Базы данных

Лекция 08 – Основы SQL

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2023

2022.10.20

Оглавление

Определение данных

Создание таблицы

Для создания таблицы используется команда **`CREATE TABLE`**.
В этой команде необходимо указать как минимум:
- имя новой таблицы;
- имена и типы данных каждого столбца.
Например:

```
CREATE TABLE dumb_table (
    first_column     text,
    second_column    integer
);
```

Создается таблица **`dumb_table`** с двумя столбцами.
Первый столбец называется **`first_column`** и имеет тип данных **`text`**.
Второй столбец называется **`second_column`** и имеет тип **`integer`**.
Имена таблицы и столбцов должны быть в синтаксисе идентификаторов.
Имена типов в большинстве случаев тоже являются идентификаторами (есть исключения).
Список столбцов заключается в скобки, а его элементы разделяются запятыми.
В именах таблиц и столбцов следует отражать, какие данные они будут содержать.

```
CREATE TABLE products (
    product_no   integer,  -- int32_t
    name         varchar,  -- как называется
    price        numeric   -- огромное вещественное число с фиксированной запятой
);
```

Попытка повторного создания

```
CREATE TABLE dumb_table (
    first_column     text,
    second_column    integer
);
ОШИБКА:  отношение "dumb_table" уже существует
```

```
CREATE TABLE IF NOT EXISTS table_name (
    ...
);
ЗАМЕЧАНИЕ:  отношение "table_name" уже существует, пропускается
CREATE TABLE
```

Удаление таблицы

```
DROP TABLE dumb_table;
DROP TABLE products;
```

Попытка удаления несуществующей таблицы считается ошибкой.

Тем не менее в SQL-скриптах часто применяют безусловное удаление таблиц перед созданием, игнорируя все сообщения об ошибках, так что они выполняют свою задачу независимо от того, существовали таблицы или нет.

Можно использовать вариант **`DROP TABLE IF EXISTS`** (не в стандарте SQL).

Значения по умолчанию

Столбцу можно назначить значение по умолчанию, которое будет присвоено столбцу при добавлении строки в том случае, если значение явно не указано.

Если значение по умолчанию не объявлено явно, им считается значение **NULL**. Это имеет смысл, если полагать, что **NULL** представляет неизвестные данные.

Значения по умолчанию указываются при определении таблицы после типа данных столбца.

```
CREATE TABLE products (
    product_no       integer,
    name             varchar,
    price            numeric      DEFAULT 9.99
);
```

Значение по умолчанию может быть также выражением, которое вычисляется не когда создаётся таблица, а в момент присваивания значения по умолчанию.

Например, столбцу типа **timestamp** в качестве значения по умолчания часто присваивается **CURRENT_TIMESTAMP**, чтобы в момент добавления строки в нём оказалось текущее время.

DEFAULT является *ограничением* (CONSTRAINT).

Генерация «последовательных номеров» для всех строк:

```
CREATE SEQUENCE products_product_no_seq AS integer;
...
CREATE TABLE products (
    product_no integer DEFAULT nextval('products_product_no_seq'),
    ...
);
```

Функция **nextval()** возвращает следующее значение из генератора последовательности.
Или, что аналогично,

```
CREATE TABLE products (
    product_no SERIAL,
    ...
);
```

Генерируемые столбцы

Генерируемый столбец является столбцом особого рода, который всегда вычисляется из других.
Есть два типа генерируемых столбцов:
- сохранённые;
- виртуальные.

Сохранённый (STORED) генерируемый столбец вычисляется при записи (добавлении или изменении) и занимает место в таблице так же, как и обычный столбец.

Виртуальный генерируемый столбец не занимает места и вычисляется при чтении.

В настоящее время в Postgres реализованы только сохранённые генерируемые столбцы.
Для создания генерируемого столбца используется предложение **GENERATED ALWAYS AS**

```
CREATE TABLE people (
    ...,
    height_cm numeric, -- рост в сантиметрах
    height_in numeric GENERATED ALWAYS AS (height_cm / 2.54) STORED
);
```

Ключевое слово **STORED**, определяющее тип хранения генерируемого столбца, обязательно.
Произвести запись непосредственно в генерируемый столбец нельзя.
Поэтому в командах **INSERT** или **UPDATE** нельзя задать значение для таких столбцов.

Отличия генерируемых столбцов от столбцов со значением по умолчанию

1) значение столбца по умолчанию вычисляется один раз, когда в таблицу впервые вставляется строка и никакое другое значение не задано;

2) значение генерируемого столбца может меняться при изменении строки и не может быть переопределено;

3) выражение значения по умолчанию не может обращаться к другим столбцам таблицы, в то время как генерирующее выражение чаще всего именно это и делают;

4) в выражении значения по умолчанию могут вызываться «*изменчивые*» функции, например, **`random()`** или функции, зависящие от времени, а для генерируемых столбцов это не допускается.

Ограничения и особенности генерирующих выражений и таблиц, их содержащих:

1) в генерирующем выражении могут использоваться только постоянные функции и не могут фигурировать подзапросы или ссылки на какие-либо значения, не относящиеся к данной строке;

2) генерирующее выражение не может обращаться к другому генерируемому столбцу;

3) генерирующее выражение не может обращаться к системным столбцам, кроме **`tableoid`**;

4) для генерируемого столбца нельзя задать значение по умолчанию;

5) генерируемый столбец не может быть частью ключа секционирования.

6) есть ограничения по наследованию.

7) права доступа к генерируемым столбцам существуют отдельно от прав для столбцов, из которых он вычисляется, поэтому их можно организовать так, чтобы определённый пользователь мог прочитать генерируемый столбец, но не мог читать оригиналы.

Секционирование — таблица разделяется на несколько частей меньшего размера:

- горизонтальное секционирование (части таблицы LOGS содержат строки по месяцам или фикс.)
- вертикальное секционирование (части таблицы NEWS содержат разные ее столбцы).

Ограничения

- ограничение DEFAULT;
- ограничения-проверки (CHECK);
- ограничения NOT NULL;
- ограничения уникальности;
- первичные ключи;
- внешние ключи;
- ограничения-исключения.

Ограничения-проверки

Ограничение-проверка — наиболее общий тип ограничений. В его определении можно указать, что значение данного столбца должно удовлетворять логическому выражению (проверке истинности). Например, цену товара можно ограничить положительными значениями:

```
CREATE TABLE products (
    product_no integer,
    name       varchar,
    price      numeric CHECK (price > 0)
);
```

Ограничение определяется после объявления типа данных, аналогично тому, как определяется значение по умолчанию.

Значения по умолчанию и ограничения могут указываться в любом порядке.

Ограничение-проверка состоит из ключевого слова **CHECK**, за которым идёт выражение в скобках.

Выражение должно включать столбец, для которого задаётся ограничение, иначе оно не имеет большого смысла.

Ограничению можно присвоить отдельное имя — это улучшит сообщения об ошибках и позволит ссылаться на это ограничение, если придется его изменять:

```
CREATE TABLE products (
    product_no integer,
    name       varchar,
    price      numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

Синтаксис:

[CONSTRAINT *constraint_name*] *constraint_definition*

Ограничение-проверка может также ссылаться на несколько столбцов. Например, чтобы гарантировать, что цена со скидкой будет всегда меньше обычной:

```
CREATE TABLE products (
    product_no       integer,
    name             varchar,
    price            numeric CHECK (price > 0),                -- ограничение столбца
    discounted_price numeric CHECK (discounted_price > 0),     -- ограничение столбца
                             CHECK (price > discounted_price) -- ограничение таблицы
);
```

Первые два ограничения определяются «постфиксно».

Третье представлено отдельным элементом в списке элементов определения таблицы (огр. табл.).

Определения столбцов и определения ограничений можно переставлять в произвольном порядке.

Первые два ограничения — это ограничения столбцов.

Третье ограничение — ограничение таблицы, поскольку записано отдельно от определений столбцов.

Ограничения столбцов также можно записать в виде ограничений таблицы, тогда как обратное не всегда возможно, так как подразумевается, что ограничение столбца ссылается только на связанный столбец. (Хотя PostgreSQL этого не требует, но для совместимости с другими СУБД лучше следовать это правилу.) Ранее приведённый пример можно переписать и так:

```
CREATE TABLE products (
    product_no integer,
    name             varchar(120),
    price            numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Или так:

```
CREATE TABLE products (
    product_no       integer,
    name             varchar,
    price            numeric CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0 AND price > discounted_price)
);
```

Ограничениям таблицы можно присваивать имена так же, как и ограничениям столбцов:

```
CREATE TABLE products (
    product_no       integer,
    name             varchar(120),
    price            numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CONSTRAINT valid_discount CHECK (price > discounted_price)
);
```

Ограничение-проверка удовлетворяется, если выражение принимает значение **TRUE** или **NULL**.

Результатом многих выражений с операндами **NULL** будет значение **NULL**, поэтому такие ограничения не будут препятствовать записи **NULL** в связанные столбцы.

Чтобы гарантировать, что столбец не содержит значения **NULL**, следует использовать ограничение **NOT NULL**.

Важное замечание

В Postgres предполагается, что условия ограничений **CHECK** являются постоянными, то есть при одинаковых данных в строке они всегда будут выдавать одинаковый результат. Поэтому postgres поддерживает ограничения **CHECK** только для тех данных, которые относятся к только создаваемой или изменяемой строке.

CHECK, нарушающий это правило, может работать в некоторых простых случаях, однако, в общем случае может случиться, что база данных придёт в состояние, когда условие ограничения окажется ложным. Например, если изменились другие строки, участвующие в его вычислении.

В результате восстановление выгруженных данных может оказаться невозможным.

Во время восстановления возможен сбой, даже если полное состояние базы данных согласуется с условием ограничения, по причине того, что строки загружаются не в том порядке, в котором это условие будет соблюдаться.

Поэтому для определения ограничений, затрагивающих другие строки и другие таблицы, следует использовать ограничения **UNIQUE**, **EXCLUDE** или **FOREIGN KEY**, либо реализовать проверки в триггере. Триггеры из дампа восстанавливаются после восстановления данных, поэтому вышеупомянутого сбоя не будет.

Пользовательские функции в ограничениях

Может случиться так, что в выражении CHECK используется пользовательская функция, поведение которой впоследствии будет изменено. Postgres не контролирует этого, и если строки в таблице перестанут удовлетворять ограничению CHECK, это останется незамеченным. В итоге при попытке загрузить выгруженные позже данные могут возникнуть проблемы. Поэтому подобные изменения рекомендуется осуществлять следующим образом:

1) удалить ограничение (используя ALTER TABLE);
2) изменить определение функции;
3) пересоздать ограничение той же командой, которая при этом перепроверит все строки таблицы.

Ограничения NOT NULL

Ограничение **NOT NULL** просто указывает, что столбцу нельзя присваивать значение NULL:

```
CREATE TABLE products (
    product_no integer      NOT NULL,
    name       varchar(120) NOT NULL,
    price      numeric
);
```

Ограничение **NOT NULL** всегда записывается как ограничение столбца и функционально эквивалентно ограничению:

CHECK (*column_name* IS NOT NULL)

Однако, если check-ограничению можно назначить имя, то ограничению **NOT NULL** имя назначить нельзя.

Для столбца можно определить больше одного ограничения, для чего их нужно указать одно за другим:

```
CREATE TABLE products (
    product_no integer      NOT NULL,
    name       varchar(120) NOT NULL,
    price      numeric      NOT NULL CHECK (price > 0)
);
```

Порядок указания ограничений не имеет значения — **порядок проверки ограничений не зависит от порядка из указания.**

Для ограничения **NOT NULL** есть и обратное ограничение — **NULL**.

Суть его состоит в указании, что столбец может иметь значение **NULL** (поведение по умолчанию).

Ограничение **NULL** отсутствует в стандарте SQL и было добавлено в Postgres только для совместимости с некоторыми другими СУБД.

Его можно использовать для быстрого переключения ограничения **NULL**/**NOT NULL** в скрипте.

```
CREATE TABLE products (
    product_no integer      NULL,
    name       varchar(120) NULL,
    price      numeric      NULL
);
```

При необходимости в любое из них можно вставить ключевое слово NOT, если потребуется.

Тем не менее, при проектировании баз данных чаще всего большинство столбцов должны быть помечены как **NOT NULL**.

Ограничения уникальности

Ограничения уникальности гарантируют, что данные в определённом столбце или группе столбцов уникальны среди всех строк таблицы.

Ограничение в виде ограничения столбца:

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name       varchar(120),
    price      numeric
);
```

Ограничение в виде ограничения таблицы:

```
CREATE TABLE products (
    product_no integer,
    name       varchar(120),
    price      numeric,
    UNIQUE (product_no)
);
```

Чтобы определить ограничение уникальности для группы столбцов, его следует записать в виде ограничения таблицы, при этом имена столбцов перечисляются через запятую:

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    UNIQUE (a, c)
);
```

Ограничение уникальности для группы столбцов указывает, что сочетание значений перечисленных столбцов должно быть уникально во всей таблице, тогда как значения каждого столбца по отдельности уникальными не должны быть (и обычно не будут).

Уникальному ограничению можно назначить имя:

```
CREATE TABLE products (
    product_no integer CONSTRAINT must_be_unique UNIQUE,
    name       varchar(120),
    price      numeric
);
```

Ограничение уникальности нарушается, если в таблице оказывается несколько строк, у которых совпадают значения всех столбцов, включённых в ограничение.

При этом **два значения NULL при сравнении никогда не считаются равными.**

Из этого следует, что даже при наличии ограничения уникальности в таблице можно сохранить строки с дублирующимися значениями, если те содержат **NULL** в одном или нескольких столбцах, участвующих в ограничении.

Это поведение соответствует стандарту SQL, но существуют СУБД, которые ведут себя иначе.

Первичные ключи

Ограничение первичного ключа означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. Для этого требуется, чтобы значения были одновременно уникальными и отличными от **NULL**. Два следующих определения почти эквивалентны:

```
CREATE TABLE products (
    product_no integer UNIQUE NOT NULL,
    name       varchar(120),
    price      numeric
);
```

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       varchar(120),
    price      numeric
);
```

Первичные ключи могут включать несколько столбцов.
Синтаксис похож на запись ограничений уникальности:

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

При добавлении первичного ключа автоматически создаётся уникальный индекс — B-дерево для столбца или группы столбцов, перечисленных в первичном ключе, и данные столбцы помечаются как **NOT NULL**.

Таблица может иметь максимум один первичный ключ.

```
ОШИБКА:  таблица "ex2" не может иметь несколько первичных ключей
СТРОКА 3:   b integer PRIMARY KEY
```

Ограничений уникальности и ограничений **NOT NULL**, которые функционально почти равнозначны первичным ключам, может быть сколько угодно, но назначить ограничением первичного ключа можно только одно.

Теория реляционных баз данных говорит, что первичный ключ должен быть в каждой таблице и ему стоит следовать, даже если это требование не является обязательным в данной реализации.

Внешние ключи

Ограничение внешнего ключа указывает, что значения столбца (или группы столбцов) должны соответствовать значениям в некоторой строке другой таблицы.

Это называется ***ссылочной целостностью*** двух связанных таблиц.

Например, имеем таблицу:

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       varchar(120),
    price      numeric
);
```

Пусть существует таблица заказов этих продуктов. Нужно, чтобы в таблице заказов содержались только заказы действительно существующих продуктов. Поэтому в ней определяется ограничение внешнего ключа, ссылающееся на таблицу продуктов:

```
CREATE TABLE orders (
    id         integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity   integer
);
```

С таким ограничением создать заказ со значением **product_no**, отсутствующим в таблице **products** (и не равным **NULL**), будет невозможно.

В такой схеме таблицу **orders** называют *подчинённой* таблицей, а **products** — *главной*.

Соответственно, столбцы называют так же *подчинённым* и *главным* (или *ссылающимся* и *целевым*).

Подчиненную таблицу можно записать короче:

```
CREATE TABLE orders (
    id         integer PRIMARY KEY,
    product_no integer REFERENCES products, -- ссылается на products PRIMATRY_KEY
    quantity   integer
);
```

Если список столбцов опущен, внешний ключ будет связан с первичным ключом главной таблицы *неявно*.

Ограничению внешнего ключа можно назначить имя.

Внешний ключ может ссылаться на группу столбцов. В этом случае его нужно записать в виде обычного ограничения таблицы:

```
CREATE TABLE t1 (
    a integer PRIMARY KEY,
    b integer,
    c integer,
    FOREIGN KEY (b, c) REFERENCES master_table (c1, c2)
);
```

Число и типы столбцов в ограничении должны соответствовать числу и типам целевых столбцов.

Дерево

Используя ограничение внешнего ключа, ссылающегося на свою же таблицу, можно организовать иерархическую структуру связей (в направлении к корню):

```
CREATE TABLE tree (
    node_id   integer PRIMARY KEY,
    parent_id integer REFERENCES tree,
    name      varchar(120),
    ...
);
```

Для узла верхнего уровня **parent_id** будет равен **NULL**, а записи со значением **parent_id**, отличным от **NULL**, будут ссылаться только на существующие строки таблицы.

Таблица может содержать несколько ограничений внешнего ключа — это используется для связи таблиц в отношении многие-ко-многим. В частности, это можно использовать для организации дерева и даже произвольного графа.

Положим, есть таблицы продуктов и заказов, но нужно, чтобы один заказ мог содержать несколько продуктов (что невозможно в схеме LUT). Это выглядит так:

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       varchar(120),
    price      numeric
);

CREATE TABLE orders (
    id               integer PRIMARY KEY,
    shipping_address text,
    ...
);

CREATE TABLE order_items (
    product_no integer REFERENCES products,
    order_id   integer REFERENCES orders,
    quantity   integer,
    PRIMARY KEY (product_no, order_id)
);
```

В последней таблице первичный ключ покрывает внешние ключи.

Удаление записей, на которые ссылаются вторичные (внешние) ключи

Внешние ключи запрещают создание заказов, не относящихся ни к одному продукту.

Но что делать, если после создания заказов с определённым продуктом мы захотим удалить его?

Варианты поведения — запретить удаление продукта, удалить продукт и связанные с ним заказы, или что-то более умное, например:

- при попытке удаления продукта, на который ссылаются заказы (через таблицу **order_items**), операция запрещается;
- при попытке удалить заказ, он удаляется вместе со всем его содержимым (каскадное удаление).

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name       varchar(120),
    price      numeric
);

CREATE TABLE orders (
    id               integer PRIMARY KEY,
    shipping_address text,
    ...
);

CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id   integer REFERENCES orders ON DELETE CASCADE,
    quantity   integer,
    PRIMARY KEY (product_no, order_id)
);
```

Ограничивающие и каскадные удаления — два наиболее распространённых варианта.

RESTRICT предотвращает удаление связанной строки.

NO ACTION (поведение по умолчанию) означает, что если зависимые строки продолжают существовать при проверке ограничения, возникает ошибка .

Главным отличием этих двух вариантов друг от друга является то, что **NO ACTION** позволяет отложить проверку в процессе транзакции, а **RESTRICT** — нет.

CASCADE указывает, что при удалении связанных строк зависимые от них будут так же автоматически удалены.

SET NULL и **SET DEFAULT** — при удалении связанных строк они назначают зависимым столбцам в подчинённой таблице значения **NULL** или значения по умолчанию, соответственно.

Следует заметить, что это не будет основанием для нарушения ограничений — если, например, в качестве действия задано **SET DEFAULT**, но значение по умолчанию не удовлетворяет ограничению внешнего ключа, операция закончится ошибкой.

Аналогично указанию **ON DELETE** существует **ON UPDATE**, которое срабатывает при изменении заданного столбца.

При этом возможные действия те же, а **CASCADE** в данном случае означает, что изменённые значения связанных столбцов будут просто скопированы в зависимые строки.

Внешний ключ должен ссылаться на столбцы, образующие первичный ключ или ограничение уникальности. Соответственно, для связанных столбцов всегда будет существовать индекс (определённый соответствующим первичным ключом или ограничением. Это приведет к тому, что проверки соответствия связанной строки будут выполняться эффективно.

Поскольку команды **DELETE** для строк главной таблицы или **UPDATE** для зависимых столбцов требуют просканировать подчинённую таблицу и найти строки, ссылающиеся на старые значения, полезно иметь индекс и для подчинённых столбцов (индекс для первичного ключа строится неявно).

Поскольку это нужно не всегда, и создавать соответствующий индекс можно по-разному, объявление внешнего ключа не создаёт автоматически индекс по связанным столбцам.

Изменение таблиц (ALTER)

Созданную и заполненную данными таблицу можно модифицировать в отношении структуры и/или ограничений. Postgres для этой цели предоставляет набор команд модификации таблиц.

Можно:

- добавлять/удалять столбцы;
- добавлять/удалять ограничения;
- изменять значения по умолчанию;
- изменять типы столбцов;
- переименовывать столбцы;
- переименовывать таблицы.

Добавление/удаление столбца

ALTER TABLE *table_name* ADD COLUMN *column_name type* [DEFAULT *value*];

Новый столбец заполняется заданным для него значением по умолчанию (**DEFAULT**) или значением **NULL**, если ничего не будет указано в качестве **DEFAULT**.

```
ALTER TABLE products ADD COLUMN description text;
```

Можно сразу определить ограничения столбца, используя обычный синтаксис:

```
ALTER TABLE products ADD COLUMN description text CHECK (description <> '');
```

**Можно использовать любые конструкции,
допустимые в определении столбца в команде CREATE TABLE.**

Значение по умолчанию, если есть, должно удовлетворять данным ограничениям. Поэтому сначала можно заполнить столбец и только потом добавить ограничение.

Удаление столбца

ALTER TABLE *table_name* DROP COLUMN *column_name*;

Данные, которые были в удаляемом столбце, исчезают.

Вместе со столбцом удаляются и включающие его ограничения таблицы.

Если на столбец ссылается ограничение внешнего ключа другой таблицы, PostgreSQL не удалит это ограничение, если не предпринять дополнительных мер.

Разрешить удаление всех зависящих от этого столбца объектов можно, добавив указание **CASCADE**:

```
ALTER TABLE products DROP COLUMN description CASCADE;
```

Добавление/удалеение ограничения

Для добавления ограничения используется синтаксис ограничения таблицы:

```
ALTER TABLE products ADD CHECK (name <> '');
ALTER TABLE products ADD CONSTRAINT some_name UNIQUE (product_no);
ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES product_groups;
```

Ограничение **NOT NULL** нельзя записать в виде ограничения таблицы, поэтому делаем так:

```
ALTER TABLE products ALTER COLUMN product_no SET NOT NULL;
```

Ограничение проходит проверку автоматически и будет добавлено, только если ему удовлетворяют данные таблицы.

Удаление ограничения

Чтобы удалить ограничение, нужно знать его имя.

Если имя не было явно присвоено пользователем, это неявно сделала система. Поэтому нужно сначала выяснить это имя, используя, например, команду psql:

```
 \d[S+]  name -- описание таблицы, представления, последовательности или индекса.
```

Узнав имя, можно удалить ограничение:

```
ALTER TABLE products DROP CONSTRAINT constraint_name;
```

У ограничений **NOT NULL** нет имён, поэтому для его удаления следует делать так:

```
ALTER TABLE products ALTER COLUMN product_no DROP NOT NULL;
```

Изменение значения по умолчанию

Новое значение по умолчанию:

```
ALTER TABLE products ALTER COLUMN price SET DEFAULT 7.77;
```

Это не влияет на существующие строки таблицы, а задает значение по умолчанию для последующих команд **INSERT**.

Удаление значения по умолчанию:

```
ALTER TABLE products ALTER COLUMN price DROP DEFAULT;
```

При этом по сути значению по умолчанию просто присваивается NULL. Как следствие, ошибки не будет, если вы попытаетесь удалить значение по умолчанию, не определённое явно, так как неявно оно существует и равно NULL.

Изменение типа данных столбца

`ALTER TABLE` *`table_name`* `ALTER COLUMN` *`column_name`* `TYPE` *`new_type`*

Пример:

```
ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);
```

Операция будет успешна, только если все существующие значения в столбце могут быть неявно приведены к новому типу.

Если требуется более сложное преобразование, необходимо добавить указание **`USING`**, определяющее, как получить новые значения из старых.

Важно:

Postgres попытается преобразовать к новому типу значение столбца по умолчанию и все связанные с этим столбцом ограничения. Преобразование может оказаться некорректным, и в результате случится нехорошее. Поэтому,

перед тем как менять тип столбца, следует удалить все его ограничения, а потом заново установить ограничения, модифицированные в соответствии с новым типом.

Переименование столбца

ALTER TABLE *table_name* RENAME COLUMN *column_name* TO *new_column_name*;

```
ALTER TABLE products RENAME COLUMN product_no TO product_number;
```

Переименование таблицы

ALTER TABLE *table_name* RENAME TO *new_table_name*;

```
ALTER TABLE products RENAME TO items;
```