ДЗ 4. Глава 5 (упражнения 2, 9, 17, 18).

2. Посмотрите, какие ограничения уже наложены на атрибуты таблицы «Успеваемость» (`progress`). Воспользуйтесь командой `\d` утилиты psql. А теперь предложите для этой таблицы ограничение уровня таблицы.

   В качестве примера рассмотрим такой вариант. Добавьте в таблицу `progress` еще один атрибут — «Форма проверки знаний» (`test_form`), который может принимать только два значения: «экзамен» или «зачет». Тогда набор допустимых значений атрибута «Оценка» (`mark`) будет зависеть от того, экзамен или зачет предусмотрены по данной дисциплине. Если предусмотрен экзамен, тогда допускаются значения 3, 4, 5, если зачет — тогда 0 (не зачтено) или 1 (зачтено).

Не забудьте, что значения NULL для атрибутов `test_form` и `mark` не допускаются.

Новое ограничение может быть таким:

```
ALTER TABLE progress
  ADD CHECK (
    ( test_form = 'экзамен' AND mark IN ( 3, 4, 5 ) )
    OR
    ( test_form = 'зачет' AND mark IN ( 0, 1 ) )
  );
```

Проверьте, как будет работать новое ограничение в модифицированной таблице `progress`. Для этого выполните команды `INSERT`, как удовлетворяющие ограничению, так и нарушающие его.

В таблице уже было ограничение на допустимые значения атрибута `mark`. Как вы думаете, не будет ли оно конфликтовать с новым ограничением? Проверьте эту гипотезу. Если ограничения конфликтуют, тогда удалите старое ограничение и снова попробуйте добавить строки в таблицу.

Подумайте, какое еще ограничение уровня таблицы можно предложить для этой таблицы?

```
edu=# \d progress
                  Table "public.progress"
   Column    |     Type     | Collation | Nullable | Default
-------------+--------------+-----------+----------+---------
 record_book | numeric(5,0) |           | not null |
 subject     | text         |           | not null |
 acad_year   | text         |           | not null |
 term        | numeric(1,0) |           | not null |
 mark        | numeric(1,0) |           | not null | 5
 test_form   | text         |           | not null |
Check constraints:
    "progress_mark_check" CHECK (mark >= 3::numeric AND mark <= 5::numeric)
    "progress_term_check" CHECK (term = 1::numeric OR term = 2::numeric)
    "progress_test_form_check" CHECK (test_form = 'экз'::text OR test_form = 'зач'::text)
Foreign-key constraints:
    "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES students(record_book) ON UPDATE CASCADE ON DELETE C
ASCADE
```

Добавляем студента :

```
edu=# INSERT INTO students VALUES (12345, 'A.A.NIKITIN', 1234, 123456)
;
INSERT 0 1
edu=# SELECT * FROM students;
 record_book |     name      | doc_ser | doc_num
-------------+---------------+---------+---------
       12345 | A.A.NIKITIN |    1234 |  123456
(1 row)
```

Вводим новое ограничение:

```
edu=# \d progress
                Table "public.progress"
   Column    |     Type     | Collation | Nullable | Default
-------------+--------------+-----------+----------+---------
 record_book | numeric(5,0) |           | not null |
 subject     | text         |           | not null |
 acad_year   | text         |           | not null |
 term        | numeric(1,0) |           | not null |
 mark        | numeric(1,0) |           | not null | 5
 test_form   | text         |           | not null |
Check constraints:
    "progress_check" CHECK (test_form = 'экзамен'::text AND (mark = ANY (ARRAY[3::numeric, 4::numeric, 5::numeric])) OR
test_form = 'зачет'::text AND (mark = ANY (ARRAY[0::numeric, 1::numeric])))
    "progress_term_check" CHECK (term = 1::numeric OR term = 2::numeric)
    "progress_test_form_check" CHECK (test_form = 'экзамен'::text OR test_form = 'зачет'::text)
Foreign-key constraints:
    "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES students(record_book) ON UPDATE CASCADE ON DELETE C
ASCADE
```

```
edu=# INSERT INTO progress VALUES (12345, 'postgresql', 'first', 1, 1, 'зачет');
INSERT 0 1
edu=# SELECT * FROM progress;
 record_book |  subject   | acad_year | term | mark | test_form
-------------+------------+-----------+------+------+-----------
       12345 | postgresql | first     |    1 |    1 | зачет
(1 row)
```

```
edu=# INSERT INTO progress VALUES (12345, 'postgresql', 'first', 1, 5, 'зачет');
ERROR:  new row for relation "progress" violates check constraint "progress_check"
DETAIL:  Failing row contains (12345, postgresql, first, 1, 5, зачет).
edu=#
```

```
edu=# INSERT INTO progress VALUES (12345, 'postgresql', 'first', 1, 5, 'экзамен');
INSERT 0 1
edu=# SELECT * FROM progress;
 record_book |  subject   | acad_year | term | mark | test_form
-------------+------------+-----------+------+------+-----------
       12345 | postgresql | first     |    1 |    1 | зачет
       12345 | postgresql | first     |    1 |    5 | экзамен
(2 rows)
```

```
edu=# INSERT INTO progress VALUES (12345, 'postgresql', 'first', 1, 1, 'экзамен');
ERROR:  new row for relation "progress" violates check constraint "progress_check"
DETAIL:  Failing row contains (12345, postgresql, first, 1, 1, экзамен).
```

Изначальное ограничение на отметки необходимо было удалить, потому что оценки на зачет идут 1 или 0, что не позволительно с предыдущими условиями.

В дальнейшем необходимо будет рассмотреть ограничение на предметы, потому что нельзя получить несколько отметок за зачет или экзамен.

Также нельзя допускать, чтобы по одному предмету был экзамен и зачет одновременно.

9. В таблице «Студенты» (`students`) есть текстовый атрибут `name`, на который наложено ограничение `NOT NULL`. Как вы думаете, что будет, если при вводе новой строки в эту таблицу дать атрибуту `name` в качестве значения пустую строку? Например:

```
INSERT INTO students ( record_book, name, doc_ser, doc_num )
  VALUES ( 12300, '', 0402, 543281 );
```

Наверное, проектируя эту таблицу, мы хотели бы все же, чтобы пустые строки в качестве значения атрибута `name` не проходили в базу данных? Какое решение вы можете предложить? Видимо, нужно добавить ограничение `CHECK` для столбца `name`. Если вы еще не изучили команду `ALTER TABLE`, то удалите таблицу `students` и создайте ее заново с учетом нового ограничения, а если вы уже познакомились с командой `ALTER TABLE`, то сделайте так:

```
ALTER TABLE students ADD CHECK ( name <> '' );
```

Добавив ограничение, попробуйте теперь вставить в таблицу `students` строку (row), в которой значение атрибута `name` было бы пустой строкой (string).

Давайте продолжим эксперименты и предложим в качестве значения атрибута `name` строку, содержащую сначала один пробел, а потом — два пробела.

```
INSERT INTO students VALUES ( 12346, ' ', 0406, 112233 );
INSERT INTO students VALUES ( 12347, '  ', 0407, 112234 );
```

Для того чтобы «увидеть» эти пробелы в выборке, сделаем так:

```
SELECT *, length( name ) FROM students;
```

Оказывается, эти невидимые значения имеют ненулевую длину. Что делать, чтобы не допустить таких значений-невидимок? Один из способов: возложить проверку таких ситуаций на прикладную программу. А что можно сделать на уровне определения таблицы `students`? Какое ограничение нужно предложить? В разделе 9.4 документации «Строковые функции и операторы» есть функция `trim`. Попробуйте воспользоваться ею. Если вы еще не изучили команду `ALTER TABLE`, то удалите таблицу `students` и создайте ее заново с учетом нового ограничения, а если уже познакомились с ней, то сделайте так:

```
ALTER TABLE students ADD CHECK (...);
```

Есть ли подобные слабые места в таблице «Успеваемость» (`progress`)?

```
edu=# ALTER TABLE students ADD CHECK (TRIM(name) <> '');
;
ALTER TABLE
```

```
edu=# \d students
                Table "public.students"
   Column    |     Type     | Collation | Nullable | Default
-------------+--------------+-----------+----------+---------
 record_book | numeric(5,0) |           | not null |
 name        | text         |           | not null |
 doc_ser     | numeric(4,0) |           |          |
 doc_num     | numeric(6,0) |           |          |
Indexes:
    "students_pkey" PRIMARY KEY, btree (record_book)
Check constraints:
    "students_name_check" CHECK (TRIM(BOTH FROM name) <> ''::text)
Referenced by:
    TABLE "progress" CONSTRAINT "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES students(record_book) ON UPDATE CA
SCADE ON DELETE CASCADE
```

```
edu=# ALTER TABLE students ADD CHECK (TRIM(name) <> ''::text);
ALTER TABLE
edu=# INSERT INTO students VALUES ( 12347, '', 0407, 112234 );
ERROR:  new row for relation "students" violates check constraint "students_name_check"
DETAIL:  Failing row contains (12347, , 407, 112234).
edu=# INSERT INTO students VALUES ( 12347, '    ', 0407, 112234 );
ERROR:  new row for relation "students" violates check constraint "students_name_check"
DETAIL:  Failing row contains (12347,     , 407, 112234).
edu=# SELECT * FROM students;
 record_book |    name     | doc_ser | doc_num
-------------+-------------+---------+---------
       12345 | A.A.NIKITIN |    1234 |  123456
(1 row)
```

В таблице progress данный «болячки» появляются в столбцах, где тип данных является text.

Для решения данной проблемы можно использовать такой же подход, как у таблицы students.

17. Представления могут быть, условно говоря, *вертикальными* и *горизонтальными*. При создании вертикального представления в список его столбцов включается лишь часть столбцов базовой таблицы (таблиц). Например:

```
CREATE VIEW airports_names AS
  SELECT airport_code, airport_name, city
  FROM airports;

SELECT * FROM airports_names;
```

В горизонтальное представление включаются не все строки базовой таблицы (таблиц), а производится их отбор с помощью фраз `WHERE` или `HAVING`.

Например:

```
CREATE VIEW siberian_airports AS
  SELECT * FROM airports
  WHERE city = 'Новосибирск' OR city = 'Кемерово';

SELECT * FROM siberian_airports;
```

Конечно, вполне возможен и смешанный вариант, когда ограничивается как список столбцов, так и множество строк при создании представления.

Подумайте, какие представления было бы целесообразно создать для нашей базы данных «Авиаперевозки». Необходимо учесть наличие различных групп пользователей, например: пилоты, диспетчеры, пассажиры, кассиры.

Создайте представления и проверьте их в работе.

```
demo=# CREATE VIEW passenger AS
SELECT passenger_name FROM tickets;
CREATE VIEW
```

```
demo=# SELECT * FROM passenger LIMIT 10;
   passenger_name
---------------------
 VALERIY TIKHONOV
 EVGENIYA ALEKSEEVA
 ARTUR GERASIMOV
 ALINA VOLKOVA
 MAKSIM ZHUKOV
 NIKOLAY EGOROV
 TATYANA KUZNECOVA
 IRINA ANTONOVA
 VALENTINA KUZNECOVA
 POLINA ZHURAVLEVA
(10 rows)
```

```
demo=# CREATE VIEW cities AS SELECT city FROM airports;
CREATE VIEW
demo=# SELECT * FROM cities;
demo=# SELECT * FROM cities LIMIT 10;
    city
------------
 Мирный
 Нижнекамск
 Новокузнецк
 Нальчик
 Владикавказ
 Чебоксары
 Надым
 Нягань
 Курск
 Саранск
(10 rows)
```

18.* Предположим, что нам понадобилось иметь в базе данных сведения о технических характеристиках самолетов, эксплуатируемых в авиакомпании. Пусть это будут такие сведения, как число членов экипажа (пилоты), тип двигателей и их количество.

Следовательно, необходимо добавить новый столбец в таблицу «Самолеты» (`aircrafts`). Дадим ему имя `specifications`, а в качестве типа данных выберем `jsonb`. Если впоследствии потребуется добавить и другие характеристики, то мы сможем это сделать, не модифицируя определение таблицы.

```
ALTER TABLE aircrafts ADD COLUMN specifications jsonb;
```

```
ALTER TABLE
```

Добавим сведения для модели самолета Airbus A320-200:

```
UPDATE aircrafts
  SET specifications =
      '{ "crew": 2,
        "engines": { "type": "IAE V2500",
                     "num": 2
                   }
       }'::jsonb
  WHERE aircraft_code = '320';
```

```
UPDATE 1
```

Посмотрим, что получилось:

```
SELECT model, specifications
  FROM aircrafts
  WHERE aircraft_code = '320';
```

```
      model      |                specifications
-----------------+-----------------------------------------
 Airbus A320-200 | {"crew": 2, "engines": {"num": 2, "type":
                    "IAE V2500"}}
(1 строка)
```

Можно посмотреть только сведения о двигателях:

```
SELECT model, specifications->'engines' AS engines
  FROM aircrafts
  WHERE aircraft_code = '320';
```

```
      model      |             engines
-----------------+---------------------------------
 Airbus A320-200 | {"num": 2, "type": "IAE V2500"}
(1 строка)
```

Чтобы получить еще более детальные сведения, например, о типе двигателей, нужно учитывать, что созданный JSON-объект имеет сложную структуру: он содержит вложенный JSON-объект. Поэтому нужно использовать оператор #> для указания пути доступа к ключу второго уровня.

```
SELECT model, specifications #> '{ engines, type }'
  FROM aircrafts
  WHERE aircraft_code = '320';
```

```
      model      |  ?column?
-----------------+-------------
 Airbus A320-200 | "IAE V2500"
(1 строка)
```

Задание. Подумайте, какие еще таблицы было бы целесообразно дополнить столбцами типа `json`/`jsonb`. Вспомните, что, например, в таблице «Билеты» (`tickets`) уже есть столбец такого типа — `contact_data`. Выполните модификации таблиц и измените в них одну-две строки для проверки правильности ваших решений.

Добавление пилотов:

```
demo=# ALTER TABLE flights ADD COLUMN pilots jsonb;
ALTER TABLE
```

```
demo=# SELECT flight_no FROM flights LIMIT 10;
 flight_no
-----------
 PG0405
 PG0402
 PG0403
 PG0404
 PG0405
 PG0404
 PG0403
 PG0405
 PG0404
 PG0403
(10 rows)

demo=# UPDATE flights
SET pilots='{
"quantity": 2,
"first pilots": {"name": "Alex", "age": 45},
"second pilot": {"name": "Andrew", "age": 30}}'::jsonb
WHERE flight_no='PG0405';
UPDATE 61
demo=# SELECT * FROM flights WHERE flight_no='PG0405';
```

```
 flight_id | flight_no |  scheduled_departure   |   scheduled_arrival    | departure_airpor
t | arrival_airport | status  | aircraft_code |    actual_departure    |     actual_arrival
     |                                                  pilots

-----------+-----------+------------------------+------------------------+-----------------
--+-----------------+---------+---------------+------------------------+-----------------
-----+-------------------------------------------------------------------------------------
-----------------------
         1 | PG0405    | 2016-09-13 08:35:00+03 | 2016-09-13 09:30:00+03 | DME
  | LED             | Arrived | 321           | 2016-09-13 08:44:00+03 | 2016-09-13 09:39:0
0+03 | {"quantity": 2, "first pilots": {"age": 45, "name": "Alex"}, "second pilot": {"age":
 30, "name": "Andrew"}}
(1 row)
```

Дополнительное описание мест в самолете:

```
demo=# ALTER TABLE seats ADD COLUMN description jsonb;
ALTER TABLE
demo=# \d seats
                    Table "bookings.seats"
     Column      |         Type          | Collation | Nullable | Default
-----------------+-----------------------+-----------+----------+---------
 aircraft_code   | character(3)          |           | not null |
 seat_no         | character varying(4)  |           | not null |
 fare_conditions | character varying(10) |           | not null |
 description     | jsonb                 |           |          |
Indexes:
    "seats_pkey" PRIMARY KEY, btree (aircraft_code, seat_no)
Check constraints:
    "seats_fare_conditions_check" CHECK (fare_conditions::text = ANY (ARRAY['Economy'::character varying::text, 'Comfort'::character varying::text
, 'Business'::character varying::text]))
Foreign-key constraints:
    "seats_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES aircrafts(aircraft_code) ON DELETE CASCADE
```

```
demo=# UPDATE seats SET description='{"first_place": {"near_window": true, "conditioner": "yes"}, "second_place": {"near_window": false, "conditioner": "no"}}
'::jsonb WHERE aircraft_code='319';
UPDATE 116
```

```
demo=# SELECT description FROM seats WHERE aircraft_code='319' LIMIT 1;
                                                      description
------------------------------------------------------------------------------------------------------------------------
 {"first_place": {"conditioner": "yes", "near_window": true}, "second_place": {"conditioner": "no", "near_window": false}}
(1 row)
```