Язык SQL, дз – 4

Бурмашев Григорий, БПМИ-208

18 октября 2022 г.

Номер 2

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
Check constraints:
    "progress_mark_check" CHECK (mark >= 3::numeric AND mark <= 5::numeric)
    "progress_term_check" CHECK (term = 1::numeric OR term = 2::numeric)
Foreign-key constraints:
    "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES students(record_book) ON UPDATE CASCADE ON DELETE CASCADE

edu=#
```

Основным атрибутом, на который хочется повесить ограничение является mark, но там ограничение уже есть.

```
edu=# ALTER TABLE progress ADD test_form text NOT NULL;
ALTER TABLE
edu=#
```

Добавил новый атрибут test_form

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

Добавил ограничение для test_form

```
edu=# INSERT INTO students VALUES (1, 'Grigory', 1, 1);
INSERT 0 1
edu=# INSERT INTO progress VALUES (1, 'my_subject', 2022, 2, 1, 'экзамен');
ERROR:  new row for relation "progress" violates check constraint "progress_check"
DETAIL:  Failing row contains (1, my_subject, 2022, 2, 1, экзамен).
edu=# INSERT INTO progress VALUES (1, 'my_subject', 2022, 2, 5, 'экзамен');
INSERT 0 1
```

Проверяю, что ограничение работает и все хорошо

```
edu=# INSERT INTO progress VALUES (1, 'my_subject', 2022, 1, 1, 'зачет');
ERROR:  new row for relation "progress" violates check constraint "progress_mark_check"
DETAIL:  Failing row contains (1, my_subject, 2022, 1, 1, зачет).
edu=#
```

Ограничения конфликтуют :(

```
edu=# ALTER TABLE progress DROP CONSTRAINT progress_mark_check
edu-# ;
ALTER TABLE
edu=# INSERT INTO progress VALUES (1, 'my_subject', 2022, 1, 1, 'зачет');
INSERT 0 1
edu=#
```

Удалил старое ограничение и все пофиксилось.

Ограничения можно выставить любого характера, например, зафиксировать набор возможных предметов, ограничить возможный год курса и прочее прочее.

Номер 9

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=# INSERT INTO students ( record_book, name, doc_ser, doc_num )
       VALUES ( 12300, '', 0402, 543281 );
INSERT 0 1
```

NULL != '', поэтому инсерт происходит успешно

```
edu=# DELETE FROM students WHERE name=''
edu-# ;
DELETE 1
edu=# ALTER TABLE students ADD CHECK ( name <> '' );
ALTER TABLE
edu=# INSERT INTO students ( record_book, name, doc_ser, doc_num )
       VALUES ( 12300, '', 0402, 543281 );
ERROR:  new row for relation "students" violates check constraint "students_name_check"
DETAIL:  Failing row contains (12300, , 402, 543281).
edu=#
```

Супер, чек начал работать

```
edu=# INSERT INTO students VALUES ( 12346, ' ', 0406, 112233 );
INSERT 0 1
edu=# INSERT INTO students VALUES ( 12347, '  ', 0407, 112234 );
INSERT 0 1
edu=# SELECT *, length( name ) FROM students;
 record_book |  name   | doc_ser | doc_num | length
-------------+---------+---------+---------+--------
           1 | Grigory |       1 |       1 |      7
       12346 |         |     406 |  112233 |      1
       12347 |         |     407 |  112234 |      2
(3 rows)
```

Но оказывается работает не так, как ожидалось

```
edu=# ALTER TABLE students ADD CHECK (trim(name) <> '');
ALTER TABLE
edu=# INSERT INTO students VALUES ( 12347, '  ', 0407, 112234 );
ERROR:  new row for relation "students" violates check constraint "students_name_check1"
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_check1"
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=#
```

Прочитал документацию и пофиксил.

В progress есть слабости у атрибутов subject и acad_year (текстовые столбцы)

Номер 17

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.

Создал представление с указанием координат каждого из аэропортов, это может быть полезно пилотам и диспетчерам

```
demo=# CREATE VIEW airports_coordinates as
SELECT airport_name, coordinates
FROM airports
;
CREATE VIEW
```

```
     airport_name     |              coordinates
----------------------+---------------------------------------
 Якутск               | (129.77099609375,62.093299865722656)
 Мирный               | (114.03900146484375,62.534698486328125)
 Хабаровск-Новый      | (135.18800354004,48.52799987793)
 Елизово              | (158.45399475097656,53.16790008544922)
 Хомутово             | (142.71800231933594,46.88869857788086)
 Владивосток          | (132.1479949951172,43.39899826049805)
 Пулково              | (30.262500762939453,59.80030059814453)
 Храброво             | (20.592599868774414,54.88999938964844)
 Кемерово             | (86.1072006225586,55.27009963989258)
 Челябинск            | (61.5033,55.305801)
```

А еще, например, отображение только Московских аэропортов, это полезно пассажирам – жителям Москвы

```
demo=# CREATE VIEW moscow_ports AS
demo-# SELECT * FROM airports
demo-# WHERE city = 'Москва';
CREATE VIEW
```

```
 airport_code | airport_name |  city  |              coordinates
      |   timezone
--------------+--------------+--------+-------------------------------
-------+---------------
 SVO          | Шереметьево  | Москва | (37.4146,55.972599)
      | Europe/Moscow
 VKO          | Внуково      | Москва | (37.2615013123,55.5914993286)
      | Europe/Moscow
 DME          | Домодедово   | Москва | (37.90629959106445,55.408798217
77344) | Europe/Moscow
(3 rows)
```

Номер 18

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.

142

Контрольные вопросы и задания

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.

Решил создать jsonb в таблице с аэропортами, можно собрать достаточно много информации об аэропортах, число конкретных сотрудников, время/стоимость поездки от аэропорта до города и прочие

```
demo=# ALTER TABLE airports_table ADD COLUMN specifications jsonb;
ALTER TABLE
```

```
demo=# UPDATE airports_table
SET specifications =
'{ "staff": 10000, "opened": 1959, "square": 570000}'::jsonb
WHERE airport_name = 'Шереметьево';
UPDATE 1
```

```
demo=# UPDATE airports_table
SET specifications =
'{ "staff": 6500, "opened": 1941, "square": 80000}'::jsonb
WHERE airport_name = 'Внуково';
UPDATE 1
```

```
demo=# SELECT * FROM airports_table WHERE airport_name IN ('Шереметьево', 'Внуково');
 airport_code | airport_name |  city  |          coordinates          |   timezone    |                    specifications
--------------+--------------+--------+-------------------------------+---------------+-------------------------------------------------
 VKO          | Внуково      | Москва | (37.2615013123,55.5914993286) | Europe/Moscow | {"staff": 6500, "opened": 1941, "square": 80000}
 SVO          | Шереметьево  | Москва | (37.4146,55.972599)           | Europe/Moscow | {"staff": 10000, "opened": 1959, "square": 570000}
(2 rows)
```