Lesson 25 - Joins

Повторение

1. Теория из прошлого урока
2. Разбор домашнего задания

Legend

Первичный ключ (Primary key)

Первичный ключ — это столбец или комбинация столбцов в таблице базы данных, которые **однозначно идентифицируют каждую запись**.

Основные свойства:

1. **Уникальность**: каждое значение первичного ключа должно быть уникальным.
2. **NOT NULL**: значения первичного ключа не могут быть NULL.
3. **Ограничение**: в каждой таблице может быть только один первичный ключ.
4. **Индекс**: автоматически создаётся уникальный индекс для ускорения поиска.

1. Создание первичного ключа:

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,        -- user_id — первичный ключ
    username VARCHAR(100),
    email VARCHAR(100)
);
```

2. Составной первичный ключ:

```
CREATE TABLE Orders (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id) -- Ключ состоит из двух столбцов
);
```

Внешний ключ (Foreign key)

Внешний ключ — это столбец (или набор столбцов) в таблице, который используется для создания связи между двумя таблицами. Он указывает на первичный ключ другой таблицы и обеспечивает целостность данных.

Основные свойства:

1. **Связь между таблицами**: внешний ключ связывает строки одной таблицы со строками другой.
2. **Целостность данных**: позволяет избежать ситуаций, когда в подчинённой таблице (child table) есть записи, указывающие на несуществующие строки в родительской таблице (parent table).
3. **Каскадные действия**: при изменении или удалении записи в родительской таблице можно настроить поведение:
   • ON DELETE CASCADE: удаление записи в родительской таблице приведёт к удалению связанных строк в подчинённой.
   • ON UPDATE CASCADE: обновление значения в родительской таблице изменит связанные строки в подчинённой.

Создание внешнего ключа

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) -- Внешний
ключ
);
```

Настройка поведения при каскадных действиях:

```
1. ON DELETE CASCADE:
Удаление строки из родительской таблицы приводит к удалению связанных строк
из подчинённой.

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE
CASCADE
);
```

```
2. ON UPDATE CASCADE:
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON UPDATE
CASCADE
);

3. SET NULL:
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE SET
NULL
);
```

Виды связей между таблицами

Связи между таблицами определяются типами отношений, которые существуют между записями в разных таблицах. Основные виды связей:

1. Один к одному (One-to-One)
Каждая запись одной таблицы связана **только с одной** записью другой таблицы, и наоборот.

Пример:
• Таблица Users: хранит информацию о пользователях.
• Таблица UserDetails: хранит дополнительную информацию о пользователях.

Реализация:

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE UserDetails (
    UserID INT PRIMARY KEY,
```

```
    Address VARCHAR(255),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Когда использовать:
• Когда данные логически разделены, но имеют строгую связь.
• Например, данные паспорта и водительского удостоверения для одного человека.

2. Один ко многим (One-to-Many)
Одна запись в таблице связана с несколькими записями в другой таблице.

Пример:
• Таблица Authors: хранит авторов книг.
• Таблица Books: хранит книги, написанные авторами.

Реализация:

```
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(255),
    AuthorID INT,
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

Когда использовать:
• Когда одна сущность может быть связана с множеством других.
• Например, авторы и их книги, клиенты и их заказы.

3. Многие ко многим (Many-to-Many)
Каждая запись одной таблицы может быть связана с несколькими записями другой таблицы, и наоборот.

Пример:
• Таблица Students: содержит список студентов.
• Таблица Courses: содержит список курсов.
• Студенты могут записываться на несколько курсов, а курсы могут иметь нескольких студентов.

Реализация через таблицу-связку:

```sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    Title VARCHAR(255)
);

CREATE TABLE Enrollments (
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID),
    PRIMARY KEY (StudentID, CourseID)
);
```

Когда использовать:
• Когда нужно моделировать сложные отношения.
• Например, студенты и курсы, врачи и пациенты, продукты и заказы.

JOINS

JOINS используются для объединения данных из двух или более таблиц на основе связанных столбцов. Это один из основных инструментов для работы с реляционными базами данных.

Типы JOINS

1. **INNER JOIN (Внутреннее соединение)**
   Возвращает только те строки, у которых есть совпадения в обеих таблицах.
   Синтаксис:

```sql
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

2. **LEFT JOIN (Левое соединение или LEFT OUTER JOIN)**
   Возвращает все строки из левой таблицы и совпадающие строки из правой. Если

совпадений нет, в столбцах правой таблицы будет NULL.
Синтаксис:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

3. **RIGHT JOIN (Правое соединение или RIGHT OUTER JOIN)**
   Возвращает все строки из правой таблицы и совпадающие строки из левой. Если совпадений нет, в столбцах левой таблицы будет NULL.
   Синтаксис:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

4. **FULL OUTER JOIN (Полное внешнее соединение)**
   Возвращает все строки из обеих таблиц, независимо от того, есть ли совпадения. Если совпадений нет, столбцы от другой таблицы заполняются NULL.
   Синтаксис:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
FULL OUTER JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

5. **CROSS JOIN (Декартово произведение)**
   Возвращает все возможные комбинации строк из обеих таблиц.
   Синтаксис:

```
SELECT Employees.Name, Departments.DepartmentName
FROM Employees
CROSS JOIN Departments;
```

6. **SELF JOIN (Самосоединение)**
   Это соединение таблицы с самой собой, полезно для работы с иерархическими структурами (например, для нахождения подчинённых сотрудников).
   Синтаксис:

```sql
SELECT A.Name AS Manager, B.Name AS Employee
FROM Employees A
LEFT JOIN Employees B
ON A.EmployeeID = B.ManagerID;
```

Примеры с урока

```sql
CREATE TABLE authors
(
    id        SERIAL PRIMARY KEY,
    full_name VARCHAR
);

CREATE TABLE books
(
    id          SERIAL PRIMARY KEY,
    title       VARCHAR,
    description VARCHAR
);

CREATE TABLE authors_books
(
    id        SERIAL PRIMARY KEY,
    author_id INT,
    book_id   INT,
    FOREIGN KEY (author_id) REFERENCES authors (id),
    FOREIGN KEY (book_id) REFERENCES books (id)
);



INSERT INTO authors(full_name)
VALUES ('Лев Толстой'),
       ('Роберт Кийосаки'),
       ('Дейл Карнеги');
-- author | book
-- 1      - 1
-- 2      - 2
-- 3      - 3
-- 1      - 4
-- 2      - 5
-- 3      - 5

INSERT INTO authors_books (author_id, book_id)
VALUES (1, 1),
```

```
       (2, 2),
       (3, 3),
       (1, 4),
       (2, 5),
       (3, 5);


-- Реализовать запрос получение id автора, названия автора, название его книги
SELECT a.id, a.full_name, b.title
FROM authors a
JOIN authors_books ab ON a.id = ab.author_id
JOIN books b ON b.id = ab.book_id;

INSERT INTO books(title)
VALUES ('Война и мир'),
       ('Анна Каренина'),
       ('Богатый папа, бедный папа'),
       ('Как завоевать друзей');

INSERT INTO books(title)
VALUES ('Богатый друг');



CREATE TABLE users
(
    id        SERIAL PRIMARY KEY,
    full_name VARCHAR,
    age       INT
);


CREATE TABLE user_details
(
    id              SERIAL PRIMARY KEY,
    inn             VARCHAR UNIQUE,
    passport_number VARCHAR UNIQUE,
    Address         VARCHAR,
    user_id         INT,
    FOREIGN KEY (user_id) REFERENCES users (id)
);

INSERT INTO users (full_name, age)
VALUES ('Vasya Federov', 30),
       ('John Doe', 45),
```

```
       ('Fed Joe', 25);

INSERT INTO users (full_name, age)
VALUES ('Lebron Federov', 24);


INSERT INTO user_details (inn, passport_number, Address, user_id)
VALUES ('123456789', 'A123456', 'Test 1', 1),
       ('123456784', 'A123454', 'Test 2', 2),
       ('123456785', 'A123457', 'Test 3', 3);


-- u.id (1, 2, 3, 4)
-- ud.user_id (1, 2, 3)

-- (1, 2, 3)

SELECT u.id, u.full_name, u.age, ud.inn, ud.address
FROM users u,
     user_details ud
WHERE u.id = ud.user_id;

SELECT u.id, u.full_name, u.age, ud.inn, ud.address
FROM users u
         JOIN user_details ud ON u.id = ud.user_id;

SELECT u.id, u.full_name, u.age, ud.inn, ud.address
FROM users u
         FULL JOIN user_details ud ON u.id = ud.user_id;


CREATE TABLE orders
(
    id       SERIAL PRIMARY KEY,
    amount   NUMERIC,
    quantity INT,
    user_id  INT,
    FOREIGN KEY (user_id) REFERENCES users (id)
);

INSERT INTO users (full_name, age)
VALUES ('Vasya Fedeorv', 35),
       ('John Ddoe', 30);

INSERT INTO orders (amount, quantity, user_id)
VALUES (200, 5, 1),
```

```
       (300, 5, 2);


INSERT INTO orders (amount, quantity, user_id)
VALUES (200, 5, 5);
```

ДЗ-25

Часть 1

```
Вводны данные

-- Таблица для департаментов
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,        -- Уникальный идентификатор
департамента
    DepartmentName VARCHAR(100) NOT NULL  -- Название департамента
);

-- Таблица для сотрудников
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,           -- Уникальный идентификатор
сотрудника
    Name VARCHAR(100) NOT NULL,           -- Имя сотрудника
    DepartmentID INT,                    -- Идентификатор департамента
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID),  --
Внешний ключ для департамента
);

-- Вставка данных в таблицу Departments
INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
(1, 'IT'),
(2, 'HR'),
(3, 'Sales');

-- Вставка данных в таблицу Employees
INSERT INTO Employees (EmployeeID, Name, DepartmentID, ManagerID)
VALUES
(1, 'Иван', 1),  -- Иван — менеджер, не имеет менеджера
(2, 'Ольга', 2),    -- Ольга — сотрудник департамента HR, подчинён Ивану
```

```
(3, 'Алексей', 1),  -- Алексей — сотрудник департамента IT, подчинён Ивану
(4, 'Марина', 3);  -- Марина — сотрудник департамента Sales, подчинён Ольге
```

```
**Задачи для практики**

1. **INNER JOIN:**
Напишите запрос, который выводит список сотрудников и их отделов.

2. **LEFT JOIN:**
Найдите всех сотрудников, включая тех, кто не привязан к отделу.

3. **RIGHT JOIN:**
Напишите запрос для отображения всех отделов и соответствующих сотрудников, включая те отделы, в которых нет сотрудников.
```

Часть 2

```
0. Привести по 2 примера на виды связей между таблицами: one-to-one, one-to-many, many-to-many

1. **Один к одному:**
• Создайте таблицы Users и Passports, где каждому пользователю соответствует только один паспорт.
• Напишите SQL-запрос для получения имени пользователя и его номера паспорта.

2. **Один ко многим:**
• Создайте таблицы Authors и Books, где каждый автор может написать несколько книг.
• Напишите запрос для получения всех книг определённого автора.

3. **Многие ко многим:**
• Создайте таблицы Teachers, Classes и таблицу-связку teachers_classes, где учителя могут вести несколько классов, а классы могут быть привязаны к нескольким учителям.
• Напишите запрос для получения списка учителей, которые ведут определённый класс.
```