Лабораторная работа №4

Тема: Глава 5 из книги. Упр 2, 9, 17, 18

Группа: М8О-109СВ-24

Выполнил: **Гимазетдинов Дмитрий Русланович**

вернуться на главную

Упражнение 2

Дано:

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.

Решение:

Рассмотрим какие ограничения уже на таблицу `progerss` у нас есть:

```
\d progress
```

```
                              Таблица "public.progress"
   Столбец   |     Тип      | Правило сортировки | Допустимость NULL | По
умолчанию
-------------+--------------+--------------------+-------------------+-----
---------
 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
Ограничения-проверки:
    "progress_mark_check" CHECK (mark >= 3::numeric AND mark <= 5::numeric)
    "progress_term_check" CHECK (term = 1::numeric OR term = 2::numeric)
Ограничения внешнего ключа:
    "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES
students(record_book) ON UPDATE CASCADE ON DELETE CASCADE
```

Ограничения, которые уже существуют:

1. **CHECK (mark >= 3 AND mark <= 5)** — Оценка может быть только от 3 до 5.
2. **CHECK (term = 1 OR term = 2)** — Семестр может быть только 1 или 2.
3. **FOREIGN KEY (record_book)** — Внешний ключ связывает с таблицей *students*, обновляется и удаляется каскадно.

Возможные дополнительные ограничения:

1. **acad_year** — Ограничить формат учебного года, например, использовать регулярное выражение, чтобы формат был "YYYY-YYYY" (например, "2023-2024").
2. **subject** — Добавить ограничение по списку возможных предметов или длине строки (например, не более 100 символов).
3. **record_book** — Ограничить диапазон значений, например, чтобы было в диапазоне от 1 до 99999.
4. **Уникальность записи** — Добавить уникальный индекс для комбинации `record_book`, `subject`, `acad_year` и `term`, чтобы одна и та же запись не могла повториться для студента по предмету в конкретный год и семестр.

Проведем операции, которые нас просили по зазданию атрибута `test_from` и выведем структуру нашей таблицы:

```
edu=# \d progress
                              Таблица "public.progress"
   Столбец   |     Тип      | Правило сортировки | Допустимость NULL | По
умолчанию
-------------+--------------+--------------------+-------------------+-----
---------
 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         |                    |                   |
Ограничения-проверки:
    "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_mark_check" CHECK (mark >= 3::numeric AND mark <= 5::numeric)
    "progress_term_check" CHECK (term = 1::numeric OR term = 2::numeric)
Ограничения внешнего ключа:
    "progress_record_book_fkey" FOREIGN KEY (record_book) REFERENCES
students(record_book) ON UPDATE CASCADE ON DELETE CASCADE
```

Создадим информацию про пользователей:

```
edu=# \d
 public | pilots   | table | postgres
 public | progress | table | postgres
 public | students | table | postgres

edu=# \d students
 record_book | numeric(5,0) |           | not null |
 name        | text         |           | not null |
 doc_ser     | numeric(4,0) |           |          |
 doc_num     | numeric(6,0) |           |          |

edu=# insert into students values (11111, 'Petr', 1111, 111111);
INSERT 0 1
edu=# insert into students values (22222, 'Dima', 2222, 222222);
INSERT 0 1
edu=#
```

Проверим вставку значний с новыми ограничениями:

```
edu=# insert into progress values ( 11111, 'math', '2024', 1, 5,
'экзамен');
INSERT 0 1
```

Получилось поставить Petr пятерку за экзамен.

Проверим получится ли поставить зачет.

```
ОШИБКА:  новая строка в отношении "progress" нарушает ограничение-проверку
"progress_mark_check"
ПОДРОБНОСТИ:  Ошибочная строка содержит (11111, math, 2024, 1, 1, зачет).
```

Нарушается проверка `"progress_mark_check"`. А именно:

```
"progress_mark_check" CHECK (mark >= 3::numeric AND mark <= 5::numeric)
```

Это означает что поле `mark` может быть от 3 до 4.

Удалим старое ограниение:

```
alter table progress drop constraint "progress_mark_check";
```

Вот что у нас получилось:

```
edu=# alter table progress drop constraint "progress_mark_check";
ALTER TABLE
edu=# insert into progress values ( 11111, 'math', '2024', 1, 1, 'зачет');
INSERT 0 1
edu=#
```

Мы смогли записать новую запись.

Посмотрим, что получилось:

```
edu=# select * from progress;
       11111 | math    | 2024      |    1 |    5 | экзамен
       11111 | math    | 2024      |    1 |    1 | зачет
```

Упражнение 9

Дано:

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:

Сейчас мы можем вставлять пустые строки и строки содержащие пробелы в таблицу с студентами по атрибуту имени.

Давайте исправим это:

```
alter table students add check( length(trim(name)) <> 0 );
ALTER TABLE
```

И теперь попробуем записать запись с пробелами и пусыми строками.

```
edu=# INSERT INTO students VALUES ( 12346, '', 0406, 112233 );
ОШИБКА:  новая строка в отношении "students" нарушает ограничение-проверку
"students_name_check"
ПОДРОБНОСТИ:  Ошибочная строка содержит (12346, , 406, 112233).
edu=# INSERT INTO students VALUES ( 12346, ' ', 0406, 112233 );
ОШИБКА:  новая строка в отношении "students" нарушает ограничение-проверку
"students_name_check"
ПОДРОБНОСТИ:  Ошибочная строка содержит (12346,  , 406, 112233).
```

Все работает!!!

В таблице `progress`, так же есть подобные недостатки, например предмет можно сделать пустой строкой. Вместо этого можно сделать перечисление предметов, или создать отлдельную таблицу с предметами и сделать ограничени по ключу!

Упражнение 17

Дано:

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.

Решение:

Предлагаю вспомнить структуру наше БД авиаперевозки.

```
demo=# \d
                  List of relations
  Schema  |         Name          |   Type   |  Owner
----------+-----------------------+----------+----------
 bookings | aircrafts             | view     | postgres
 bookings | aircrafts_data        | table    | postgres
 bookings | airports              | view     | postgres
 bookings | airports_data         | table    | postgres
 bookings | boarding_passes       | table    | postgres
 bookings | bookings              | table    | postgres
 bookings | flights               | table    | postgres
 bookings | flights_flight_id_seq | sequence | postgres
 bookings | flights_v             | view     | postgres
 bookings | routes                | view     | postgres
 bookings | seats                 | table    | postgres
 bookings | ticket_flights        | table    | postgres
 bookings | tickets               | table    | postgres
(13 rows)
```

У нас в ней есть уже 3 вьюшки, которые слушат для мультиязычности таблицы. Можно еще создать следующие вьюшки:

- посчитать количество перелетов определенной продолжительности из таблицы `routes`
- узнать какие аэропорта начинаются с гласной.

Кол-во перелетов определенной продолжительности:

```
create view w_durations as
select distinct
    duration,
    count(*) over(partition by duration) as count

from routes;

select * from w_durations;
```

```
duration | count
----------+-------
 00:25:00 |    24
 00:30:00 |    22
 00:35:00 |    10
 00:40:00 |     2
 00:45:00 |     8
 00:50:00 |    26
 00:55:00 |    26
 01:00:00 |    14
 01:05:00 |    18
 01:10:00 |     8
 01:15:00 |    20
 01:20:00 |    10
 01:25:00 |    22
 01:30:00 |    16
 01:35:00 |     4
 01:40:00 |    10
 01:45:00 |    26
 01:50:00 |    18
 01:55:00 |    18
...
```

```
create view w_beda as
    select airport_name
    from airports
    where lower(trim(substring(airport_name for 1))) in ('а', 'у', 'ы',
'я', 'и', 'е', 'о', 'ю', 'э')
    order by airport_name;

select * from w_beda;
```

```
demo=#
select * from w_beda;
CREATE VIEW
     airport_name
----------------------
 Абакан
 Анадырь
 Астрахань
 Елизово
 Емельяново
 Иваново-Южный
 Игнатьево
 Ижевск
 Иркутск
 Омск-Центральный
```

```
 Оренбург-Центральный
 Орск
 Уйташ
 Ульяновск-Восточный
 Усинск
 Усть-Илимск
 Усть-Кут
 Уфа
 Ухта
 Элиста
 Якутск
(21 rows)
```

Упражнение 18

Дано: Добавить технические характеристики самолетов.

Решение:

Добавим 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;
```

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';
```

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)
```

Посмотрим 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)
```

Посмотрим информацию во вложенном `jsonb`:

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

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

Чтобы добавить новый атрибут для хранения списка предоставляемой еды в формате JSONB в таблице seats, можно использовать следующий SQL-запрос:

```
ALTER TABLE bookings.seats
ADD COLUMN meals jsonb;
```

Чтобы добавить список еды для определенного места, можно выполнить такой запрос:

```
UPDATE bookings.seats
SET meals = '{
                "завтрак": "первое",
```

```
                "обед": "чай"
            }'::jsonb
WHERE aircraft_code = '319' AND seat_no = '2C';
```

Получаем:

```
demo=# select * from seats where aircraft_code = '319' and meals is not
null;
 aircraft_code | seat_no | fare_conditions |                meals
---------------+---------+-----------------+-------------------------------
-------
 319           | 2C      | Business        | {"обед": "чай", "завтрак":
"первое"}
(1 row)
```
