Министерство науки и образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э.

Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)



**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ К ЛАБОРАТОРНЫМ РАБОТАМ**

**ПО КУРСУ «Базы данных (ЮР)»**

**Лабораторная работа № 6** **«****Введение в PostgreSQL. Изменение данных»**

Авторы:

Кудрявцев А.П., [kudryavtsevap@bmstu.ru](mailto:kudryavtsevap@bmstu.ru)

Фомин М.М.

Ланцберг А.В.

Москва, 2024

## Индексы

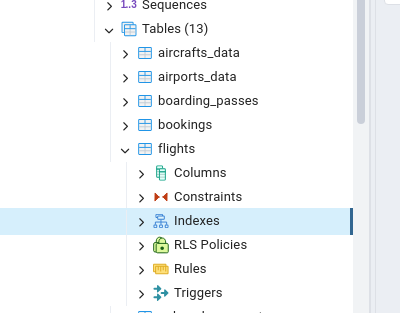
1.1. Общие сведения

Индекс — это структура данных, которая увеличивает скорость извлечения данных, предоставляя быстрый способ поиска строк в таблице. Индекс в PostgreSQL работает как предметный указатель в книге, предоставляя быструю ссылку на страницу, где можно найти определенное содержание.

PostgreSQL поддерживает различные типы индексов. Мы ограничимся рассмотрением только индексов на основе B-дерева. Индекс — специальная структура данных, которая связана с таблицей и создается на основе данных, содержащихся в ней. Основная цель создания индексов — повышение производительности функционирования базы данных.

Строки в таблицах хранятся в неупорядоченном виде. При выполнении операций выборки, обновления и удаления СУБД должна отыскать нужные строки. Для ускорения этого поиска и создается индекс. В принципе он организован таким образом: на основе данных, содержащихся в конкретной строке таблицы, формируется значение элемента (записи) индекса, соответствующего этой строке. Для поддержания соответствия между элементом индекса и строкой таблицы в каждый элемент помещается указатель на строку. Индекс является упорядоченной структурой. Элементы (записи) в нем хранятся в отсортированном виде, что значительно ускоряет поиск данных в индексе. После отыскания в нем требуемой записи СУБД переходит к соответствующей строке таблицы по прямой ссылке. Записи индекса могут формироваться на основе значений одного или нескольких полей соответствующих строк таблицы. Значения этих полей могут комбинироваться и преобразовываться различными способами. Все это определяет разработчик базы данных при создании индекса.

Для того, чтобы увидеть индексы, созданные для таблицы в pgAdmin 4, раскройте БД и выберите на дереве объектов нужную таблицу, щелкните по ней. Откроются дочерние узлы, связанные с выбранной таблицей, в том числе и индексы.



Если раскроем узел *Indexes*, то увидим перечень созданных индексов[[1]](#footnote-1) (БД demo не использует явные индексы, но мы исправим это).

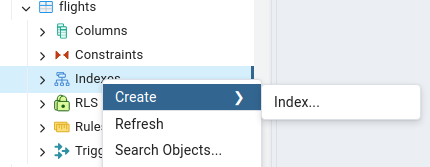
1.2. Создание индекса

Каждый индекс, который был создан самой СУБД, имеет типовое имя, состоящее из следующих компонентов: – имени таблицы и суффикса pkey — для первичного ключа; – имени таблицы, имен столбцов, по которым создан индекс, и суффикса key — для уникального ключа, а суффикс idx – для индекса. В описании также присутствует список столбцов, по которым создан индекс, и тип индекса — в данном случае это btree, т. е. B-дерево.

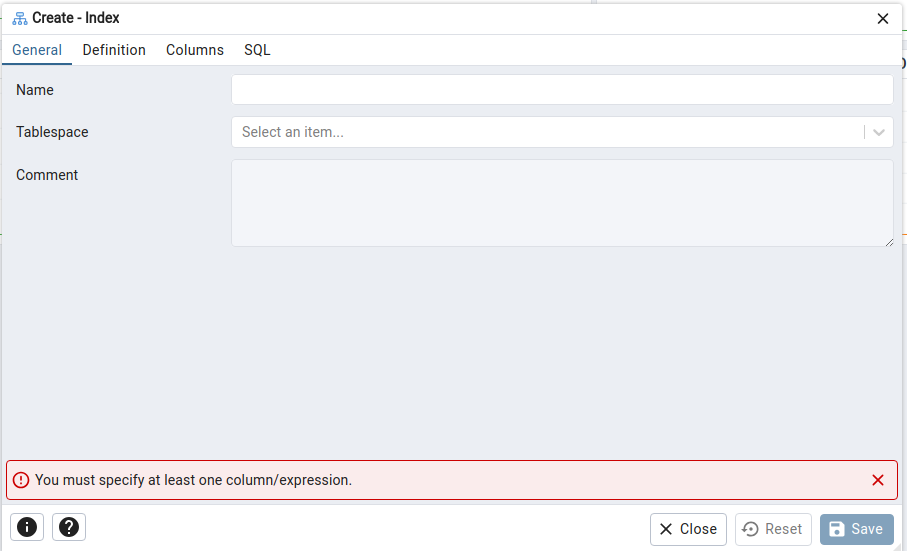
PostgreSQL может создавать индексы различных типов, но по умолчанию используется так называемое B-дерево. Такой индекс подходит для большинства типовых задач. Здесь мы будем рассматривать только индексы на основе B-дерева. Наличие индекса может ускорить выборку строк из таблицы, если он создан по столбцам, на основе значений которых и производится выборка. Поэтому, как правило, при разработке и эксплуатации баз данных не ограничиваются только индексами, которые автоматически создает СУБД, а создают дополнительные индексы с учетом наиболее часто выполняющихся выборок.

Сделаем допущение, что БД demo - очень большая и приложение, которое реализует бизнес-логику, часто обращается к таблице flights для выборки строк, содержащих заданные аэропорты вылета (departure\_airport), посадки (arrival\_airport) и день. И мы уверены, что создание индекса, содержащего поля departure\_airport,arrival\_airport и scheduled\_departure, сократит время выполнения запросов.

Создадим этот индекс. Нажмем правой клавишей на узел Indexes таблицы flights и выберем команду *Create/Index…*:



Откроется диалоговое окно создания индекса:

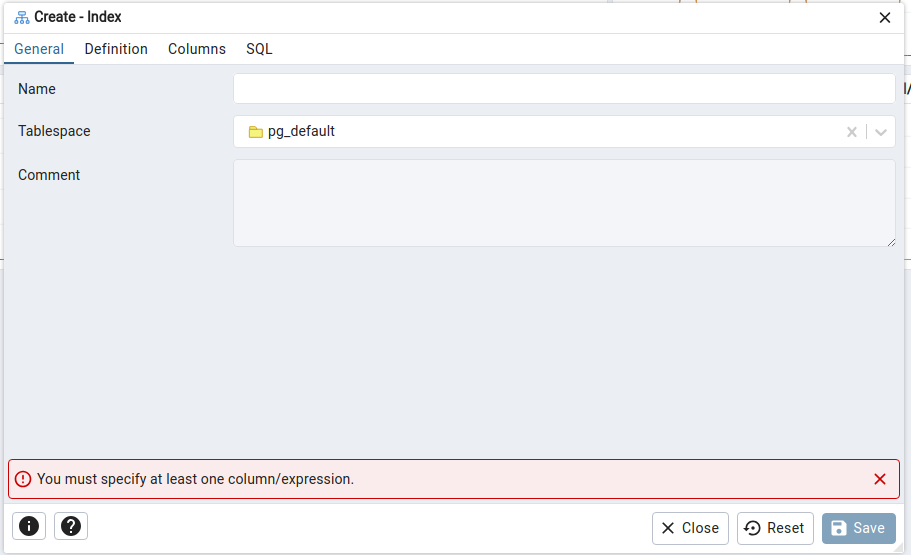


На вкладке *General* (Общие) поле Name предназначено для ввода описательного имени индекса. Если имя индекса не вводить, то Postgres сформирует его автоматически, в нашем случае оно будет таким - departure\_airport\_arrival\_airport\_scheduled\_departure\_idx, что вполне информативно. Так что оставим поле *Name* пустым.

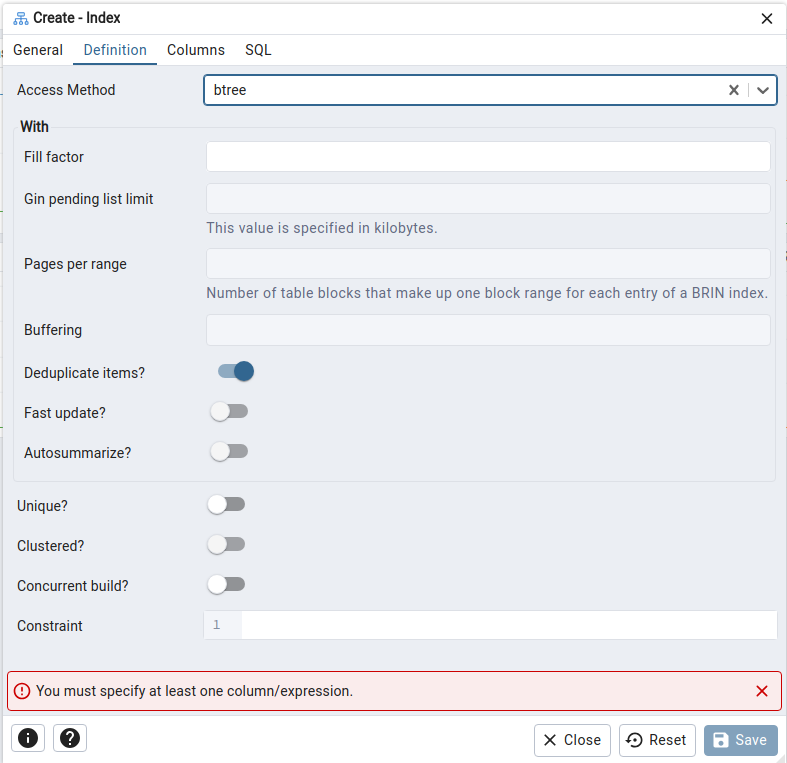
Поле *Tablespace* предоставляет нам выбор табличного пространства, где будет находится индекс. БД demo расположена в одном табличном пространстве, поэтому можем выбрать единственный вариант - *pg\_default* из выпадающего списка или оставить поле пустым.

Поле Comment не нуждается в пояснении.

Таким образом, вкладка *General* после заполнения выглядит так:

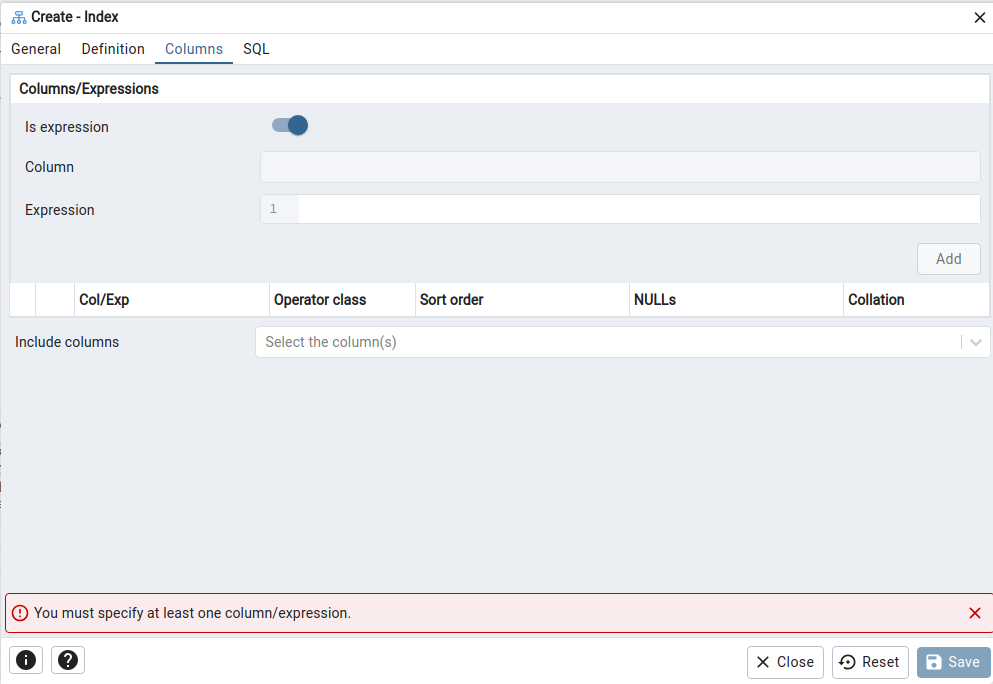


Переходим на следующую вкладку *Definition* (Определение):

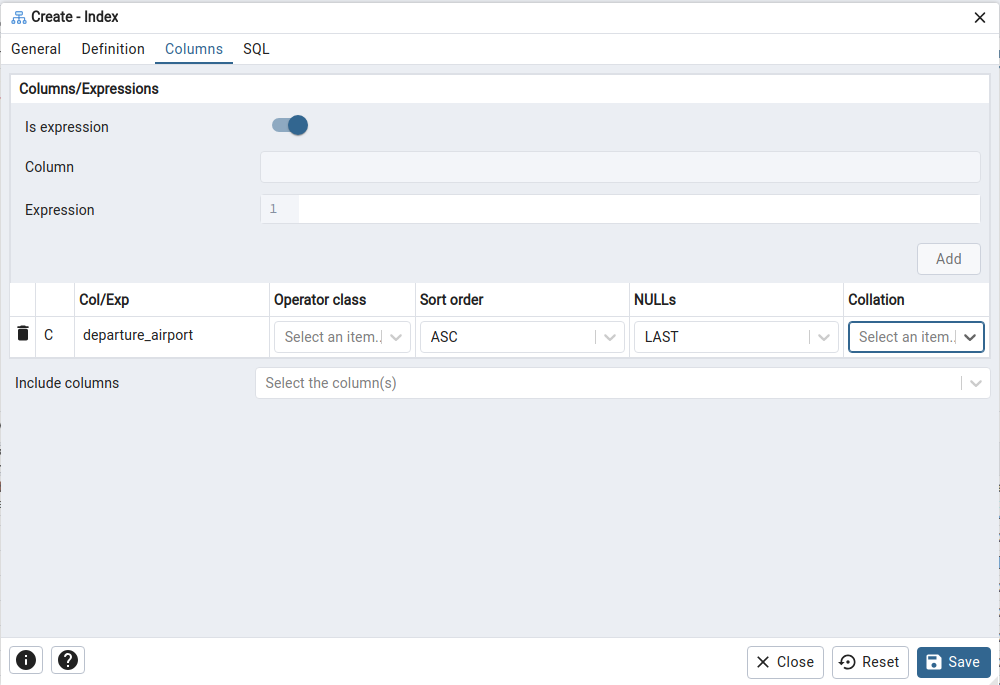


Эту вкладку мы оставляем без изменения. Но хочу обратить внимание на два поля: *Access Method* и *Unique*. Первое поле — это выбор типа индекса – btree, о котором было упомянуто выше. Второе поле задает уникальность индекса (если «включить»). То есть нет двух строк, с одинаковыми полями (или выражения), перечисленными в индексе. В нашем случае индекс не является уникальным, поскольку между аэропортами за сутки может выполняться несколько рейсов.

Переходим на вкладку *Columns* (Колонки).



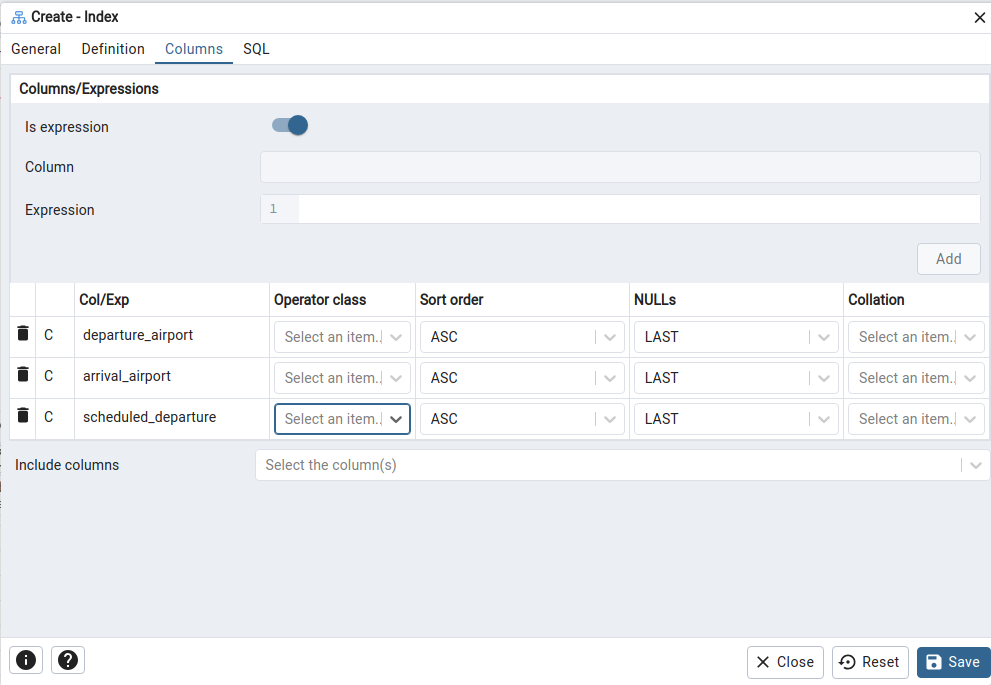
На этой вкладке мы должны перечислить поля таблицы, составляющие индекс. Наш индекс состоит из полей таблицы, поэтому гасим выключатели   
*Is expression*. И из списка Column выбираем поле *departure\_*airport, после этого нажимаем на кнопку *Add* и получаем такой вид вкладки:



Для нас наиболее интересны колонки *Sort order* и *NULLs*. *Sort order* указывает как будет выполняться сортировка по возрастанию *ASC* или по убыванию *DESC*. NULLs задает сортировку для случая, когда значение поля не определено (NULL). Если установлено *LAST*, то строка поместится после строк, имеющих значение поля, отличное от NULL. Если выбрать *FIRST*, строка помещается перед строками, имеющих значение поля отличное от NULL.

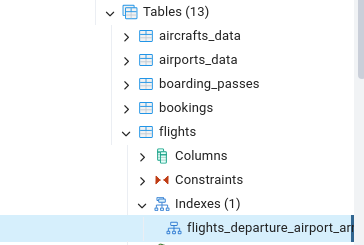
Аэропорты не имеют значения NULL, и сортировка в алфавитном порядке вполне нам подходит (как и большинстве случаях).

Аналогичным образом добавим оставшиеся два поля:



Последняя вкладка SQL содержит скрипт, сформированный согласно введенными нами данными. Сохраним.

В результате создали индекс, который отобразился под узлом *Indexes*.



1.3. Рекомендации по созданию индекса

Хотя хорошо известно, что индексы повышают производительность базы данных, следует знать, как их заставить работать должным образом. Добавление ненужных или неподходящих индексов к таблице может даже привести к снижению производительности. Ниже предоставлены некоторые рекомендации по созданию эффективных индексов в БД.

* Индекс имеет смысл, если нужно обеспечить доступ одновременно не более, чем к 4-5% данных таблицы. Альтернативой использования индекса для доступа к данным строки является полное последовательное чтение таблицы от начала до конца, что называется полным сканированием таблицы. Полное сканирование таблицы больше подходит для запросов, которые требуют извлечения большего процента данных таблицы. Помните, что применение индексов для извлечения строк требует двух операций чтения: индекса и затем таблицы.
* Избегайте создания индексов для сравнительно небольших таблиц. Для таких таблиц больше подходит полное сканирование. В случае маленьких таблиц нет необходимости в хранении данных и таблиц и индексов.
* Создавайте первичные ключи для всех таблиц. При назначении столбца в качестве первичного колюча Postgres автоматически создаст индекс по этому столбцу.
* Индексируйте столбцы, участвующие в многотабличных операциях соединения (JOIN).
* Индексируйте столбцы, которые часто используются в конструкциях WHERE.
* Индексируйте столбцы, участвующие в операциях ORDER BY и GROUP BY или других операциях, таких как UNION и DISTINCT, включающих сортировку. Поскольку индексы уже отсортированы, объем работы по выполнению необходимой сортировки данных для упомянутых операций будет существенно сокращен.
* Столбцы, стоящие из длинно-символьных строк, обычно плохие кандидаты на индексацию.
* Столбцы, которые часто обновляются, в идеале не должны быть индексированы из-за связанных с этим накладных расходов.
* Индексируйте таблицы, в которых мало строк имеют одинаковые значения.
* Сохраняйте количество индексов небольшим.
* Составные индексы могут понадобиться там, где одностолбцовые значения сами по себе не уникальны. В составных индексах первым столбцом ключа должен быть столбец, в котором количество строк с одинаковым значением минимально.

Всегда помните золотое правило индексации таблиц: индекс таблицы должен быть основан на типах запросов, которые будут выполняться над столбцами этой таблицы. Для таблицы можно создавать более одного индекса: например, можно создать индекс на столбце X, или столбце Y, или обоих сразу, а также один составной индекс на обоих столбцах. Принимая правильное решение относительно того, какие индексы следует создавать, подумайте о наиболее часто используемых типах запросов данных таблицы.

## 2. Изменение данных

2.1. Оператор INSERT

Оператор PostgreSQL INSERT позволяет вставить новую строку в таблицу. Вот основной синтаксис оператора INSERT:

INSERT INTO table1(column1, column2, …)

VALUES (value1, value2, …);

Списки столбцов и значений должны располагаться в одном и том же порядке.

В INSERT операторе есть необязательное RETURNING предложение, которое возвращает информацию о вставленной строке. Если вы хотите вернуть всю вставленную строку, используйте звездочку (\*) после RETURNING ключевого слова:

INSERT INTO table1(column1, column2, …)

VALUES (value1, value2, …)

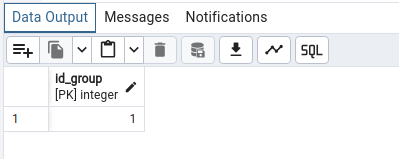
RETURNING \*;

Если вы хотите вернуть некоторую информацию о вставленной строке, вы можете указать один или несколько столбцов после предложения RETURNING. Например, следующий оператор возвращает номер id\_group вставленной строки:

INSERT INTO lab.group ( chair, num\_group, year\_start )

VALUES ('ИУ6', 1, 2022 )

RETURNING id\_group;



Чтобы вставить несколько строк в таблицу с помощью одного INSERT оператора, используйте следующий синтаксис:

INSERT INTO table\_name (column\_list)

VALUES

(value\_list\_1),

(value\_list\_2),

...

(value\_list\_n);

Чтобы вставить несколько строк и вернуть вставленные строки, вы добавляете RETURNING предложение следующим образом:

INSERT INTO table\_name (column\_list)

VALUES

(value\_list\_1),

(value\_list\_2),

...

(value\_list\_n)

RETURNING \* | output\_expression AS output\_name;

Вставка нескольких строк одновременно имеет преимущества перед вставкой одной строки за раз:

* **Производительность**. Вставка нескольких строк в одном операторе часто более эффективна, чем несколько отдельных вставок, поскольку уменьшает количество двусторонних обращений между приложением и сервером PostgreSQL.
* **Атомарность**: весь INSERT оператор является атомарным, что означает, что либо все строки вставляются, либо ни одна не вставляется. Это обеспечивает согласованность данных.

Следующий оператор вставляет три строки в таблицу student:

INSERT INTO lab.student( id\_group, name, card, year\_start )

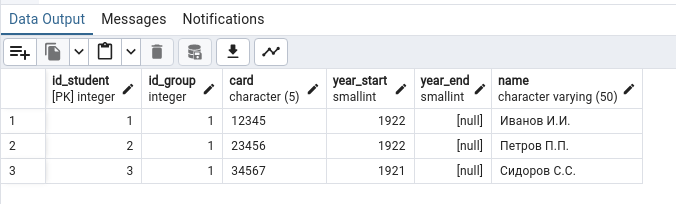
VALUES( 1, 'Иванов И.И.', '12345', 1922 ),

( 1, 'Петров П.П.', '23456', 1922 ),

( 1, 'Сидоров С.С.', '34567', 1921 );

Посмотрим содержимое таблицы student:

SELECT \* FROM lab.student;



2.2. Оператор UPDATE

Оператор PostgreSQL UPDATE позволяет обновлять данные в одном или нескольких столбцах одной или нескольких строк таблицы. Вот основной синтаксис оператора UPDATE:

UPDATE table\_name

SET column1 = value1,

column2 = value2,

...

WHERE condition;

Условие condition определяет, какие строки следует изменить. Если вы опустите предложение WHERE, оператор UPDATE обновит все строки в таблице.

В операторе UPDATE cесть необязательное предложение RETURNING, которое возвращает обновленные строки:

UPDATE table\_name

SET column1 = value1,

column2 = value2,

...

WHERE condition

RETURNING \* | output\_expression AS output\_name;

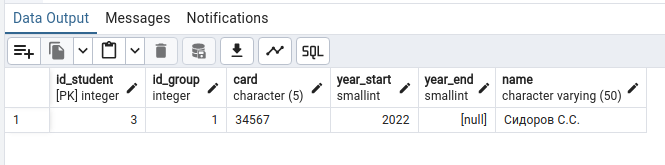
В следующем примере используется оператор UPDATE для обновления даты создания записи в таблице accounts:

UPDATE lab.student

SET year\_start = 2022

WHERE id\_student = 3

RETURNING \*;



3.3. Оператор DELETE

Оператор PostgreSQL DELETE позволяет удалить одну или несколько строк из таблицы. Ниже показан основной синтаксис оператора DELETE:

DELETE FROM table\_name

WHERE condition;

Предложение WHERE condition определяет, какие строки следует удалить.

Предложение WHERE является необязательным. Если вы опустите это WHERE, инструкция DELETE удалит все строки в таблице.

DELETE FROM table\_name

WHERE condition

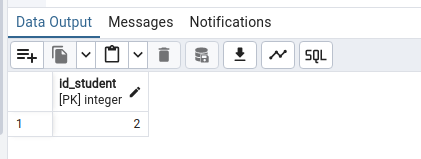
RETURNING (select\_list | \*);

Если вы хотите вернуть значения в определенном столбце, вы также можете указать их после ключевого слова RETURNING.

DELETE FROM lab.student

WHERE id\_student = 2

RETURNING id\_student;



## 3. Контрольные вопросы

1. В каких случаях имеет смысл создавать индексы? Какие колонки следует включать в индекс и почему?
2. Имеет ли значение порядок указания колонок при создании индекса?
3. Можно ли не указывать в операторе INSERT INTO список столбцов, а ограничиться только таблицей?
4. Если в операторе INSERT INTO в списке столбцов указаны не все столбцы, то что произойдет?
5. Если удалим в таблице student все строки с помощью оператора DELETE, а потом вставим в таблицу строку, например:

INSERT INTO student( name, …)  
VALUES ('Моргунов М.М.', …)  
  
Чему будет равен id\_student?

## 5. Практическое задание

1. Заполнить данными созданные таблицы с использованием SQL-скриптов для вставки соответствующих строк в таблицы БД. Объем данных должен составлять не менее 10 экземпляров для каждой из стержневых сущностей и 1000 экземпляров для целевой сущности[[2]](#footnote-2).
2. Необходимо подготовить три запроса к одной таблице, содержащий фильтрацию по нескольким полям.
3. Необходимо определить индексы, позволяющие ускорить запросы. Определить какие индексы следует сделать уникальными.
4. Сформулировать по три запроса на изменение и удаление из базы данных. Запросы должны быть сформулированы в терминах предметной области.
5. Написать SQL-скрипты, для которые будет срабатывать ограничения целостности.

Использование языков программирования для заполнения таблиц приветствуется.

## 6. Список использованных источников

1. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. - СПб.: БХВ-Петербург, 2018.
2. Postgres Pro Standard: Документация: <https://postgrespro.ru/docs/postgrespro>
3. pgAdmin Documentation.

<https://www.pgadmin.org/docs/>

1. Кроме тех случае, когда индексы создаются неявно, например, при создании ограничения уникальности (UNIQUE). [↑](#footnote-ref-1)
2. Для заполнения таблиц можно использовать функцию generate\_series. Например, SELECT x FROM generate\_series( 1, 1000 ) x выводит числа от 1 до 1000. [↑](#footnote-ref-2)