

Домашнее задание 3

Выполнил - Третьяков Александр Юрьевич

Упражнение 9

Задание

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`)?

Решение

Проверим наличие ограничений в таблице `students`

```
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)
Referenced by:
    TABLE "progress" CONSTRAINT "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES students(record_book) ON UPDATE CASCADE ON DELETE CASCADE
```

Вставка записи которая содержит `name` равное пустой строке будет выполнена, потому что это значение не `NULL`

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

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

Добавим ограничение на пустые строки в таблицу

Если попробовать сразу после `insert` сделанного ранее добавить ограничение, то возникнет ошибка, означающая что в таблице уже есть строки которые содержат `name = ''`

```
edu=# ALTER TABLE students ADD CHECK ( name <> '' );
ERROR:  check constraint "students_name_check" of relation "students" is violated by some row
edu=#
```

Удалим строку, добавим ограничение

```

edu=# delete from students where name = '';
DELETE 1
edu=# ALTER TABLE students ADD CHECK ( name <> '' );
ALTER TABLE
edu=# \d 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 (name <> ''::text)
Referenced by:
    TABLE "progress" CONSTRAINT "progress_record_book_fkey" FO

```

Выполнил вставку с `name = ''`, `name = ' '` и `name = ' '`. Проверим результат

```

edu=# INSERT INTO students VALUES ( 12347, ' ', 0406, 112233 );
ERROR: new row for relation "students" violates check constraint "students_name_check"
DETAIL: Failing row contains (12347, , 406, 112233).
edu=# INSERT INTO students VALUES ( 12347, ' ', 0406, 112233 );
INSERT 0 1
edu=# INSERT INTO students VALUES ( 12348, ' ', 0407, 112234 );
INSERT 0 1
edu=# SELECT *, length( name ) FROM students;

```

record_book	name	doc_ser	doc_num	length
12345	Иванов Иван	1234	567890	11
12346	Петров Петр	1235	567891	11
12347		406	112233	1
12348		407	112234	2

```

(4 rows)

```

Что бы не допустить значений "невидимок" можем воспользоваться следующим ограничением

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

Проведем тестирование невалидных случаев

```
edu=# INSERT INTO students VALUES (12300, '', 0402, 543281);
ERROR:  new row for relation "students" violates check constraint "students_name_check"
DETAIL:  Failing row contains (12300, , 402, 543281).
edu=#
edu=# INSERT INTO students VALUES (12346, '', 0406, 112233);
ERROR:  new row for relation "students" violates check constraint "students_name_check1"
DETAIL:  Failing row contains (12346, , 406, 112233).
edu=#
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=#
```

И валидных

```
edu=# INSERT INTO students VALUES (12348, 'Николаев Николай', 0408, 112235);
INSERT 0 1
edu=#
```

Таблица **progress** также уязвима для вставки пустых строк или строк состоящих только из пробелов. Нужно добавить аналогичные ограничения:

```
-- Для subject
ALTER TABLE progress ADD CHECK (trim(subject) <> '');

-- Для acad_year
ALTER TABLE progress ADD CHECK (trim(acad_year) <> '');
```