Let's create a simple employee management scenario using a composite key with an employees table that tracks employee roles in different departments. In this case, we'll use a composite key consisting of employee\_id and department\_id to ensure that each employee can have only one role in a department.

**Business Case Scenario**

**Scenario**: In a company, an employee can work in multiple departments, but they can only hold one role per department at any given time. To model this relationship, we'll create an employees table to store employee information and a roles table to link employees to their respective roles in departments.

**Step 1: Create the employees Table**

Here’s how we can create the employees table:

sql

Copy code

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY, -- Primary key for the employee

name VARCHAR(100) NOT NULL, -- Name of the employee, must not be null

hire\_date DATE NOT NULL -- Date the employee was hired, must not be null

);

**Step 2: Create the roles Table with Composite Key**

Now, we’ll create a roles table that uses a composite key consisting of employee\_id and department\_id.

sql

Copy code

CREATE TABLE roles (

employee\_id INT NOT NULL, -- Reference to the employee (foreign key)

department\_id INT NOT NULL, -- Reference to the department (foreign key)

role VARCHAR(50) NOT NULL, -- Role of the employee in the department

PRIMARY KEY (employee\_id, department\_id), -- Composite primary key

FOREIGN KEY (employee\_id) REFERENCES employees(employee\_id) ON DELETE CASCADE, -- Foreign key constraint

FOREIGN KEY (department\_id) REFERENCES departments(department\_id) ON DELETE CASCADE -- Foreign key constraint

);

**Explanation of the Composite Key**

* **Composite Primary Key**: The PRIMARY KEY (employee\_id, department\_id) statement indicates that the combination of employee\_id and department\_id must be unique. This means an employee can only hold one role in a specific department.
* **Foreign Keys**:
  + The employee\_id references the employee\_id in the employees table.
  + The department\_id would reference a departments table (which we'll assume exists for this example).

**Step 3: Create the departments Table**

To complete the scenario, let’s also create a departments table.

sql

Copy code

CREATE TABLE departments (

department\_id SERIAL PRIMARY KEY, -- Primary key for the department

department\_name VARCHAR(100) NOT NULL UNIQUE -- Name of the department, must be unique

);

**Step 4: Insert Sample Data**

Now, let’s insert some sample data into the employees, departments, and roles tables.

sql

Copy code

-- Insert sample departments

INSERT INTO departments (department\_name) VALUES

('Human Resources'),

('Engineering'),

('Marketing');

-- Insert sample employees

INSERT INTO employees (name, hire\_date) VALUES

('Alice Johnson', '2022-01-15'),

('Bob Smith', '2021-05-30'),

('Charlie Brown', '2023-03-25');

-- Insert sample roles

INSERT INTO roles (employee\_id, department\_id, role) VALUES

(1, 1, 'Recruiter'), -- Alice in HR

(1, 2, 'Software Engineer'), -- Alice in Engineering

(2, 1, 'HR Manager'), -- Bob in HR

(2, 3, 'Marketing Specialist'), -- Bob in Marketing

(3, 2, 'Data Scientist'); -- Charlie in Engineering

**Step 5: Demonstrate the Composite Key**

Now, let’s demonstrate the uniqueness enforced by the composite key.

1. **Inserting a Record with Duplicate Composite Key**:

sql

Copy code

INSERT INTO roles (employee\_id, department\_id, role) VALUES

(1, 1, 'Senior Recruiter'); -- This will fail because Alice already has a role in HR

This will fail because the combination of employee\_id = 1 (Alice) and department\_id = 1 (HR) already exists in the roles table.

1. **Inserting a Valid Record**:

sql

Copy code

INSERT INTO roles (employee\_id, department\_id, role) VALUES

(3, 1, 'HR Intern'); -- This will succeed because Charlie has no current role in HR

**Step 6: Check the Data**

You can check the data in the roles table to verify the inserts:

sql

Copy code

SELECT \* FROM roles;

**Sample Output**

| **employee\_id** | **department\_id** | **role** |
| --- | --- | --- |
| 1 | 1 | Recruiter |
| 1 | 2 | Software Engineer |
| 2 | 1 | HR Manager |
| 2 | 3 | Marketing Specialist |
| 3 | 2 | Data Scientist |
| 3 | 1 | HR Intern |

**Summary of the DDL**

Here’s the complete DDL for the tables, including the composite key:

sql

Copy code

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

hire\_date DATE NOT NULL

);

CREATE TABLE departments (

department\_id SERIAL PRIMARY KEY,

department\_name VARCHAR(100) NOT NULL UNIQUE

);

CREATE TABLE roles (

employee\_id INT NOT NULL,

department\_id INT NOT NULL,

role VARCHAR(50) NOT NULL,

PRIMARY KEY (employee\_id, department\_id),

FOREIGN KEY (employee\_id) REFERENCES employees(employee\_id) ON DELETE CASCADE,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id) ON DELETE CASCADE

);

**Business Scenario Question**

**Question**: How does using a composite key in the roles table help maintain data integrity, and what happens if an employee tries to take on more than one role in the same department?

This question encourages discussion about the importance of composite keys in enforcing unique relationships between multiple fields and how they can help in modeling real-world relationships (like employee roles in departments) accurately.