**Лабораторная работа № 3  
1. Создание простых запросов на выборку**

**Теоретические сведения**

Рассмотрим следующие вопросы:

* выборка данных из одной таблицы с помощью оператора *SELECT*;
* использование в запросах операторов и встроенных функций PostgreSQL.

Один из основных инструментов для выполнения запросов к базам данных - оператор SELECT. Он служит для извлечения нужных данных из таблиц и является основным средством взаимодействия с данными в базах данных. Уникальной особенностью SELECT является его способность представлять результат запроса в виде таблицы. Это обеспечивает структурированный подход к данным, где каждая строка и столбец имеют свою определенную роль.

Полученные результаты запроса могут быть использованы для создания новых таблиц, что особенно полезно, когда требуется анализировать данные из различных источников. Создание новых таблиц на основе результатов запроса облегчает последующие операции анализа и обработки данных.

Кроме того, таблица, сформированная в результате запроса, может служить основой для дальнейших запросов. Это позволяет эффективно использовать результаты предыдущих запросов для получения дополнительной информации из баз данных.

Таким образом, оператор SELECT является ключевым инструментом для работы с базами данных, позволяя не только извлекать информацию, но и создавать новые таблицы и выполнять последующие запросы на основе полученных данных. Это делает его неотъемлемой частью инструментария профессиональных аналитиков и специалистов в области баз данных.Общая форма оператора *SELECT*:

Общая форма оператора *SELECT*:

*SELECT столбцы FROM таблицы*

*[WHERE условия]*

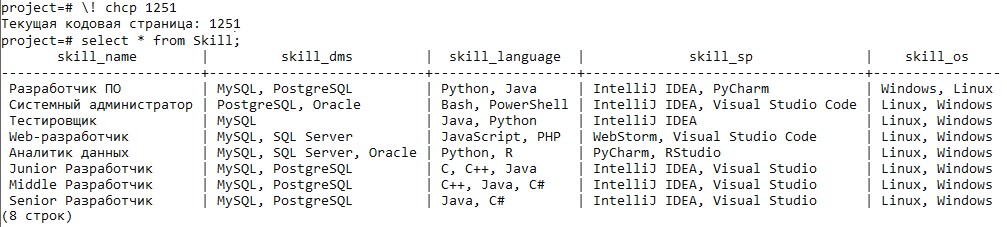
*[GROUP BY группа [HAVING групповые\_условия] ]*

*[ORDER BY имя\_поля]*

*[LIMIT пределы];*

Оператор *SELECT* имеет много опций. Их можно использовать или не исполь­зовать, но они должны указываться в том порядке, в каком они приведены. Если требуется вывести все столбцы таблицы, необязательно перечислять их после ключевого слова *select*, достаточно заменить этот список символом \*.

select \* from Skill;



Список столбцов в операторе *select* используют, если нужно изменить порядок следования столбцов в результирующей таблице или выбрать часть столбцов.

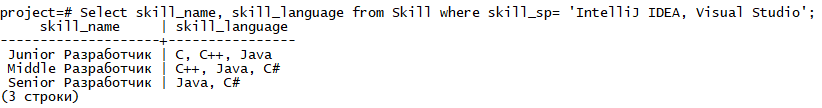
В PostgreSQL, помимо использования оператора SELECT для получения таблиц, существуют и другие способы получения данных в виде таблиц:

* Временные таблицы (Temporary Tables): В PostgreSQL можно создавать временные таблицы, которые существуют только в рамках текущей сессии или транзакции. Они могут быть созданы и заполнены результатами запроса или другими данными, и после завершения сессии или транзакции автоматически удаляются.
* Представления (Views): Представления представляют собой виртуальные таблицы, которые основаны на результатах запросов к одной или нескольким реальным таблицам. Они сохраняются в базе данных и могут использоваться для упрощения сложных запросов или сокрытия деталей структуры данных.
* Функции (Functions): В PostgreSQL можно создавать функции, которые возвращают таблицы в качестве результата. Это могут быть хранимые процедуры, которые выполняют какие-либо операции над данными и возвращают результат в виде таблицы.
* Расширения (Extensions): Некоторые расширения PostgreSQL предоставляют специальные типы данных или функции для работы с данными. Некоторые из них могут возвращать данные в виде таблиц.
* Foreign Data Wrappers (FDW): PostgreSQL поддерживает FDW, которые позволяют создавать таблицы, представляющие данные из внешних источников данных, таких как другие базы данных, файлы или веб-службы.

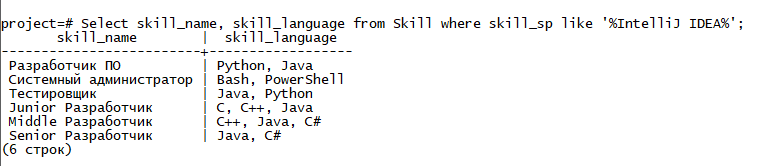
Эти способы позволяют получать данные в виде таблиц и дополняют функциональность оператора SELECT, обеспечивая более широкие возможности работы с данными в PostgreSQL.

**Условия выборки**. Гораздо чаще встречается ситуация, когда необходимо изменить количество выводимых строк. Для выбора записей, удовлетворяющих определен­ным критериям поиска, можно использовать конструкцию *WHERE*.

Select skill\_name, skill\_language from Skill where skill\_sp= 'IntelliJ IDEA, Visual Studio';

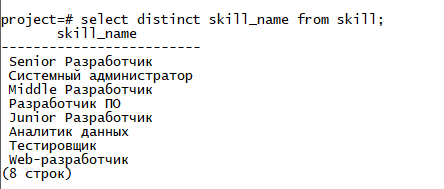


В запросе можно использовать слово LIKE если мы хотим найти не полную строчку, а часть содержимого из нее:  
Select skill\_name, skill\_language from Skill where skill\_sp like '%IntelliJ IDEA%';



В запросе можно использовать ключевое слово *DISTINCT*, для выбора уникальных значений из столбца или комбинации столбцов. Когда вы используете DISTINCT, PostgreSQL возвращает только уникальные строки из результирующего набора, удаляя любые дубликаты, то есть результат не будет содержать в себе повторений уже существующих значений, например:

select distinct skill\_name from skill;

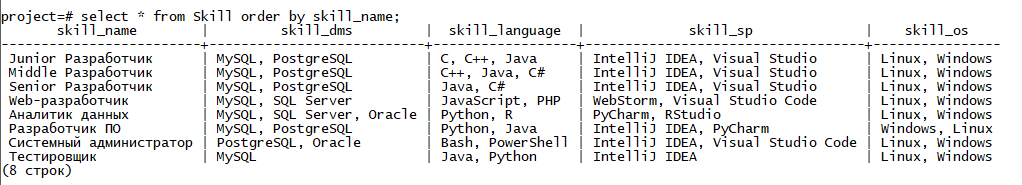


**Сортировка**.

Сортировка записей в PostgreSQL играет важную роль в организации данных для удобства анализа и работы с ними. При выполнении запроса на выборку данных, результаты по умолчанию возвращаются в порядке, в котором они хранятся в базе данных. Однако, для упорядочивания результатов по определенному столбцу, используется конструкция ORDER BY, за которой следует имя столбца, по которому необходимо провести сортировку, например:

select \* from Skill order by skill\_name;

select \* from Skill order by skill\_name;

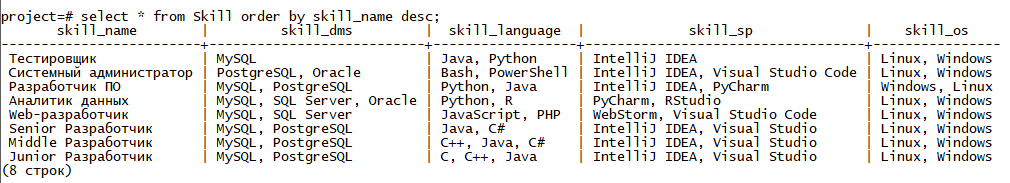


Помимо этого, сортировка может быть осуществлена по нескольким столбцам, перечисленным через запятую после ключевого слова ORDER BY. Нет ограничений на количество столбцов, указываемых для сортировки. Например, если вы хотите отсортировать данные сначала по столбцу A, а затем по столбцу B, вы можете сделать это следующим образом:

*SELECT [column1,…, column N] FROM [table\_name] ORDER BY[column1,…, column N];*

По умолчанию сортировка производится в прямом порядке (записи располагаются от наименьшего значения поля сортировки до наибольшего). Обратный порядок сортировки реализуется с помощью ключевого слова DESC. Например, если вы хотите отсортировать данные по столбцу A в обратном порядке:

select \* from Skill order by skill\_name desc;



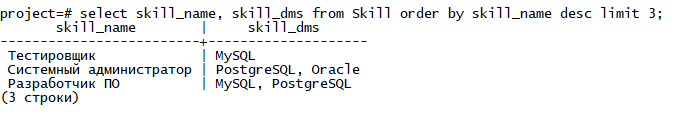
Вы можете применить обратный порядок сортировки к каждому столбцу по отдельности, указав ключевое слово DESC после каждого столбца, по которому вы хотите осуществить сортировку в обратном порядке:

*SELECT [column1,…, column N] FROM [table\_name] ORDER BY [column1] DESC, …., [column N] DESC;*

Для прямой сортировки существует ключевое слово *asc*. Оно используется для явного указания прямой сортировки (по возрастанию), хотя оно не обязательно, так как прямая сортировка является поведением по умолчанию.

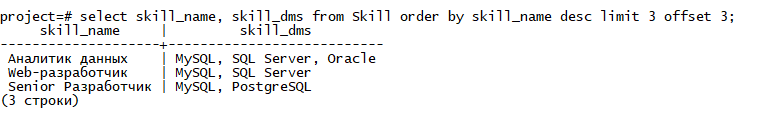
**Ограничение выборки**. Результат выборки может содержать тысячи записей, вывод и обработка которых занимают значительное время. Поэтому информацию часто разбивают на страницы и предоставляют ее пользователю частями. Постраничная навигация используется при помощи ключевого слова *limit*, за которым следует число выводимых записей. Следующий запрос извлекает первые 3 записи, при этом осуществляется обратная сортировка по полю *skill\_name*:

select skill\_name, skill\_dms from Skill order by skill\_name desc limit 3;



Для извлечения следующих трех записей используется ключевое слово OFFSET и *limit* с двумя цифрами. Первая указывает позицию, начиная с которой необходимо вернуть результат, вторая цифра – число извлекаемых записей, например:

select skill\_name, skill\_dms from Skill order by skill\_name desc limit 3 offset 3;



При определении смещения нумерация строк начинается с нуля.

**Группировка записей**. Конструкция *GROUP ВУ* позволяет группировать извлекаемые строки. Она полезна в комбинации с функциями, применяемыми к группам строк. Эти функции (табл. 1) называются агрегатами (суммирующими функциями) и вычисляют одно значение для каждой группы, создаваемой конструкцией *group by*. Функции позволяют узнать число строк в группе, подсчитать среднее значение, получить сумму значений столбцов. Результирующее значение рассчитывается для значений, не равных *null* (исключение – функция *count(\*)*). Допустимо использование этих функций в запросах без группировки (вся выборка – одна группа).

Пример использования функции *count( )*, которая возвращает число строк в таблице, значения указанного столбца для которых отличны от *NULL*:

Select count (skill\_name) from Skill;

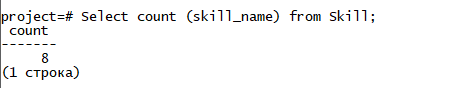
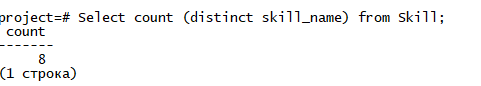


Таблица 1

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| *AVG ( [DISTINCT]  expr)* | Возвращает среднее значение аргумента *expr*. В качестве аргумента обычно выступает имя столбца. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |
| *COUNT ( )* | Подсчитывает число записей и имеет несколько форм. Форма *COUNT (выражение)* возвращает число записей в таблице, поле *выражение* для которых не равно *null*. Форма *count(\*)* возвращает общее число строк в таблице независимо от того, принимает какое-либо поле значение *null* или нет. Форма *COUNT (DISTINCT выражение1, выражение2, ... )* позволяет использовать ключевое слово *distinct*, которое позволяет подсчитать только уникальные значения столбца |
| *MIN ( [DISTINCT]  expr)* | Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце *expr*. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |
| *MAX ( [DISTINCT]  expr)* | Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце *expr*. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |
| *STD (expr)* | Возвращает стандартное среднеквадратичное отклонение в аргументе *expr* |
| *STDDEV\_SAMP (expr)* | Возвращает выборочное среднеквадратичное отклонение в аргументе *expr* |
| *SUM ( [DISTINCT]  expr)* | Возвращает сумму величин в столбце *expr*. Необязательное слово *distinct* позволяет обрабатывать только уникальные значения столбца *expr* |

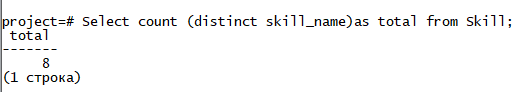
Использование ключевого слова *distinct* с функцией *count( )* позволяет вернуть число уникальных значений *cli\_ID* в таблице *Client*, например:

Select count (distinct skill\_name) from Skill;

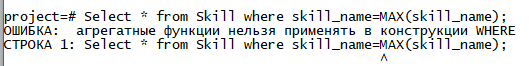


В *SELECT*-запросе столбцу можно назначить новое имя с помощью оператора *as*. Например, результату функции *count( )* присваивается псевдоним *total*:

Select count (distinct skill\_name)as total from Skill;

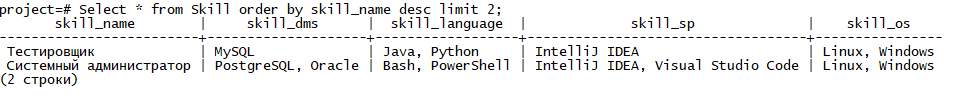


Использование функций в конструкции *where* приведет к ошибке. В следующем примере показана попытка извлечения из таблицы *Skill* записи с максимальным значением поля *skill\_name*:

Select \* from Skill where skill\_name=MAX(skill\_name);

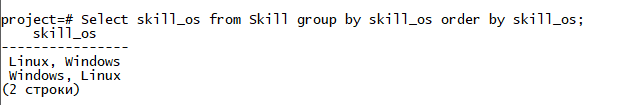
Решение задачи следует искать в использовании конструкции *order by*:

Select \* from Skill order by skill\_name desc limit 2;



Для извлечения уникальных записей используют конструкцию *group by* с именем столбца, по которому группируется результат:

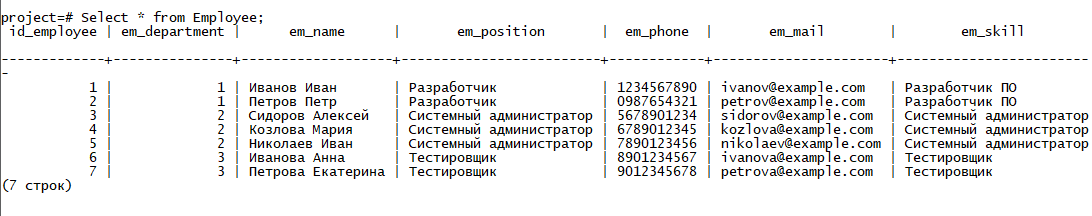
Select skill\_os from Skill group by skill\_os order by skill\_os;



При использовании *group by* возможно использование условия *where*:

Часто при задании условий требуется ограничить выборку по результату функ­ции (например, выбрать каталоги, где число товарных позиций больше 5). Использование для этих целей конструкции *where* приводит к ошибке. Для решения этой проблемы вместо ключевого слова *where* используется ключевое слово *having*, располагающееся за конструкцией *group by*:

Select \* from Employee;



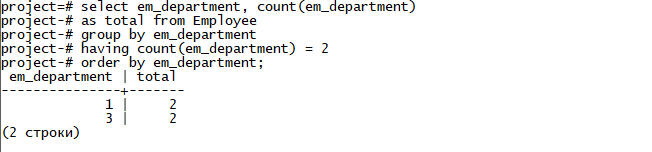
select em\_department, count(em\_department)

as total from Employee

group by em\_department

having count(em\_department) = 2

order by em\_department;



При этом в случае использования ключевого слова *where* сначала производится выборка из таблицы с применением условия и лишь затем группировка результата, а в случае использования ключевого слова *having* сначала происходит группировка таблицы и лишь затем выборка с применением условия. Допускается использование условия *having* без группировки *group by*.

JOIN позволяет совместно использовать информацию из нескольких таблиц в одном запросе, что делает его мощным инструментом для анализа и извлечения данных из базы данных.

Существует несколько видов JOIN в psql:

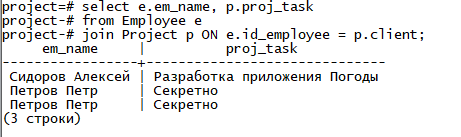
* INNER JOIN: Возвращает только те строки, для которых есть совпадение в обеих таблицах.
* LEFT JOIN (или LEFT OUTER JOIN): Возвращает все строки из левой таблицы (первой указанной в запросе) и соответствующие строки из правой таблицы.
* RIGHT JOIN (или RIGHT OUTER JOIN): Возвращает все строки из правой таблицы и соответствующие строки из левой таблицы.
* FULL JOIN (или FULL OUTER JOIN): Возвращает все строки из обеих таблиц, объединяя их по условию JOIN.

Предположим, что вы хотите получить информацию о сотрудниках и проектах, над которыми они работают.

select e.em\_name, p.proj\_task

from Employee e

join Project p ON e.id\_employee = p.client;



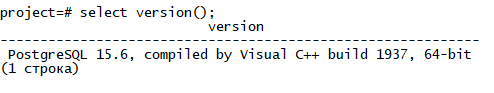
**Использование функций**.

Для решения конкретных задач при выборке данных из базы удобны встроенные функции PostgreSQL. Они предназначены для использования в выражениях SELECT и WHERE, а также в выражениях GROUP BY для группировки данных (подробнее о группировке см. выше).

Каждая функция имеет уникальное имя и может принимать ноль или более аргументов, которые перечисляются через запятую в круглых скобках. Даже если функция не требует аргументов, круглые скобки необходимо указывать. Пробелы между именем функции и круглыми скобками запрещены.

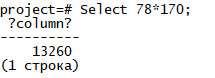
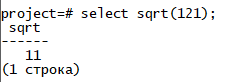
В PostgreSQL доступно множество функций, наиболее полезные из которых приведены в документации. Например, можно использовать функцию для получения версии сервера PostgreSQL:

select version();



Отметим также возможность использования оператора *SELECT* без таблиц вообще. В такой форме *SELECT* можно использовать как калькулятор:

select 78\*170; select sqrt(121);

Можно вычислить любое выражение без указания таблиц, получив доступ ко всему разнообразию математических и других операторов и функций. Возможность выполнять математические расчеты на уровне *SELECT* позволяет проводить финансовый анализ значений таблиц и отображать полученные результаты в отчетах. Во всех выражениях PostgreSQL (как в любом языке программирования) можно использовать скобки, чтобы контролировать порядок вычислений.

**Операторы**. Под операторами подразумеваются конструкции языка, которые производят преобразование данных. Данные, над которыми совершается операция, называются операндами.

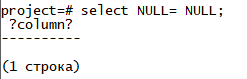
В PostgreSQL используются три типа операторов:

* арифметические операторы;
* операторы сравнения;
* логические операторы.

*Арифметические операции.*В PostgreSQL используются обычные арифметические операции: сложение (+), вычитание (–), умножение (\*), деление (/) и целочисленное деление *DIV* (деление и отсечение дробной части). Деление на 0 дает безопасный результат *NULL*.

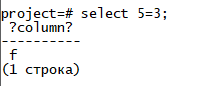
*Операторы сравнения.* При работе с операторами сравнения необходимо помнить о том, что, за ис­ключением нескольких особо оговариваемых случаев, сравнение чего-либо со зна­чением *NULL* дает в результате *NULL*. Это касается и сравнения значения *NULL* со значением *NULL*:

select *NULL= NULL;*



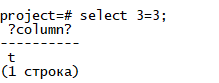
Вот так будет выглядеть false-значение (f):

select *5=3;*



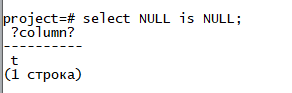
Вот так будет выглядеть true-значение (t):

select *3=3;*



Корректнее использовать следующий запрос:

select *NULL is NULL;*



Поэтому следует быть предельно внимательными при работе с операторами сравнения, если операнды могут принимать значения *NULL*.

Наиболее часто используемые операторы сравнения приведены в табл. 2.

*Логические операторы.* PostgreSQL поддерживает все обычные логические операции, которые можно использовать в выражениях. Логические выражения в PostgreSQL могут принимать значения 1 (истина), 0 (ложь) или *NULL*.

Кроме того, следует учитывать, что PostgreSQL интерпретирует любое ненулевое значение, отличное от *NULL*, как значение «истина». Основные логические операторы приведены в табл. 3.

**Все виды операторов:**

Таблица 2

|  |  |
| --- | --- |
| **Оператор** | **Значение** |
| **=** | Оператор равенства. Возвращает 1 (истина), если операнды равны, и 0 (ложь), если не равны |
| <=> | Оператор эквивалентности. Аналогичен обычному равенству, но возвращает только два значения: 1 (истина) и 0 (ложь). *NULL* не возвращает |
| <> | Оператор неравенства. Возвращает 1 (истина), если операнды не равны, и 0 (ложь), если равны |
| < | Оператор «меньше». Возвращает 1 (истина), если левый операнд меньше правого, и 0 (ложь) – в противном случае |
| <= | Оператор «меньше или равно». Возвращает 1 (истина), если левый операнд меньше правого или они равны, и 0 (ложь) – в противном случае |
| > | Оператор «больше». Возвращает 1 (истина), если левый операнд больше правого, и 0 (ложь) – в противном случае |
| >= | Оператор «больше или равно». Возвращает 1 (истина), если левый операнд больше правого или они равны, и 0 (ложь) – в противном случае |
| *n BETWEEN min   AND max* | Проверка диапазона. Возвращает 1 (истина), если проверяемое значение *n* находится между *min* и *max*, и 0 (ложь) – в противном случае |
| *IS NULL* и  *IS NOT NULL* | Позволяют прове­рить, является ли значение значением *NULL* или нет |
| *n IN (множество)* | Принадлежность к множеству. Возвращает 1 (истина), если проверяемое значение *n* входит в список, и 0 (ложь) – в противном случае. В качестве множества может использоваться список литеральных значений или выражений или подзапрос |

Таблица 3

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Пример** | **Значение** |
| *AND* | *n AND m* | Логическое *И*: истина *AND* истина = истина,  ложь *AND* любое = ложь. Все остальные выражения оцениваются как *NULL* |
| *OR* | *n OR m* | Логическое *ИЛИ*: истина *OR* любое = истина,  *NULL* *OR* ложь = *NULL,*  *NULL OR NULL* = *NULL,* ложь *OR* ложь = ложь |
| *NOT* | *NOT n* | Логическое НЕТ: *NOT* истина = ложь, *NOT* ложь = истина.  *NOT NULL = NULL* |
| *XOR* | *n XOR m* | Логическое *исключающее ИЛИ*: истина *XOR* истина = ложь,  истина *XOR* ложь = истина, ложь *XOR* истина = истина,  ложь *XOR* ложь = ложь,  *NULL XOR* любое *= NULL*, любое *XOR NULL = NULL* |

**Временные таблицы**.

Когда необходимо сохранить результирующую таблицу, прибегают к временным таблицам. Создание временных таблиц осуществляется при помощи оператора *CREATE temporary table,* синтаксискоторого ничем не отличается от синтаксиса оператора *CREATE table*.

Временная таблица автоматически удаляется по завершении соединения с сервером, а ее имя действительно только в течение данного соединения. Это означает, что два разных клиента могут использовать временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем.

**Подзапросы.**

Представляют собой запросы, вложенные в другие SQL-запросы. Они могут быть использованы для выполнения более сложных операций, где результат одного запроса используется в качестве условия или данных для другого запроса.

Вот основные характеристики и особенности подзапросов:

1. Вложенность: Подзапросы могут быть вложены друг в друга или использованы внутри других SQL-конструкций, таких как операторы SELECT, INSERT, UPDATE и DELETE.

2. Целевые операции: Подзапросы могут выполняться для различных целей, таких как фильтрация результатов, сравнение значений, определение условий и дополнительная обработка данных.

3. Местоположение: Подзапросы могут находиться в различных частях SQL-запроса, таких как в блоке WHERE, HAVING, FROM или SELECT, в зависимости от того, какие данные требуется обработать.

4. Виды подзапросов: В PostgreSQL подзапросы могут быть коррелированными и некоррелированными. Коррелированные подзапросы зависят от внешнего запроса и могут ссылаться на значения внешних столбцов. Некоррелированные подзапросы выполняются независимо от внешнего запроса.

5. Операторы: В подзапросах можно использовать различные операторы сравнения, логические операторы и агрегатные функции для обработки данных и вычисления результатов.

Пример использования подзапросов:

SELECT \*

FROM Client

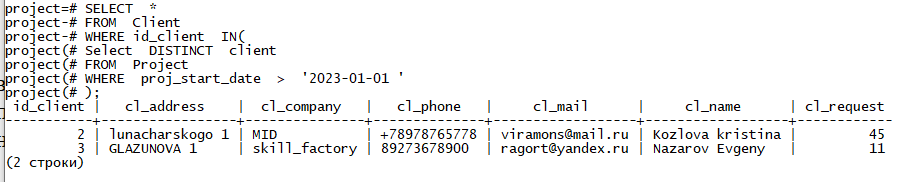
WHERE id\_client IN(

Select DISTINCT client

FROM Project

WHERE proj\_start\_date > '2023-01-01 '

);

****

Этот запрос использует подзапрос для выбора уникальных идентификаторов клиентов (client) из таблицы Project, где дата начала проекта (proj\_start\_date) позднее 1 января 2023 года. Затем основной запрос выбирает всех клиентов из таблицы Client, чьи идентификаторы присутствуют в результате подзапроса, то есть клиентов, у которых есть проекты, начавшиеся после указанной даты.

**Начало выполнения Лабораторной работы № 3**

**Теоретические сведения**

Рассмотрим следующие вопросы:

* использование объединений в запросах к нескольким таблицам;
* создание вложенных запросов.

**Практическая работа**

При выполнении лабораторной работы необходимо придумать 10 запросов:

* для заданной предметной области построить многотабличные запросы на выборку с использованием объединения;
* для заданной предметной области построить запросы на выборку, содержащий вложенный запрос;
* для заданной предметной области построить многотабличные запросы объединение таблиц.
* для заданной предметной области построить запрос, выполняющий вычисления и сравнения.
* \*для заданной предметной области построить запрос, затрагивающий все таблицы вашей базы данных отсортировав их.
* составить отчет по лабораторной работе.

**Пример выполнения работы**

1. Создадим простой запрос на выборку к таблице *Project*, который выводит два айдишника, первый как самый поздний проект для сдачи, другой как самый ближайший

select

(select id\_project

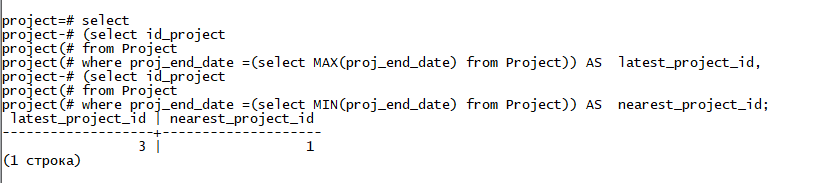
from Project

where proj\_end\_date =(select MAX(proj\_end\_date) from Project)) AS latest\_project\_id,

(select id\_project

from Project

where proj\_end\_date =(select MIN(proj\_end\_date) from Project)) AS nearest\_project\_id;

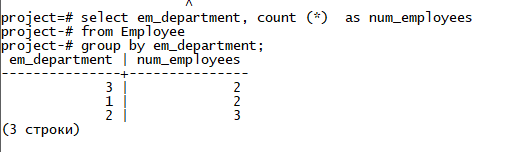


2. Создадим простой запрос на выборку к таблице Employee, который будет подсчитывать количество сотрудников в каждом отделе*.* Для этого используем функцию *count( )* вместе с выражением *group by*:

select em\_department, count (\*) as num\_employees

from Employee

group by em\_department;

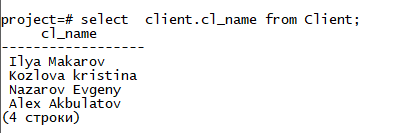


**Создание сложных запросов на выборку**

В реаль­ных приложениях часто требуется использовать сразу несколько таблиц БД. Запросы, которые обращаются одновременно к нескольким таблицам, называются многотабличными или сложными запросами.

**Абсолютные ссылки на базы данных и таблицы**. В запросе мож­но прямо указывать необходимую БД и таблицу. Напри­мер, можно представить ссылку на столбец *cl\_name* из таблицы *Client* в виде *client.cl\_name.* Аналогично, можно уточнить БД, таблица из которой упоминается в запросе. Если необходимо, то вместе с БД и таблицей можно указать и столбец, например:

select client.cl\_name from Client;



При использовании сложных запросов это позволяет избежать двусмысленности при указании источника необходи­мой информации.

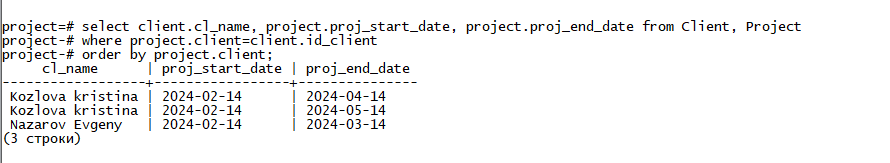
**Использование объединений для запросов к нескольким таблицам**. Хорошо спроектированная реляционная БД эффективна из-за связей между таблицами. При выборе информации из нескольких таблиц такие связи называют объединениями.

В качестве примера объединения двух таблиц рассмотрим запрос, извлекающий из employee имена клиентов с датами начала и конца их проектов:

select client.cl\_name, project.proj\_start\_date, project.proj\_end\_date from Client, Project

where project.client=client.id\_client

order by project.client;

 Или

select

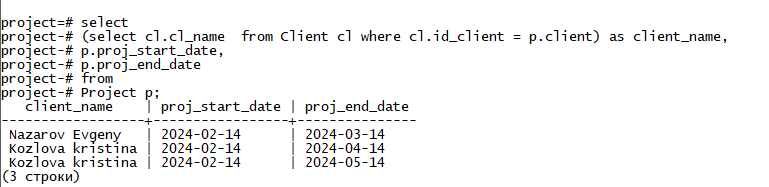
(select cl.cl\_name from Client cl where cl.id\_client = p.client) as client\_name,

p.proj\_start\_date,

p.proj\_end\_date

from

Project p;



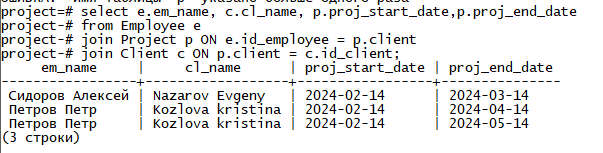
Или

select e.em\_name, c.cl\_name, p.proj\_start\_date,p.proj\_end\_date

from Employee e

join Project p ON e.id\_employee = p.client

join Client c ON p.client = c.id\_client;



Объединение нескольких таблиц аналогично объединению двух таблиц. Например, зная айди проекта, выдать все отделы, работающие над ним, сначала выдать сотрудников из одного отдела, а потом сотрудников из другого:

**Самообъединение таблиц**. Можно объединить таб­лицу саму с собой (когда интересуют связи между строками одной и той же таблицы). Предположим, что мы хотим найти других сотрудников, у которых навык имена совпадают. Мы можем использовать самообъединение следующим образом:.

Insert into Employee (em\_department, em\_name, em\_position, em\_phone, em\_mail, em\_skill)

Values

(1, 'John Doe' , 'Software Engineer ' , ' 123456789' , ' [john@exampple.com](mailto:john@exampple.com)', 'Разработчик ПО'),

(2, 'Jane Smith' , 'System Administrator' , ' 987654321' , ' [jane@exampple.com](mailto:jane@exampple.com)', 'Системный администратор'),

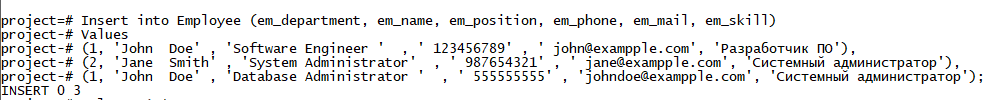
(1, 'John Doe' , 'Database Administrator ' , ' 555555555' , '[johndoe@exampple.com](mailto:johndoe@exampple.com)', 'Системный администратор');

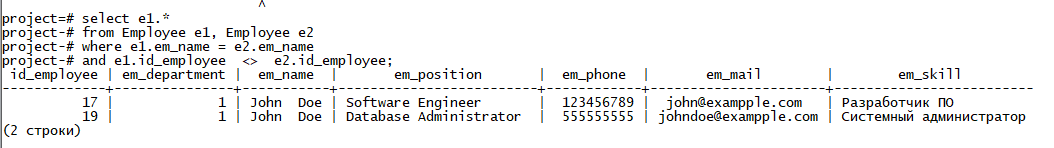
select e1.\*

from Employee e1, Employee e2

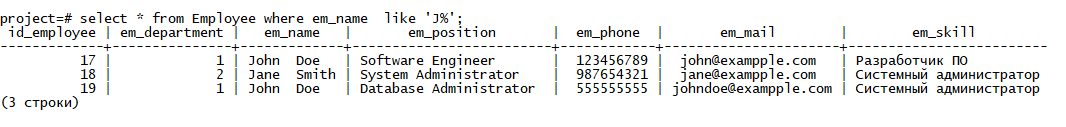
where e1.em\_name = e2.em\_name

and e1.id\_employee <> e2.id\_employee;





А так можно отыскать всех, у кого имена начинаются на “j”:  
select \* from Employee where em\_name like 'J%';



**Основное объединение**. Набор таблиц, перечисленных в выражении *FROM* и разделенных запятыми, – это декартово произведение (полное или перекрестное объединение), которое возвращает полный набор ком­бинаций. Добавление к нему условного выражения *WHERE* превраща­ет его в объединение по эквивалентности, ограни­чивающее число возвращаемых запросом строк.

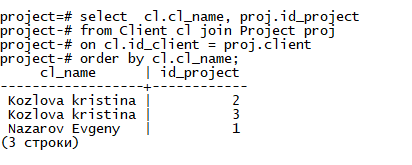
Вместо запятой в выражении *FROM* можно использовать ключевое слово *JOIN*. В этом случае вместо *WHERE* лучше использовать ключевое слово *ON*:

select cl.cl\_name, proj.id\_project

from Client cl join Project proj

on cl.id\_client = proj.client

order by cl.cl\_name;

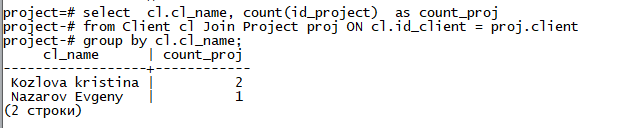


Вместо *JOIN* с тем же результатом можно использовать *CROSS JOIN* (перекрестное объединение) или *INNER JOIN* (внутреннее объединение). Пример запроса, выдающего клиентов и сколько проектов они заключили:

select cl.cl\_name, count(id\_project) as count\_proj

from Client cl Join Project proj ON cl.id\_client = proj.client

group by cl.cl\_name;

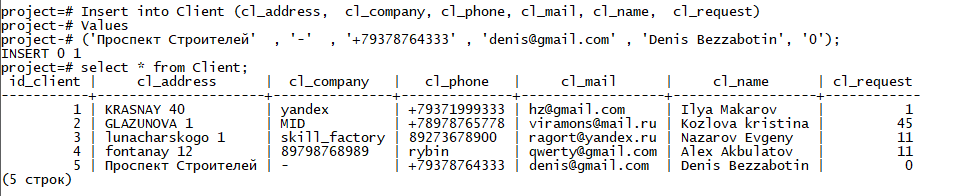


Допустим, в списке клиентов появляется новый клиент «Kandratij»:

Insert into Client (cl\_address, cl\_company, cl\_phone, cl\_mail, cl\_name, cl\_request)

Values

('Проспект Строителей' , '-' , '+79378764333' , '[denis@gmail.com](mailto:denis@gmail.com)' , 'Denis Bezzabotin', '0');

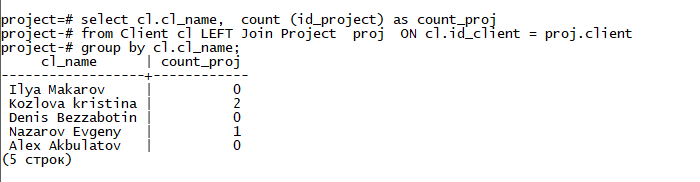


Предыдущий запрос не отразит наличие нового клиента (таблица *project* не содержит записей, относящихся к новому клиенту). Выходом является использование левого объединения (таблица *client* должна быть левой таблицей):

select cl.cl\_name, count (id\_project) as count\_proj

from Client cl LEFT Join Project proj ON cl.id\_client = proj.client

group by cl.cl\_name;



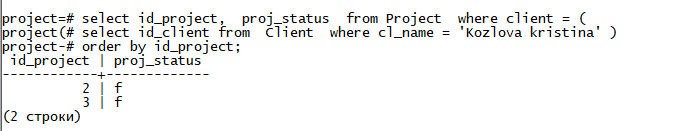
**Вложенный запрос**. Позволяет использовать результат, возвращаемый одним запросом, в другом запросе. Так как результат возвращает только оператор *select*, то в качестве вложенного запроса всегда выступает *SELECT*-запрос. В качестве внешнего запроса может выступать запрос с участием любого SQL-оператора: *select, insert, update, delete, create table* и др.

Пусть требуется вывести номер и статус проекта из таблицы *project* для клиентов «Базы данных» таблицы *client*:

select id\_project, proj\_status from Project where client = (

select id\_client from Client where cl\_name = 'Kozlova kristina' )

order by id\_project;



Получить аналогичный результат можно при помощи многотабличного запроса, но имеется ряд задач, которые решаются только при помощи вложенных запросов. Вложенный запрос может применяться не только с условием *WHERE*, но и в конструкциях *DISTINCT, GROUP BY, ORDER BY, LIMIT* и т. д. Различают:

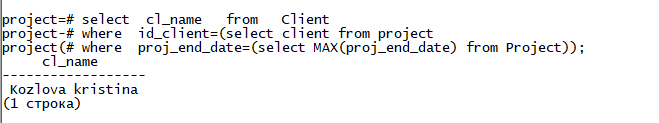
* вложенные запросы, возвращающие одно значение;
* вложенные запросы, возвращающие несколько строк.

В первом случае вложенный запрос возвращает скалярное значение или литерал, которое используется во внешнем запросе (подставляет результат на место своего выполнения). Например, необходимо определить имя клиента, c самой ранней сдачей проекта:

select cl\_name from Client

where id\_client=(select client from project

where proj\_end\_date=(select MAX(proj\_end\_date) from Project));



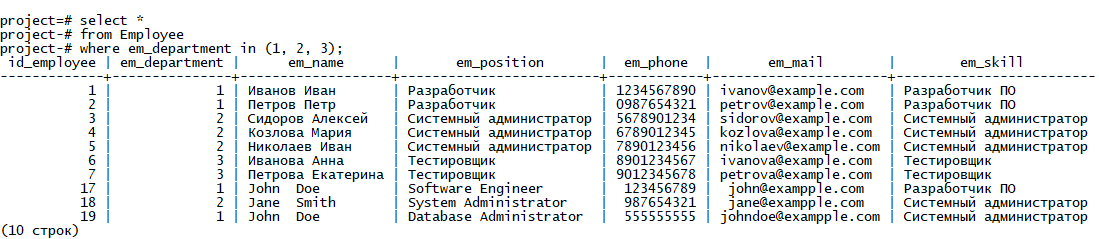
Наиболее часто вложенные запросы используются в операциях сравнения в условиях, которые задаются ключевыми словами *WHERE, HAVING* или *ON*.

Чтобы выбрать строки из таблицы *Employee*, допустим, нам нужно выбрать всех сотрудников из определенных отделов, следует воспользоваться конструкцией *IN*:

select \*

from Employee

where em\_department in (1, 2, 3);

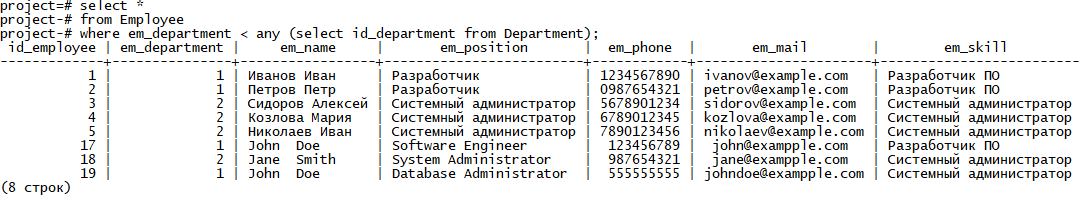


Ключевое слово *ANY* может применяться с использованием любого оператора сравнения. Используется логика *ИЛИ*, т. е. достаточно, чтобы срабатывало хотя бы одно из многих условий. Запрос вида *WHERE X > ANY (SELECT Y …)* можно интерпретировать как «где *X* больше хотя бы одного выбранного *Y*». Соответственно, запрос вида *WHERE X < ANY (SELECT Y …)* интерпретируется как «где *X* меньше хотя бы одного выбранного *Y*». Рассмотрим запрос, который выберет всех сотрудников, чей идентификатор отдела (em\_department) меньше, чем любой идентификатор отдела из таблицы Departament.:

select \*

from Employee

where em\_department < any (select id\_department from Department);

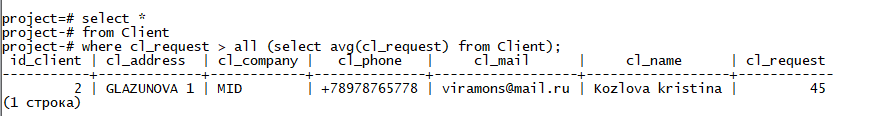


Ключевое слово *ALL* также может применяться с использованием любого оператора сравнения, но при этом используется логика *И*, то есть должны срабатывать все условия. Запрос вида *WHERE X > ALL (SELECT Y …)* интерпретируется как «где *X* больше любого выбранного *Y*». Соответственно, запрос вида *WHERE X < ALL (SELECT Y …)* интерпретируется как «где *X* меньше, чем все выбранные *Y*». Рассмотрим запрос, возвращающий всех клиентов, количество запросов которых превышает среднее кол-во:

select \*

from Client

where cl\_request > all (select avg(cl\_request) from Client);



Результирующая таблица, возвращаемая вложенным запросом, может не содержать ни одной строки. Для проверки этого факта могут использоваться ключевые слова *EXISTS* и *NOT EXISTS*.

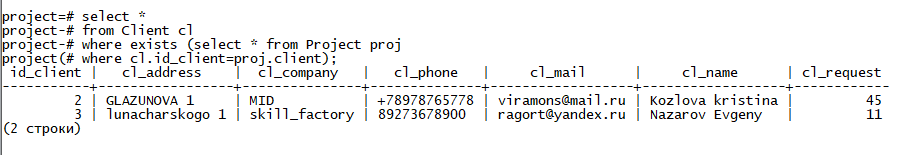
Запрос, формирующий список клиентов, имеющих хотя бы один проект, можно записать следующим образом:

select \*

from Client cl

where exists (select \* from Project proj

where cl.id\_client=proj.client);



**Пример запроса для выполнения лабораторной рабоыт:**

Создадим многотабличный запрос для поиска всех сотрудников отделов, работающих над проектом конкретного клиента. Нам нужно сначала найти отделы, которые работают над этим проектом, а затем выбрать всех сотрудников из этих отделов. select e.\*

from Employee e

where e.em\_department in (

select pd.id\_department

from Project\_Department pd

join Project p ON pd.id\_project=p.id\_project

where p.client =3

);

