрос вернет строки, в которых находится информация о предмете и соответствующей оценке***select** *\****from** subject, progress  
**where** subject.**id** = progress.**id\_subject**;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | **name** | **count\_of\_hours** | **id\_student** | **id\_subject** | **rating** |
| *1* | *Технологии программирования* | *18* | *1* | 1 | 4 |
| *1* | *Технологии программирования* | *18* | *2* | 1 | 5 |
| *2* | *История* | *36* | *4* | 2 | 5 |
| *2* | *История* | *36* | *2* | 2 | 4 |
| 2 | История | 36 | 3 | 2 | 3 |

-- Данный запрос работает точно так же как и предыдущий

*-- В данном случае для столбца id в where не обязательно указывать имя таблицы subject, т.к. в таблице progress такого столбца нет*

**select** *\****from** subject, progress  
**where id** = progress.**id\_subject**;

-- Пример

*-- Чтобы быстрее писать запросы, удобнее использовать синонимы для таблиц  
-- Запись subject s в разделе from означает, что "s" следует интерпретировать как "subject". Аналогично с progress p***select** p.**id\_student**, p.**rating**, **s**.**name as** subject\_name  
**from** subject **s**, progress p  
**where s**.**id** = p.**id\_subject**;

|  |  |  |
| --- | --- | --- |
| **id\_student** | **rating** | **name** |
| *1* | 4 | *Технологии программирования* |
| *2* | 5 | *Технологии программирования* |
| *4* | 5 | *История* |
| *2* | 4 | *История* |
| 3 | 3 | История |

Кроме предложения where в запросе могут встречаться и другие, например, order by и limit.

***Предложение order by***

Предложение order by используется для сортировки строк в указанном порядке. В нем указываются столбцы (в порядке их приоритета при сортировке) и рядом с каждым из них – порядок сортировки (asc – по возрастанию, desc – по убыванию; если порядок не указан, то по умолчанию – asc). *Без указания предложения order by порядок вывода строк считается неопределенным.*

-- Пример

*-- Получить все строки из progress, отсортировать по rating по убыванию***select** *\** **from** progress **order by rating desc**;

|  |  |  |
| --- | --- | --- |
| **id\_student** | **id\_subject** | **rating** |
| *2* | 1 | 5 |
| *4* | 2 | 5 |
| *1* | 1 | 4 |
| *2* | 2 | 4 |
| 3 | 2 | 3 |

-- Пример

*-- Получить все строки из progress, отсортировать по rating по убыванию; если у двух строк одинаковые значения rating, то между собой они будут сортироваться по id\_subject по возрастанию.***select** *\** **from** progress **order by rating desc**, **id\_student asc**;

|  |  |  |
| --- | --- | --- |
| **id\_student** | **id\_subject** | **rating** |
| *2* | 1 | 5 |
| *4* | 2 | 5 |
| *1* | 1 | 4 |
| *2* | 2 | 4 |
| 3 | 2 | 3 |

***Предложение limit***

Предложение limit ограничивает максимальное количество строк, которое будет возвращено одним запросом, а так же имеет вложенное предложение offset, показывающее, сколько строк нужно пропустить, прежде чем начать выдавать строки.

-- Пример

*-- Получить не больше 2 строк из progress, сортировка по rating по возрастанию***select** *\** **from** progress **order by rating limit** 3;

|  |  |  |
| --- | --- | --- |
| **id\_student** | **id\_subject** | **rating** |
| *3* | 2 | 3 |
| *1* | 1 | 4 |
| *2* | 2 | 4 |

-- Пример

*-- Получить не более 2 строк из progress, сортировка по id\_student по возрастанию, пропустить одну строку перед выводом результата  
-- Важно, что в списке столбцов для select совсем не обязательно должны быть те столбцы, по которым ведутся фильтр, сортировка и т.д.***select id\_subject**, **rating from** progress **order by id\_student limit** 2 **offset** 1;

|  |  |
| --- | --- |
| **id\_subject** | **rating** |
| 1 | 5 |
| 2 | 4 |

Результаты нескольких select-запросов, если они содержат одинаковое количество столбцов с совместимыми типами, можно различным образом комбинировать, используя предложения union, intersect, except.

Выполнить union, except, intersect можно с любыми запросами при совместимости типов. Важно помнить, что без order by порядок вывода не определён.

Важно отметить, что во всех запросах, кроме самого первого, нельзя указывать предложения order by и limit. Для первого они указываются после все запросов, с которыми делается union, intersect или except.

***Предложение union***

Предложение union объединяет результаты нескольких запросов в одну выборку.

-- Пример

*-- Вернет две строки***select 'Первый'**, 1 **union select 'Второй'**, 2;

|  |  |
| --- | --- |
| **?column?** | **?column?** |
| Второй | 2 |
| Первый | 1 |

-- Пример

*-- Вернет одну строку, так как при union убираются дубликаты; если нужно сохранить дубликаты, нужно использовать union all***select 'Первый'**, 1 **union select 'Первый'**, 1;

|  |  |
| --- | --- |
| **?column?** | **?column?** |
| Первый | 1 |

***Предложение instersect***

Предложение instersect создает пересечение результатов запросов.

-- Пример

*-- Вернет пустое множество***select 'Первый'**, 1 **intersect select 'Второй'**, 2;

-- Пример

*-- Вернет одну строку***select 'Первый'**, 1 **intersect select 'Первый'**, 1;

|  |  |
| --- | --- |
| **?column?** | **?column?** |
| Первый | 1 |

***Предложение except***

Предложение except включает в результат только те строки, которые есть в выборке из первого запроса, но отсутствуют в выборке из второго.

-- Пример

*-- Вернет одну строку ("Первый", 1)***select 'Первый'**, 1 **except select 'Второй'**, 2;

|  |  |
| --- | --- |
| **?column?** | **?column?** |
| Первый | 1 |

-- Пример

*-- Вернет пустое множество***select 'Первый'**, 1 **except select 'Первый'**, 1;

***Предложение join***

Одна из операций реляционной алгебры – соединение таблиц. В результате этой операции из двух таблиц получается одна, каждая строка которой содержит столбцы обеих таблиц и некоторые значения, зависящие от способа и условий соединения. Для выполнения этой операции служит предложение join в команде select.

В этом разделе рассмотрен базовые примеры применения предложения join различных видов. *Подробнее про различные виды соединений можно прочитать в документации к команде select.*

***Внутреннее соединение (inner join).*** Соединяет строки двух различных таблиц, отвечающие заданному условию соединения. Результат получается такой же, как и при выполнении простого запроса select из двух таблиц с условием в where. В PostgreSQL это соединение существует исключительно для удобства записи. Порядок таблиц в условии соединения не важен. Пример показан ниже.

-- Пример 25

**select s**.**name**, **s**.**surname**, g.**name as** study\_group  
**from** student **s**, "group" g  
**where s**.**id\_group** = g.**id**;

*-- Результат такой же, как выше. Вместо inner join можно писать просто join***select s**.**name**, **s**.**surname**, g.**name as** study\_group  
**from** student **s inner join** "group" g **on s**.**id\_group** = g.**id**;

|  |  |  |
| --- | --- | --- |
| **name** | **surname** | **study\_group** |
| Василий | Олоев | ИВТб-3301-01-00 |
| Николай | Котов | ИВТб-3301-01-00 |
| Олег | Зотов | ИВТб-3302-02-00 |
| Константин | Романов | ИВТб-4301-01-00 |

***Перекрёстное соединение (cross join).*** Соединяет две таблицы, возвращая декартово произведение их строк. Не принимает условие для фильтра. Результат получается такой же, как и при выполнении запроса select из двух таблиц без условия в where, или при выполнении соединения inner join on true. В PostgreSQL это соединение существует исключительно для удобства записи. Пример показан ниже.

-- Пример 26

**select s**.**name**, **s**.**surname**, g.**name as** study\_group  
**from** student **s**, "group" g;

*-- Результат такой же, как выше.***select s**.**name**, **s**.**surname**, g.**name as** study\_group  
**from** student **s cross join** "group" g;

|  |  |  |
| --- | --- | --- |
| **name** | **surname** | **study\_group** |
| Николай | Котов | ИВТб-3301-01-00 |
| Василий | Олоев | ИВТб-3301-01-00 |
| Олег | Зотов | ИВТб-3301-01-00 |
| Константин | Романов | ИВТб-3301-01-00 |
| Николай | Котов | ИВТб-3302-02-00 |
| Василий | Олоев | ИВТб-3302-02-00 |
| Олег | Зотов | ИВТб-3302-02-00 |
| Константин | Романов | ИВТб-3302-02-00 |
| Николай | Котов | ИВТб-4301-01-00 |
| Василий | Олоев | ИВТб-4301-01-00 |
| Олег | Зотов | ИВТб-4301-01-00 |
| Константин | Романов | ИВТб-4301-01-00 |

***Левое внешнее соединение (left outer join).*** Работает так же, как внутреннее соединение, но добавляет в результат все строки из таблицы слева, для которых не нашлись строки таблицы справа, удовлетворяющие условию. Для таких строк в столбцах из правой таблицы будет стоят null. Порядок таблиц в условии важен. Пример показан ниже.

-- Пример

**select s**.**name**, **s**.**surname**, g.**name as** study\_group  
**from** student **s**, "group" g  
**where s**.**id\_group** = g.**id**;

|  |  |  |
| --- | --- | --- |
| **name** | **surname** | **study\_group** |
| Василий | Олоев | ИВТб-3301-01-00 |
| Николай | Котов | ИВТб-3301-01-00 |
| Олег | Зотов | ИВТб-3302-02-00 |
| Константин | Романов | ИВТб-4301-01-00 |

*-- Предположив, что в таблице student есть запись с id\_group = null, результат по сравнению с запросов выше будет включать и записи с id\_group = null.*

**select s**.**name**, **s**.**surname**, g.**name as** study\_group  
**from** student **s left join** "group" g **on s**.**id\_group** = g.**id**;

|  |  |  |
| --- | --- | --- |
| **name** | **surname** | **study\_group** |
| Василий | Олоев | ИВТб-3301-01-00 |
| Николай | Котов | ИВТб-3301-01-00 |
| Олег | Зотов | ИВТб-3302-02-00 |
| Константин | Романов | ИВТб-4301-01-00 |
| Елизавета | Никулина | null |

***Правое внешнее соединение (right outer join).*** Работает так же, как левое внешнее соединение, но вместо строк из таблицы слева, для которых не нашлось пары, добавляет строки из таблицы справа. Аналогично, для таких строк в столбцах из левой таблицы будет стоять null. Порядок таблиц в условии важен. Любой right outer join можно свести к left outer join, поменяв левую и правую таблицы в условии местами.

***Полное внешнее соединение (full outer join).*** Работает так же, как внутреннее соединение, но добавляет в результат все строки из таблицы слева, для которых не нашлось пары, и все строки из таблицы справа, для которых не нашлось пары. Отсутствующие значение заполняются null. Порядок таблиц в условии соединения не важен.

***Представления***

*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.

Как правило, при разработке приложений, использующих СУБД, делаются представления для многих таблиц, которые имеют внешние ключи.

Более того, в некоторых случаях устанавливается за правило – в клиентском приложении выполнять чтение только из представлений. Такое правило не очень обоснованно, т. к. представления иногда создают препятствия для оптимизатора и чтение из них может работать гораздо дольше, чем напрямую из таблиц. Но такие случаи редки и относятся к большим сложным запросам. Если опыт показывает, что использование представления не замедляет работу приложения, лучше всегда использовать представление, чем многократно писать запрос.

Так же, как и любые объекты БД, представления лучше размещать в схемах. В имени представления желательно использовать суффикс «\_v».

*Подробнее о представлениях в PostgreSQL можно прочитать в документации.*

Ниже показаны примеры работы с представлениями.

-- Пример

*-- Для создания представления используется ключевое слово view.*

*Create or replace означает, что если представление с данным именем уже существует, то оно будет заменено.*

*-- В данном примере создается представление для таблицы студентов, содержащее их ФИО, курс обучения и название учебной группы.***create or replace view** students\_groups\_v **as  
select s**.**name**, **s**.**middle\_name**, **s**.**surname**, **s**.**cource**, g.**name as** group\_name  
**from** student **s**, "group" g  
**where s**.**id\_group** = g.**id**;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **name** | **middle\_name** | **surname** | **course** | **group** |
| Василий | Петрович | Олоев | 3 | ИВТб-3301-01-00 |
| Николай | Сергеевич | Котов | 3 | ИВТб-3301-01-00 |
| Олег | Геогриевич | Зотов | 3 | ИВТб-3302-02-00 |
| Константин | Владимирович | Романов | 4 | ИВТб-4301-01-00 |

-- Пример

*-- В данном примере создается представление, отражающее успеваемость студентов. Для соединения строк таблиц student, subject, progress используется внутреннее соединение join.*  
**create or replace view** students\_progress\_subject\_v **as  
select s**.**name**, **s**.**middle\_name**, **s**.**surname**, **s**.**cource**, su.**name as** subject, p.**rating  
from** student **s join** progress p **on s**.**id** = p.**id\_student**

**join** subject su **on** p.**id\_subject** = su.**id**;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **name** | **middle\_name** | **surname** | **course** | **subject** | **rating** |
| Николай | Сергеевич | Котов | 3 | Технологии программирования | 4 |
| Василий | Петрович | Олоев | 3 | Технологии программирования | 5 |
| Константин | Владимирович | Романов | 4 | История | 5 |
| Василий | Петрович | Олоев | 3 | История | 4 |
| Олег | Геогриевич | Зотов | 3 | История | 3 |

-- Пример

*-- В данном примере создается представление, отражающее занятость преподователей.*  
**create or replace view** teachers\_subjects\_v **as  
select** t.**name**, t.**middle\_name**, t.**surname**, t.**position**, **s**.**name as** subject, too.**occupation\_type  
from** teacher t, subject **s**, type\_of\_occupation too  
**where** t.**id** = too.**id\_teacher and s**.**id** = too.**id\_subject**;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **name** | **middle\_name** | **surname** | **position** | **subject** | **occupation\_type** |
| Семен | Петрович | Катаев | instructor | Технологии программирования | laboratory work |
| Ирина | Олеговна | Коснырева | docent | Технологии программирования | lecture |
| Дмитрий | Алексеевич | Мусанов | professor | История | lecture |

1. 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 |

***Агрегатные функции***

Часто при выполнении запросов нужно получить результат выполнения какой-либо функции над множеством значений (сумма, среднее значение, подсчет количества, объединение в массив или строку (text) и т.д.). Для этого используются агрегатные функции.

В PostgreSQL существует большое количество различных агрегатных функций. *Подробнее про них написано в документации.*

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.

В таблице 1 перечислены некоторые агрегатные функции общего назначения.

Таблица 1 – Некоторый агрегатные функции общего назначения

|  |  |
| --- | --- |
| Функция | Описание |
| avg(*выражение*) | арифметическое среднее для всех входных значений |
| max(*выражение*) | максимальное значение *выражения* среди всех входных данных |
| min(*выражение*) | минимальное значение *выражения* среди всех входных данных |
| string\_agg(*выражение*, *разделитель*) | входные данные складываются в строку через заданный разделитель |
| sum(*выражение*) | сумма значений *выражения* по всем входным данным |
| count(\*) | количество входных строк |
| count(*выражение*) | количество входных строк, для которых значение *выражения* не равно NULL |

-- Пример

*-- Два запроса ниже эквиваленты и возвращают количество записей в student***select** *count*(*\**) **from** student;  
**select** *count*(1) **from** student;

|  |
| --- |
| **count** |
| 5 |

-- Пример

*-- Возвращает строку с четырьмя значениями: сумма по столбцу number\_of\_hours, минимальное, максимальное и среднее значения в number\_of\_hours***select** *sum*(**number\_of\_hours**), *min*(**number\_of\_hours**), *max*(**number\_of\_hours**), *avg*(**number\_of\_hours**) **from** subject;

|  |  |  |  |
| --- | --- | --- | --- |
| **sum** | **min** | **max** | **avg** |
| 54 | 18 | 36 | 27 |

-- Пример

*-- Этот запрос выбирает всех стеднтов и сданные ими предметы, отображает:  
-- subjects - список предметов (имена, разделенные запятой)  
-- average\_rating - средний балл  
-- Записи отсортированы по возрастанию по стреднему баллу  
-- Важно отметить, что все столбцы st.id, st.course нужно указать в group by, чтобы выполнить группировку, поскольку они указаны в select***select** st.**id**, st.**cource**,  
*string\_agg*(sb.**name**, **', '**) **as** subjects, *avg*(p.**rating**) **as** average\_rating  
**from** student st, subject sb, progress p  
**where** p.**id\_subject** = sb.**id and** p.**id\_student** = st.**id  
group by** st.**id**, st.**cource order by** average\_rating **asc**;

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **course** | **subjects** | **average\_rating** |
| 3 | 3 | История | 3 |
| 1 | 3 | Технологии программирования | 4 |
| 2 | 3 | Технологии программирования, История | 4,5 |
| 4 | 4 | История | 5 |

***Предложение having***

Иногда возникает потребность сделать выборку, в которой строки отвечают определённому условию, вычисляемому с помощью агрегатных функций. Такие условия следует записывать не в предложение where, а в предложение having. Оно отличается от where: where фильтрует отдельные строки до применения group by, а having фильтрует строки групп, созданных предложением group by.

-- Пример 34

*-- Этот запрос вернет записи, у которых средний балл больше 4***select** st.**id**, st.**cource**,  
*string\_agg*(sb.**name**, **', '**) **as** subjects, *avg*(p.**rating**) **as** average\_rating  
**from** student st, subject sb, progress p  
**where** p.**id\_subject** = sb.**id and** p.**id\_student** = st.**id  
group by** st.**id**, st.**cource having** *avg*(p.**rating**) > 4  
**order by** average\_rating **asc**;

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **course** | **subjects** | **average\_rating** |
| 2 | 3 | Технологии программирования, История | 4,5 |
| 4 | 4 | История | 5 |

Ниже представлен скрипт, который получился в ходе выполнения данной лабораторной работы.

*--Заполнение таблиц***do $$  
declare** groupID **bigint**;  
 subjectID **bigint**;  
 teacherID **bigint**;  
 student1 **bigint**;  
 student2 **bigint**;  
 student3 **bigint**;  
 student4 **bigint**;  
**begin  
  
insert into** "group"(**name**, **number\_of\_students**)  
**values** (**'ИВТб-3301-01-00'**, 15)  
**returning id into** groupID;  
**insert into** student(**name**, **surname**, **middle\_name**, **id\_group**, **address**, **cource**)  
**values** (**'Николай'**, **'Котов'**, **'Сергеевич'**, groupID, **'г. Киров ул. Ломоносова 16а'**, 3)  
**returning id into** student1;  
**insert into** student(**name**, **surname**, **middle\_name**, **id\_group**, **address**, **cource**)  
**values** (**'Василий'**, **'Олоев'**, **'Петрович'**, groupID, **'г. Киров ул. Свободы 4'**, 3)  
**returning id into** student2;  
  
**insert into** "group"(**name**, **number\_of\_students**)  
**values** (**'ИВТб-3302-02-00'**, 20)  
**returning id into** groupID;  
**insert into** student(**name**, **surname**, **middle\_name**, **id\_group**, **address**, **cource**)  
**values** (**'Олег'**, **'Зотов'**, **'Георгиевич'**, groupID, **'г. Киров ул. Воровского 122'**, 3)  
**returning id into** student3;  
  
**insert into** "group"(**name**, **number\_of\_students**)  
**values** (**'ИВТб-4301-01-00'**, 12)  
**returning id into** groupID;  
**insert into** student(**name**, **surname**, **middle\_name**, **id\_group**, **address**, **cource**)  
**values** (**'Константин'**, **'Романов'**, **'Владимирович'**, groupID, **'г. Киров ул. Ленина 52'**, 4)  
**returning id into** student4;  
  
**insert into** subject(**name**, **number\_of\_hours**)  
**values** (**'Технологии программирования'**, 18)  
**returning id into** subjectID;  
**insert into** teacher(**name**, **surname**, **middle\_name**, **position**, **phone\_number**)  
**values** (**'Семен'**, **'Катаев'**, **'Петрович'**, **'instructor'**, **'89530000000'**)  
**returning id into** teacherID;  
**insert into** type\_of\_occupation(**id\_subject**, **id\_teacher**, **occupation\_type**)  
**values** (subjectID, teacherID, **'laboratory work'**);  
**insert into** progress(**id\_subject**, **id\_student**, **rating**)  
**values** (subjectID, student1, 4);  
**insert into** progress(**id\_subject**, **id\_student**, **rating**)  
**values** (subjectID, student2, 5);  
  
**insert into** teacher(**name**, **surname**, **middle\_name**, **position**, **phone\_number**)  
**values** (**'Ирина'**, **'Коснырева'**, **'Олеговна'**, **'docent'**, **'89530000002'**)  
**returning id into** teacherID;  
**insert into** type\_of\_occupation(**id\_subject**, **id\_teacher**, **occupation\_type**)  
**values** (subjectID, teacherID, **'lecture'**);  
  
**insert into** subject(**name**, **number\_of\_hours**)  
**values** (**'История'**, 36)  
**returning id into** subjectID;  
**insert into** teacher(**name**, **surname**, **middle\_name**, **position**, **phone\_number**)  
**values** (**'Дмитрий'**, **'Мусанов'**, **'Алексеевич'**, **'professor'**, **'89530000001'**)  
**returning id into** teacherID;  
**insert into** type\_of\_occupation(**id\_subject**, **id\_teacher**, **occupation\_type**)  
**values** (subjectID, teacherID, **'lecture'**);  
**insert into** progress(**id\_subject**, **id\_student**, **rating**)  
**values** (subjectID, student4, 5);  
**insert into** progress(**id\_subject**, **id\_student**, **rating**)  
**values** (subjectID, student2, 4);  
**insert into** progress(**id\_subject**, **id\_student**, **rating**)  
**values** (subjectID, student3, 3);  
**end**;  
**$$ language** plpgsql;  
  
*--Создание представлений***create or replace view** students\_groups\_v **as  
 select s**.**name**, **s**.**middle\_name**, **s**.**surname**, **s**.**cource**, g.**name as** group\_name  
 **from** student **s**, "group" g  
 **where s**.**id\_group** = g.**id**;  
  
**create or replace view** students\_progress\_subject\_v **as  
 select s**.**name**, **s**.**middle\_name**, **s**.**surname**, **s**.**cource**, su.**name as** subject, p.**rating  
 from** student **s join** progress p **on s**.**id** = p.**id\_student join** subject su **on** p.**id\_subject** = su.**id**;  
  
**create or replace view** teachers\_subjects\_v **as  
 select** t.**name**, t.**middle\_name**, t.**surname**, t.**position**, **s**.**name as** subject, too.**occupation\_type  
 from** teacher t, subject **s**, type\_of\_occupation too  
 **where** t.**id** = too.**id\_teacher and s**.**id** = too.**id\_subject**;  
  
*--Создание представления с числовыми данными***create or replace view** subjects\_v **as** (**select 'Минимальное значение'**, **number\_of\_hours**, **id from** subject **order by number\_of\_hours asc limit** 1)  
**union all** (**select 'Максимальное значение'**, **number\_of\_hours**, **id from** subject **order by number\_of\_hours desc limit** 1)  
**union all  
select 'Среднее значение'**, *avg*(**s**.**number\_of\_hours**), **null from** subject **s  
union all  
select 'Сумма значений'**, *sum*(**s**.**number\_of\_hours**), **null from** subject **s**;

Далее представлено несколько дополнительных команд, которые могут быть использованы при выполнении данной лабораторной работы.

***Команда update***

Команда update используется для изменения существующих строк таблицы. Она позволяет поменять значения в столбцах некоторых строк, удовлетворяющих определённому условию. Столбцы, не включенные в команду, сохранят свои предыдущие значения. Для выбора тех строк, в которых будет произведено изменение, используется предложение where, как в команде select.

*Подробнее про синтаксис команды update можно прочитать в документации.*

-- Пример

*-- Обновляет в таблице student строку c id = 1, устанавливая столбцу id\_group значение 2.*

*-- Остальные столбцы не изменят своего значения.***update** student **set id\_group** = 2 **where id** = 1;

-- Пример

*-- В одной команде можно обновить несколько полей.*

*-- Обновляет в таблице student строку c id = 1, устанавливая столбцу id\_group значение 2, столбцу address = 'г. Киров ул. Преображенская 34'.*

*-- Остальные столбцы не изменят своего значения.***update** student **set id\_group** = 2, **address** = **'г. Киров ул. Преображенская 34' where id** = 1;

-- Пример 37

*-- Обновляет все строки в subject, увеличивая значение столбца number\_of\_hours на 10.***update** subject **set number\_of\_hours** = **number\_of\_hours** + 10;

-- Пример

*-- Команда ниже не выполнится, возникнет ошибка, поскольку у таблицы group нет записи с id = 8***update** student **set id\_group** = 8 **where id** = 1;

***Команда delete***

Для удаления строк из таблицы используется команда delete. Она позволяет удалить некоторые строки, удовлетворяющие определенному условию. Для выбора тех строк, которые будут удалены, используется предложение where, как в команде select. *Подробнее про синтаксис команды delete можно прочитать в документации.*

-- Пример

*-- Удалит из таблицы progress строки, в которых rating != 0*

**delete from** progress **where rating** != 0;

-- Пример 40

*-- Команда не выполнится, возникнет ошибка, т.к. на subject с id = 1 есть внешние ссылки (определено ограничением foreign key в progress и type\_of\_occupation)***delete from** subject **where id** = 1;

-- Пример

*-- Команда должна удалить все записи из таблицы subject, но не выполнится по той же причине, что и в предыдущем примере***delete from** subject;

***Операторы и встроенные функции, работа с датами***

Во многих примерах ранее использовались арифметические и логические операторы, а также различные встроенные функции (например, now(), возвращающая дату начала текущей транзакции).

PostgreSQL предоставляет большое количество операторов и встроенных функций. *Подробнее можно прочитать в документации.*

Поскольку их количество очень велико, а написание совпадает во многих случаях с общепринятым в языках программирования, здесь не будет приводится списков операторов и встроенных функций. Вместо этого ниже показаны примеры различных часто встречаемых или неочевидных ситуаций, которые могут связанных с их использованием.

-- Пример

*-- Есть три логических оператора: and, or, not. Приоритет стандартный***select not** 1 < 2 **and** 3 < 4 **or** 5 = 5; *-- true*

-- Пример

*-- Оператор "не равно" можно записывать как в виде <>, так и !=***select** 1 <> 1 **or** 5 != 5; *-- false*

-- Пример

*-- Нельзя сравнивать значение с null с помощью = или <>. Нужно использовать предикаты is null, is not null.***select** 1 **is null or** 5 **is not null**; *-- true*

-- Пример 45

*-- Для проверки вхождения в промежуток есть предикаты between, now between***select** 3 **between** 3 **and** 5; *-- true***select** 1 **not between** -1 **and** 5; *-- false*

-- Пример 46

*-- Для конкатенации строк используется оператор || или функция concat***select 'Hello, '**||**'Postgre!'**; *-- Hello, Postgre!***select** *concat*(**'Hello, '**, **'Postgre!'**); *-- Hello, Postgre!*

-- Пример 47

*-- Разница между ними в том, что строка||null = null, а функция concat воспринимает null, как пустую строку***select 'abc'**||**null**; *-- null***select** *concat*(**'abc'**, **null**, **'de'**); *-- abcde*

-- Пример

*-- Чтобы вставить перенос строки, можно сделать так. Функция chr возвращает символ по его коду в кодировке БД***select ' first'**||*chr*(13)|| **'second'**; *-- first \n second*

-- Пример

*-- Но если нужно поставить одинарную кавычку в строке, можно поставить одинарных кавычки подряд***select 'Don**''**t worry'**; *-- Don't worry*

-- Пример

*-- Для приведения в верхний или нижний регистр используются функции lower, upper***select** *lower*(**'AbC'**); *-- abc***select** *upper*(**'AbC'**); *-- ABC*

-- Пример

*-- Функция trim обрезает пробельные символы по бокам строки***select** *trim*(**' a b c '**); *-- "a b c"*

-- Пример

*-- Конструкцию case можно использовать по аналогии со switch или для выборки разных значений по условию***select case when** 1 = 1 **then 'AAA' else 'OOO' end**; *-- AAA*

-- Пример

*-- Полезна функция coalesce(a, b, c, ...), которая вернет первый аргумент, отличный от null***select** *coalesce*(**null**, **null**, 42, **null**, 33); *-- 42*

-- Пример

*-- Строки возможно проверять по шаблону с помощью оператора like. В шаблоне % означает любое количество символов, \_ - один символ***select 'abcdef' like '%cdef'**; *-- true***select 'abcdef' like 'a\_cdef'**; *-- true***select 'abcdef' like '%c%'**; *-- true***select 'abcdef' like 'ab\_'**; *-- false*

-- Пример

*-- Часто преобразование типов происходит неявно***select** 42||**'...'**; *-- "42..."*

-- Пример

*-- Но можно делать его вручную с помощью оператора ::***select** 42::*text*; *-- "42"*

-- Пример

*-- Возвращает timestamp на начало текущей транзакции***select** *now*();

В двух последних примерах было рассмотрено преобразование типов. Часто оно используется при написании дат в запросах (преобразовать строку в дату). В этом случае принимается практически любой стандартный формат даты. Тем не менее, он во многом определяется местоположением клиента, и, чтобы избежать неожиданных эффектов, лучше использовать функции to\_date, to\_timestamp, принимающих вторым аргументом формат даты. Примеры показаны ниже.

-- Пример

*-- Два запроса ниже аналогичны, если формат даты на клиенте совпадает с указанным***select '12-05-2019'**::*date*;  
**select** *to\_date*(**'12-05-2019'**, **'DD-MM-YYYY'**);

-- Пример

*-- Два запроса ниже аналогичны, если формат даты на клиенте совпадает с указанным***select '12-05-2019 12:08'**::**timestamp**;  
**select** *to\_timestamp*(**'12-05-2019 12:08'**, **'DD-MM-YYYY HH:MI'**);

Если суммировать date и число, к дате прибавится указанное количество дней. Суммировать timestamp и число нельзя. Для более очевидного прибавления временных промежутков лучше суммирование не с числами, а с временными интервалами. Важно помнить, что при сложении date и временного интервала получается timestamp. Примеры показаны ниже.

-- Пример

*-- Прибавляет один день к дате***select** *to\_date*(**'12-05-2019'**, **'DD-MM-YYYY'**) + 1; *-- 13-05-2019*

-- Пример

*-- Складывать timestamp и число нельзя***select** *to\_timestamp*(**'12-05-2019 12:08'**, **'DD-MM-YYYY HH:MI'**) + 1; *-- Ошибка*

-- Пример 62

*-- К датам лучше прибавлять временные интервалы***select** *to\_date*(**'12-05-2019'**, **'DD-MM-YYYY'**) + **interval '1 day'**; *-- 13-05-2019 00:00***select** *to\_date*(**'12-05-2019'**, **'DD-MM-YYYY'**) - **interval '5 year'**; *-- 12-05-2014 00:00***select** *to\_timestamp*(**'12-05-2019 12:08'**, **'DD-MM-YYYY HH:MI'**) + **interval '1 year 2 month 11 second'**; *-- 12-07-2020 00:08:11*

-- Пример

*-- Даты также можно вычитать, результат - количество дней***select** *to\_date*(**'12-05-2019'**, **'DD-MM-YYYY'**) - *to\_date*(**'10-05-2019'**, **'DD-MM-YYYY'**); *-- 2*

-- Пример

*-- Если вычесть date из timestamp, timestamp из date или timestamp из timestamp, получится временной интервал***select** *to\_timestamp*(**'12-05-2019 12:08'**, **'DD-MM-YYYY HH:MI'**) -  
 *to\_date*(**'10-05-2019'**, **'DD-MM-YYYY'**);  
*-- 0 years 0 mons 2 days 0 hours 8 mins 0.00 secs*

***Подзапросы и предложение with***

Если запрос что-либо возвращает, к результатам его работы можно обратиться в другом запросе. Например, в разделе from помимо таблиц можно указывать подзапросы (при этом обязательно давать им имена).

Примеры использования подзапросов показаны ниже.

-- Пример 65

*-- Бесполезный пример***select** *\** **from** (**select** *\** **from** student) s;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **id** | **name** | **surname** | **middle\_name** | **id\_group** | **address** | **cource** |
| *1* | *Николай* | *Котов* | *Сергеевич* | *4* | *г. Киров*  *ул. Ломоносова 16а* | *3* |
| *2* | *Василий* | *Олоев* | *Петрович* | *4* | *г. Киров*  *ул. Свободы 4* | *3* |
| *3* | *Олег* | *Зотов* | *Георгиевич* | *5* | *г. Киров*  *ул. Воровского 122* | *3* |
| *4* | *Константин* | *Романов* | *Владимирович* | *6* | *г. Киров*  *ул. Ленина 52* | *4* |

-- Пример

*-- Запрос может обращаться к полям вложенного запроса не глубже, чем на один уровень  
-- Вложенные запросы одного уровня не могут обращаться друг к другу***select** s.*\** **from** ( *-- select s.\* здесь сделать нельзя***select** ss.*\** **from** (**select** *\** **from** student) ss *-- Дописать на этом же уровне еще один вложенный запрос, который обращается к ss, нельзя*) s;

-- Пример

*-- Если возвращается одна строка, то подзапросы можно использовать так (с оператором =)*

**delete from** student **where id** = (**select id\_student from** progress **where rating** = **'5'**);

-- Пример 68

*-- Если значений несколько, можно использовать оператор in***select name**, **surname from** student **where id in** (**select id\_student from** progress **where rating** = **'4'**);

|  |  |
| --- | --- |
| **name** | **surname** |
| *Николай* | *Котов* |
| *Василий* | *Олоев* |

Использование подзапросов очень полезно и многие задачи нельзя решить без их применения. Тем не менее, с этим связаны некоторые проблем:

* область видимости результатов подзапроса ограничена;
* если для выполнения всего запроса необходимо СУБД обратиться много раз к результатам подзапроса, он может выполняться каждый раз.

Указанные выше проблемы решаются с помощью предложения with. Оно может размещаться перед командам select, insert, update и delete. В нем можно расположить несколько именованных подзапросов и далее использовать их результаты. Такие подзапросы называют *общими табличными выражениями* (Common Table Expressions, CTE).

Каждое CTE может обращаться к результатам работы вышестоящих CTE в предложении with. Создаваемая в этом случае выборка вычисляется только один раз и сохраняется во временную таблицу, существующую на время выполнения всего запроса. Кроме того, запросы с предложениями with гораздо проще читать и понимать.

*Подробнее о предложении with и CTE можно прочитать в документации.*

Ниже показаны примеры использования предложения with.

-- Пример 69

*--Создание таблицы good\_students***create table public**.good\_students(  
 **id bigserial primary key**,  
 **name varchar**(30) **not null**,  
 **surname varchar**(30) **not null**,  
 **middle\_name varchar**(30),  
 **id\_group bigint not null references public**.group(**id**),  
 **address varchar**(100),  
 **cource int check** (**cource** >= 1 **and cource** <=5)  
);

*-- Этот запрос перенесет записи из таблицы student в таблицу good\_students данные студентов с оценками > 4***with** items\_to\_insert **as** (**select id\_student from** progress **where rating** < 4)  
**insert into** good\_students (**select id**, **name**, **surname**, **middle\_name**, **id\_group**, **address**, **cource from** student **where id in** (**select** *\** **from** items\_to\_insert));

***Предложение returning***

Полезным часто бывает то, что команды insert, update и delete могут возвращать строки. Для этого используется предложение returning, которое ставится в конце запроса и возвращает строки, которые были затронуты выполнением команды. Для update возвращаются уже измененные значения строк.

Результат, получаемый с помощью returning, нельзя использовать как подзапрос, но можно использовать в предложении with. С помощью этого, например, просто сохранять ссылочную целостность при заполнении таблиц данными: в предложении with можно вставлять строки, на которые будут ссылаться вставляемые в основной части запроса.

Ниже показаны примеры использования предложения returning.

-- Пример 70

*-- Этот запрос обновит таблицу subject и вернет все обновленные строки***update** subject **set number\_of\_hours** = **number\_of\_hours** - 2 **where number\_of\_hours** >= 30 **returning** *\**;

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **count\_of\_hours** |
| *2* | *История* | *34* |