

ЗАНЯТИЕ 1.4

Углубление в SQL



Ирина Хомутова

Software developer in Eltex LTD



irinavkhomutova@icloud.com

—

Таблицы

Создание и удаление таблиц

Создание таблиц

CREATE TABLE - команда создания таблиц

CREATE TABLE название_таблицы

(названиестолбца1 типданных атрибуты_столбца1,


названиестолбца2 типданных атрибуты_столбца2,

.....

названиестолбцаN типданных атрибуты_столбцаN,

атрибуты_таблицы

);



```
CREATE TABLE customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
);
```

Удаление таблиц

```
DROP TABLE table1 [, table2, ...];
```

```
[]
```

```
DROP TABLE customers;
```

```
[]
```




Ограничения столбцов и таблиц

PRIMARY KEY

С помощью выражения PRIMARY KEY столбец можно сделать первичным ключом.

Первичный ключ уникально идентифицирует строку в таблице.

Первичные ключи могут быть простыми и составными



```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
)
```

```
CREATE TABLE Customers
```

```
(
```

```
  Id SERIAL,
```

```
  FirstName CHARACTER VARYING(30),
```

```
  LastName CHARACTER VARYING(30),
```

```
  Email CHARACTER VARYING(30),
```

```
  Age INTEGER,
```

```
  PRIMARY KEY(Id)
```

```
);
```

```
CREATE TABLE OrderLines
(
    OrderId INTEGER,
    ProductId INTEGER,
    Quantity INTEGER,
    Price MONEY,
    PRIMARY KEY(OrderId, ProductId)
);
```

UNIQUE

В столбце в каждой строке уникальные/не совпадающие значения

[]

```
CREATE TABLE Customers
```

```
(
```

```
    Id INT PRIMARY KEY IDENTITY,
```

```
    Age INT,
```

```
    FirstName NVARCHAR(20),
```

```
    LastName NVARCHAR(20),
```

```
    Email VARCHAR(30) UNIQUE,
```

```
    Phone VARCHAR(20) UNIQUE
```

```
)
```

```
CREATE TABLE Customers
```

```
(  
  Id SERIAL PRIMARY KEY,  
  FirstName CHARACTER VARYING(20),  
  LastName CHARACTER VARYING(20),  
  Email CHARACTER VARYING(30),  
  Phone CHARACTER VARYING(30),  
  Age INTEGER,  
  UNIQUE(Email, Phone)  
);
```

```
CREATE TABLE Customers
```

```
(  
    Id SERIAL PRIMARY KEY,  
    FirstName CHARACTER VARYING(20),  
    LastName CHARACTER VARYING(20),  
    Email CHARACTER VARYING(30),  
    Phone CHARACTER VARYING(30),  
    Age INTEGER,  
    UNIQUE(Email),  
    UNIQUE(Phone)  
);
```

NULL и NOT NULL

Указание, может ли столбец принимать значение NULL

Если ничего не указано, то допускается значение NULL

Столбец выступающий в роли первичного ключа всегда NOT NULL

[]

```
CREATE TABLE Customers
```

```
(  
    Id SERIAL PRIMARY KEY,  
    FirstName CHARACTER VARYING(20) NOT NULL,  
    LastName CHARACTER VARYING(20) NOT NULL,  
    Age INTEGER  
);
```

DEFAULT

Определение значения по умолчанию

[]

```
CREATE TABLE Customers
(  
    Id SERIAL PRIMARY KEY,  
    FirstName VARCHAR(20),  
    LastName VARCHAR(20),  
    Age INTEGER DEFAULT 18  
);
```

CHECK

Ограничения для диапазона значений, которые могут храниться в столбце

```
[]  
CREATE TABLE Customers  
(  
    Id SERIAL PRIMARY KEY,  
    FirstName VARCHAR(20),  
    LastName VARCHAR(20),  
    Age INTEGER DEFAULT 18 CHECK(Age > 0 AND Age < 100),  
    Email VARCHAR(30) UNIQUE CHECK(Email != ''),  
    Phone VARCHAR(20) UNIQUE CHECK(Phone != '')  
);
```

```
CREATE TABLE Customers
(
    Id SERIAL PRIMARY KEY,
    Age INTEGER DEFAULT 18,
    FirstName VARCHAR(20),
    LastName VARCHAR(20),
    Email VARCHAR(30) UNIQUE,
    Phone VARCHAR(20) UNIQUE,
    CHECK((Age >0 AND Age<100) AND (Email !='') AND (Phone !=''))
);
```

CONSTRAINT

Установка имени ограничений.

В качестве ограничений могут использоваться PRIMARY KEY, UNIQUE, CHECK.

Необязательно задавать имена ограничений, при установке соответствующих атрибутов SQL Server автоматически определяет их имена

```
[ ]
CREATE TABLE Customers
(
    Id SERIAL CONSTRAINT customer_Id PRIMARY KEY,
    Age INTEGER CONSTRAINT customers_age_check CHECK(Age >0 AND
Age < 100),
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    Email VARCHAR(30) CONSTRAINT customers_email_key UNIQUE,
    Phone VARCHAR(20) CONSTRAINT customers_phone_key UNIQUE
);
```

```
CREATE TABLE Customers
```

```
(
```

```
  Id SERIAL,
```

```
  Age INTEGER,
```

```
  FirstName VARCHAR(20) NOT NULL,
```

```
  LastName VARCHAR(20) NOT NULL,
```

```
  Email VARCHAR(30),
```

```
  Phone VARCHAR(20),
```

```
  CONSTRAINT customer_Id PRIMARY KEY(Id),
```

```
  CONSTRAINT customers_age_check CHECK(Age >0 AND Age < 100),
```

```
  CONSTRAINT customers_email_key UNIQUE(Email),
```

```
  CONSTRAINT customers_phone_key UNIQUE(Phone)
```

```
);
```

Изменение таблиц

ALTER TABLE название_таблицы

{

ADD *названиестолбца типданныхстолбца [ограничениястолбца]* |

DROP COLUMN название_столбца |

ALTER COLUMN *названиестолбца параметрыстолбца* |

ADD [CONSTRAINT] определение_ограничения |

DROP [CONSTRAINT] имя_ограничения

}

Добавление данных

INSERT INTO имя_таблицы (столбец1, столбец2, ... столбецN)

VALUES (значение1, значение2, ... значениеN)

INSERT INTO Products (ProductName, Price, Manufacturer)

VALUES ('iPhone X', 71000, 'Apple');

Возвращение значений

RETURNING

[]

INSERT INTO Products

(ProductName, Manufacturer, Price, ProductCount)

VALUES('Desire 12', 'HTC', 8, 21000)

RETURNING id;

Обновление данных

UPDATE имя_таблицы

SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN

[WHERE условие_обновления]

[]

UPDATE Products

SET Price = Price + 3000;

UPDATE Products

SET Manufacturer = 'Samsung Inc.'

WHERE Manufacturer = 'Samsung';

Удаление данных

DELETE FROM имя_таблицы

[WHERE условие_удаления]

[]

DELETE FROM Products

WHERE Manufacturer='HTC' AND Price < 15000;

Обобщенные табличные выражения

Обобщенным табличным выражением (ОТВ) (*Common Table Expression* - сокращенно *CTE*) называется именованное табличное выражение, поддерживаемое языком SQL.

WITH имя_ОТВ (список__столбцов) AS

(внутренний__запрос)

внешний_запрос

Используются в следующих двух типах запросов:

- нерекурсивных;
- рекурсивных.

ОТВ и нерекурсивные запросы

Можно использовать в качестве альтернативы производным таблицам и представлениям. Определяется посредством предложения WITH и дополнительного запроса, который ссылается на имя, используемое в предложении WITH.

```
[ ]  
WITH sumprod(id, sumwieght)  
as (  
  SELECT id, sum(weight) as sw  
  FROM public.products  
  group by id  
)  
SELECT sid, pid, jid, num, sumwieght  
FROM public.supply  
inner join sumprod  
on id = pid  
where sid = 'S1';
```

ОТВ и рекурсивные запросы

Вычисление чего-то итерациями до того, как будет выполнено некоторое условие

WITH имя_ОТВ (список__столбцов) AS

(стартовый__запрос

union [all]

рекурсивный__запрос__к__имя_ОТВ

) внешний_запрос

Задача

Посчитать факториал:

$$n! = 1234 \dots (n-1)n$$

[]

```
WITH RECURSIVE r AS (
```

```
-- стартовая часть рекурсии (т.н. "anchor")
```

```
SELECT
```

```
  1 AS i,
```

```
  1 AS factorial
```

```
UNION
```

-- рекурсивная часть

SELECT

i+1 AS i,

factorial * (i+1) as factorial

FROM r

WHERE i < 10

)

SELECT * FROM r;

Алгоритм примерно такой:

1. Извлечь стартовые данные

2. Подставить полученные данные с предыдущей итерации в «рекурсивную» часть запроса.

3. Если в текущей итерации рекурсивной части не пустая строка, то добавляем ее в результирующую выборку, а также пометить данные, как данные для следующего вызова рекурсивной части (п. 2), иначе завершить обработку

Представления

CREATE VIEW имя_представления

AS запрос_представления

Хранимый запрос к базе данных

Иногда называют виртуальная таблица, но в этой таблице данные не хранятся, а хранится только сам запрос.

Можно обращаться как к обычной таблице.

Для хранения сложных запросов и обращения к полученным данным из простых запросов.

CREATE OR REPLACE view spj

as

SELECT job.name AS job_name,

job.city AS job_city,

products.color,

products.name AS product_name,

products.weight,

products.city AS product_city,

shipper.name AS shipper_name,

shipper.raiting,

shipper.city AS shipper_city

FROM supply

JOIN job ON job.id = supply.jid

JOIN products ON products.id = supply.pid

JOIN shipper ON shipper.id = supply.sid;

Различия CTE и VIEW

- Представления могут быть проиндексированы, но ОТВ не могут
- ОТВ отлично работают с рекурсией
- Представления - физические объекты БД, можно обращаться из нескольких запросов:
 - гибкость
 - централизованный подход
- ОТВ - временные:
 - создаются, когда будут использоваться
 - удаляются после использования
 - не хранится статистика на сервере

ИТОГИ ЗАНЯТИЯ

Мы сегодня изучили:

- Создание и модификация таблиц
- Вставка и модификация данных
- рекурсивные и нерекурсивные запросы

—

ВОПРОСЫ

Домашнее задание

Домашнее задание

Спроектируйте базу данных для следующих сущностей:

- Язык (в смысле английский, французский и тп)
- Народность (в смысле славяне, англосаксы и тп)
- Страны (в смысле Россия, Германия и тп)

Правила следующие:

- На одном языке может говорить несколько народностей
- Одна народность может входить в несколько стран
- Каждая страна может состоять из нескольких народностей

Пришлите скрипты создания таблиц и заполнения их 5-ю строками данных



НЕТОЛОГИЯ
групп

Спасибо за
внимание!

Ирина Хомутова



irinaikhomutova@icloud.com