DDL – это управление таблицами, данными и тд. Типа добавить столб, убавить столб. Тут даны первые команды, потом дополним еще другими.

Команды:

1. CREATE TABLE table­­\_name – создание таблицы
2. ALTER TABLE table­­\_name – изменение таблицы

ADD COLUMN column\_name data type – добавить колонку

RENAME TO new\_table\_name – новое имя **таблицы**

RENAME COLUMN old\_column\_name TO new\_column\_name – новое имя **столбца**

ALTER COLUMN column\_name SET DATA TYPE data\_type – изменить тип данных **столбца**

1. DROP TABLE table\_name – удаление таблицы
2. TRUNCATE TABLE table\_name – удаляет всё наглухо. Даже логи не сохраняются, но она не может удалить данные, на которые есть ссылки из других таблиц по внешнему ключу и т.п., при этом сама таблица не удаляется, только данные
3. DROP COLUMN column\_name – удаление колонки

**Создание таблицы**

create table student

(

student\_id serial,

first\_name varchar,

last\_name varchar,

birthday date,

phone varchar

);

create table cathedra

(

cathedra\_id serial,

cathedra\_name varchar,

dean varchar

);

alter table student

add column middle\_name varchar;

ALTER TABLE student

ADD COLUMN rating float;

ALTER TABLE student

ADD COLUMN enrolled date;

ALTER TABLE student

DROP COLUMN middle\_name;

ALTER TABLE student

RENAME TO students;

ALTER TABLE students

RENAME phone TO phone\_number;

ALTER TABLE cathedra

RENAME TO chair;

ALTER TABLE chair

RENAME cathedra\_id TO chair\_id;

ALTER TABLE chair

RENAME cathedra\_name TO chair\_name;

ALTER TABLE students

ALTER COLUMN first\_name SET DATA TYPE varchar(30);

ALTER TABLE students

ALTER COLUMN last\_name SET DATA TYPE varchar(64);

create table faculty

(

faculty\_id serial primary key,

faculty\_name varchar(64)

);

insert into faculty (faculty\_name)

values

('biological'),

('chemistry'),

('physics');

select \*

from faculty;

TRUNCATE faculty RESTART IDENTITY;

DROP TABLE faculty

Что стоит иметь в виду, все эти функции доступны и через интерфейс приложения. Однако, если такие таблицы надо создавать по многу раз, или как-то автоматизировать, то тут как раз и подойдет эта тема для написания скриптов и т.д.

**Primary key**

В колонке первичного ключа не может быть одно и то же значение для нескольких строк. Кроме того нулевого значения там также не может быть. У primary key есть аналог – unique not null. Именно так выражение работает на 99% аналогично первичному ключу, но есть убрать not null, то unique еще может быть и пустым. Но у юник вроде нет автоувеличения.

Пару слов про ограничения. Если мы его сами не именуем, то оно именуется само по «название таблицы\_название колонки\_ключ». Проименовать мы можем и сами через такое в момент создания таблицы:

create table chair

(

chair \_id serial,

chair \_name varchar,

dean varchar,

CONSTRAINT PK\_chair\_chair\_id\_key PRIMARY KEY(chair\_id)

);

Для удаления ограничения используем такое:

Alter table chair

Drop constraint chair\_chair\_id\_key

А для добавления:

Alter table chair

Add primary key(chair\_id)

**FOREING KEY**

Форейн ки нужен для того, чтобы одна таблица могла правильно ссылаться на другую, если его не добавить, то данные одной таблицы могут ссылаться на несуществующие данные в другой таблице. Внешний ключ можно добавить как сразу, так и уже после создания таблицы, запись производится так:

**add constraint** FK\_book\_publisher **foreign key**(publisher\_id) **references** publisher(publisher\_id)

буквально:

добавить ограничение -название ограничения- указание, что это внешний ключ(его имя) ссылается на таблицу паблишер(паблишер\_ид)

в таком случае при попытке указания внешнего ключа, который отсутствует в другой таблице, будет выскакивать ошибка.

**CHECK**

Кроме ограничения по ключу есть еще и логические ограничения. Ниже будет приведен пример, как создать такое ограничение.

alter table book

add column price decimal constraint CHK\_book\_price CHECK (price >=0);

здесь мы изменяем таблицу книга

затем добавляем колонку с названием цена, тип децимал и добавляем ограничение, указываем имя, спецслово и что условие – цена больше или равна нулю. Имя ограничения по соглашению начинается с CHK, затем идет имя таблицы, затем название столбика. Если потом попытаться добавить цену меньше нуля, то будет выдаваться ошибка.

ERROR: ОШИБКА: новая строка в отношении "book" нарушает ограничение-проверку "chk\_book\_price" DETAIL: Ошибочная строка содержит (1, title1, 123, -1).

**DEFAULT**

Следующее, но не совсем ограничение, - default. Оно отвечает за необязательные при заполнении поля. Если при вводе данных такое поле не заполняется, туда по дефолту устанавливается определенное значение, которое мы прописали.

create table customer

(

customer\_id serial, -- для сериал мы не проставляем нумерацию

full\_name text,

status char default 'r',

constraint PK\_customer\_customer\_id primary key (customer\_id),

constraint chk\_customer\_status check (status = 'r' or status = 'v')

);

Для удаления дефолта делаем так

Alter table \_\_\_\_\_

Alter column \_\_\_\_\_ drop default

Если хотим добавить

Alter table \_\_\_\_\_

Alter column \_\_\_\_\_ set default ‘tt’

**Последовательности**

Есть такой тип данных как serial. Он автоинкрементируемый, то есть увеличивается на единичку, значения генерируются автоматически. Последовательности можно создавать и ими можно управлять.

Для создания последовательности мы используем фразу CREATE SEQUENCE seq1, не забывая указать имя.

Для того, чтобы сгенерировать следующее значение в последовательности, мы используем **SELECT nextvat(‘seq1’),** передавая имя последовательности. По умолчанию генерируется 1, и на 1 и увеличивается.

**SELECT currval('seq1') –** возвращает текущее значение.

**SELECT lastval();**  - возвращает последнее значение в текущей сессии, последовательность никакую не принимает, целиком по сессии.

Если мы хотим назначить число в последовательность, у нас есть два варианта. Первый принимает в себя название последовательности, значение последовательности и ТРУ. Тру означает, что в момент вызова кода значение последовательности сразу же будет применено, и сразу станет таким, какое мы передали.

**select setval('seq1', 16, true);**

**SELECT currval('seq1');**

**SELECT nextval('seq1');**

Если мы проставляем ФОЛС, то система запоминает, что мы хотим проставить туда что-то определенное, но при этом запишет переданное значение только после того, как мы вызовем nextval.

**select setval('seq1', 16, false);**

**SELECT currval('seq1');**

**SELECT nextval('seq1');**

Мы можем указывать, на какое значение должна инкрементироваться последовательность, а также применять конструкцию if not exists. При этом первое значение всегда будет единичкой, если мы явно не указываем обратное.

**create sequence if not exists seq2 increment 23;**

**select nextval('seq2');**

Что касается некоторых других условий.

create sequence if not exists seq3

increment 8

minvalue 0

maxvalue 128

start with 0;

select nextval('seq3');

В данном случае отсчет начинается с 0, при повторных вызовах следующего значения будет происходить увеличение на 8, так до 128. При достижении 128 и попытке еще одного вызова будет появляться ошибка.

ERROR: ОШИБКА: функция nextval достигла максимума для последовательности "seq3" (128)

Есть возможность переименования последовательности. Работает как и с таблицами.

Alter sequence seq3 rename to seq4.

И рестарт последовательности.

Alter sequence seq4 restart with 16

Удаление происходит через DROP

DROP SEQUENCE seq4

**Далее**

create table book

(

book\_id int not null,

title text not null,

isbn varchar(32) not null,

publisher\_id int not null,

constraint PK\_book\_book\_id primary key(book\_id)

);

При создании таблиц мы часто указываем айди для строк. Если мы для айди выбираем тип данных не **serial**, а **int,** то в этом случае автоинкрементации не происходит. В данном случае мы можем назначить последовательность на это поле.

Для создания такой последовательности мы используем такой синтаксис:

Create sequence if not exists **book\_book\_id\_seq**

Start with 1 owned by book.book\_id

Что буквально означает:

«создать последовательность, если уже не существует, имя (обычно это название таблицы и столбца)

Начинается с 1 и принадлежит название таблицы-название столбца.

Однако, исполнение этого кода не говорит о том, что мы можем избегать явного указания book\_id при занесении данных. Если же бы будем вводить все данные, кроме book\_id, мы получим ошибку, что значение book\_id не может быть пустым. Для того, чтобы автоинкрементация работала при неуказании значения для данного поля, мы должны прописать следующее:

**Alter table book**

**Alter column book\_id set default nextval(‘book\_book\_id\_seq’)**

Что говорит нам о необходимости изменить таблицу – изменить колонку book\_id, и назначить в нее функцию nextval для нашей последовательности, в этом случае все будет работать.

**insert into book(title, isbn, publisher\_id)**

**values ('title2', '2123', 2);**

**select \* from book**

**Проблемы serial.**У сериал есть проблемы в том плане, если мы ставим автоинкрементирование, а потом вручную добавляем еще одну и более строк данных в таблицу, этот сериал не увидит в реальном времени, что у него там появились эти записи. Он выдаст ошибку, что число уже занято.

Появилась новая телега, которая помогает этого избежать.

create table book

(

book\_id **int GENERATED ALWAYS AS IDENTITY (start with 10 increment by 2) not null**,

title text not null,

isbn varchar(32) not null,

publisher\_id int not null,

constraint PK\_book\_book\_id primary key(book\_id)

);

Эта надпись помогает избежать проблем, вместо ALWAYS еще есть BY DEFAULT, но он уже не будет выдавать ошибку при попытке залезть в автоинкрементацию. После IDENTITY идет инструкция, с чего начать и на сколько увеличивать

Если вставлять данные так

**insert into book(title, isbn, publisher\_id)**

**values ('title2', '2123', 2);**

то все будет работать,

если попробовать вот так

**insert into book**

**values (1, 'title2', '2123', 2);**

то будет ошибка, потому что постгрес запрещает нам вставлять данные явно

ERROR: ОШИБКА: вставить данные в столбец "book\_id" нельзя DETAIL: Столбец "book\_id" является столбцом идентификации со свойством GENERATED ALWAYS. HINT: Для переопределения укажите OVERRIDING SYSTEM VALUE. OVERRIDING SYSTEM VALUE пишется следующей строкой после **insert into**

**INSERT**

Вставка данных в таблицы. Если мы хотим вставить данные в конкретные столбцы, то используем вот такой синтаксис. При условии, что столбцы, которые мы пропускаем, не имеют ограничения NOT NULL

**Insert into \_\_\_\_ \_id, rating, smth\_else**

**values (‘11’, 11, ‘121’)**

Если же мы хотим вставить данные во все столбцы, то используем

**Insert into \_\_\_\_**

**values (‘11’, 11, ‘121’, «2222», «3333»)**

и кое что совсем новое – можно выбирать данные из одной таблицы, создавая другую, и пихая в новую те данные, которые были выбраны. У меня была таблица с экзаменами, название и дата, этот синтаксис позволил выбрать часть экзаменов и поместить их в другую таблицу.

**select \***

**into late\_exams**

**from exam**

**where exam\_date >= '2021-12-27';**

Однако дополнить таким образом данную таблицу уже невозможно. Для добавления данных надо использовать следующее:

**insert into late\_exams**

**select \***

**from exam**

**where exam\_date < '2021-12-27';**

**UPDATE**

Для того, чтобы изменить данные в конкретной строке конкретного столбца конкретной таблицы, надо использовать команду UPDATE,

**update author**

**set full\_name = 'Alex', rating = '4.61'**

**where author\_id = 1**

нужно отметить, что после обновления, строка переедет в самый низ, поэтому надо будет при выборке указывать соритровку

**DELETE**

удаление данных по условию

**delete from author**

**where rating < 4.3**

чтобы удалить пустые строчки, нужно ставить не = NULL, а IS NULL

**delete from author**

**where rating is null**

если мы хотим удалить данные по всей таблице, пишем просто **delete from author,** она работает как и TRUNCATE, но единственное что она пишет логи и работает медленнее транкейта

**RETURNING**

иногда нам нужно вернуть данные по модифицированной строке. Например, ввести данные и узнать, под каким айди были введены данные

insert into book (title, isbn, publisher\_id)

**values**

**('title2', '909', 7)**

**returning book\_id**

постгрес вернет book\_id, а вот если написать **returning \*,** то постгрес вернет все, что было вставлено.

**update author**

**set full\_name = 'Gandalf', rating = 5**

**where author\_id = 2**

**returning \***

домашка

1

create table exam

(

exam\_id serial UNIQUE NOT NULL,

exam\_name text,

exam\_date date

);

select \* from exam;

2

alter table exam

drop constraint exam\_exam\_id\_key;

3

alter table exam

add primary key(exam\_id);

4

create table person

(

person\_id int primary key,

first\_name varchar(64),

surname varchar(64)

);

5

create table passport

(

passport\_id int primary key,

serial\_number int not null,

reg\_address text,

person\_id int,

CONSTRAINT FK\_person\_passport foreign key(person\_id) references person(person\_id)

);

6

alter table book

add column freight decimal constraint CHK\_book\_freight CHECK (freight > 0 and freight < 100)

7

insert into book(title, isbn, publisher\_id, freight)

values

('title111', '121212', 1, 150)

8

create table student

(

student\_id serial,

full\_name text,

course int default 1

)

9

insert into student(full\_name)

values

('Kesha');

select \* from student

10

alter table student

alter column course drop default

11

12 и 13

create sequence products\_product\_id\_seq

start with 78 owned by products.product\_id;

alter table products

alter column product\_id set default nextval('products\_product\_id\_seq')

insert into products(product\_name, discontinued)

values ('fishman', 1)