**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**Контрольная работа №5**  
**на тему «Информационная система Вуза»**

по дисциплине «Управление данными»

Выполнил: студент группы ИСТ-312, Кандиков М.В.

«15» Декабря 2024 г. \_\_\_\_\_\_\_\_\_\_\_/М.В. Кандиков /

Принял: старший преподаватель кафедры БИС

К. С. -Х. Н. Медведев Сергей Алексеевич

«20» Декабря 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ С.Аа. Медведев /

**Содержание**

**1. Титульный лист 1**

**2. Содержание 2**

**3. Основная часть 3**

3.1. Предметная область 3

3.2. Список нормализованных таблиц 3

3.2. Запросы в реляционной алгебре 4

3.3 SQL код для создания БД 4

3.4 Реализация запросов в виде программируемых объектов 11

3.5 Описание структуры базы данных 13

3.6 Выводы 18

**Основная часть**

**Предметная область:**

Разработка информационной системы для управления данными в вузе. Основная цель системы — организовать учет данных о студентах, преподавателях, кафедрах, группах, стипендиях, специальностях и дисциплинах, чтобы упростить процессы обработки и анализа информации.

**Функциональные задачи системы:**

1. Ведение подробной информации о студентах, включая их статус обучения и принадлежность к группам.
2. Управление назначением стипендий, учет их типов и сроков выплат.
3. Учет дисциплин, которые преподаются в рамках определенных специальностей.
4. Формирование аналитической информации для поддержки учебного процесса.

Данная система призвана обеспечивать надежное хранение данных и предоставлять гибкие инструменты для их использования.

**Список таблиц:**

1. **Студенты:**
   * Поля: StudentID (PK), PassportNumber (AK), LastName, FirstName, MiddleName, Status, Phone, Email, GroupID (FK).
2. **Преподаватели:**
   * Поля: TeacherID (PK), PassportNumber (AK), LastName, FirstName, MiddleName, WorkExperience, Position, Phone, Email, DepartmentID (FK).
3. **Кафедры:**
   * Поля: DepartmentID (PK), DepartmentName (AK).
4. **Группы:**
   * Поля: GroupID (PK), GroupName (AK), EnrollmentYear, SpecialtyID (FK).
5. **Стипендии:**
   * Поля: ScholarshipID (PK), Type (AK), Amount.
6. **Специальности:**
   * Поля: SpecialtyID (PK), SpecialtyName (AK), Code, DurationYears.
7. **Дисциплины:**
   * Поля: DisciplineID (PK), DisciplineName (AK), Description, ExamType, Hours.
8. **Связи многие ко многим:**
   * Преподаватели\_Дисциплины (TeacherID (FK), DisciplineID (FK)).
   * Дисциплины\_Специальности (DisciplineID (FK), SpecialtyID (FK)).
   * Студенты\_Стипендии (StudentID (FK), ScholarshipID (FK) , StartDate, EndDate).

**Запросы в реляционной алгебре:**

1. Получить список студентов, обучающихся в заданной группе:

πStudentID,LastName,FirstName(σGroupID=G1(Students))

1. Получить преподавателей, работающих на заданной кафедре:

πTeacherID,LastName,FirstName(σDepartmentID=D1(Teachers))

1. Получить список студентов, получающих заданный тип стипендии:

πStudentID,LastName,FirstName​(σType=′academic′​(Student⋈Student\_Scholarship⋈Scholarship))

1. Получить список дисциплин для заданной специальности:

πDisciplineName​(Discipline⋈Discipline\_Specialty⋈Specialty)

1. Получить количество студентов в каждой группе:

πGroupID,COUNT(StudentID)(Student)

**SQL код для создания БД:**

-- Таблица Departments

CREATE TABLE IF NOT EXISTS Departments (

DepartmentID SERIAL PRIMARY KEY,

DepartmentName VARCHAR(100) NOT NULL UNIQUE

);

-- Таблица Specialties

CREATE TABLE IF NOT EXISTS Specialties (

SpecialtyID SERIAL PRIMARY KEY,

SpecialtyName VARCHAR(100) NOT NULL UNIQUE,

Code VARCHAR(10) NOT NULL,

DurationYears INT NOT NULL

);

-- Таблица Groups

CREATE TABLE IF NOT EXISTS Groups (

GroupID SERIAL PRIMARY KEY,

GroupName VARCHAR(50) NOT NULL UNIQUE,

EnrollmentYear INT NOT NULL,

SpecialtyID INT NOT NULL,

CONSTRAINT FK\_Group\_Specialty FOREIGN KEY (SpecialtyID) REFERENCES Specialties(SpecialtyID) ON UPDATE CASCADE

);

-- Таблица Students

CREATE TABLE IF NOT EXISTS Students (

StudentID SERIAL PRIMARY KEY,

PassportNumber VARCHAR(20) NOT NULL UNIQUE,

LastName VARCHAR(50) NOT NULL,

FirstName VARCHAR(50) NOT NULL,

MiddleName VARCHAR(50),

Status VARCHAR(20) NOT NULL,

Phone VARCHAR(15) NOT NULL UNIQUE,

Email VARCHAR(100) NOT NULL UNIQUE,

GroupID INT NOT NULL,

CONSTRAINT FK\_Student\_Group FOREIGN KEY (GroupID) REFERENCES Groups(GroupID) ON UPDATE CASCADE

);

-- Таблица Teachers

CREATE TABLE IF NOT EXISTS Teachers (

TeacherID SERIAL PRIMARY KEY,

PassportNumber VARCHAR(20) NOT NULL UNIQUE,

LastName VARCHAR(50) NOT NULL,

FirstName VARCHAR(50) NOT NULL,

MiddleName VARCHAR(50),

WorkExperience INT NOT NULL,

Position VARCHAR(50) NOT NULL,

Phone VARCHAR(15) NOT NULL UNIQUE,

Email VARCHAR(100) NOT NULL UNIQUE,

DepartmentID INT NOT NULL,

CONSTRAINT FK\_Teacher\_Department FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID) ON UPDATE CASCADE

);

-- Таблица Scholarships

CREATE TABLE IF NOT EXISTS Scholarships (

ScholarshipID SERIAL PRIMARY KEY,

Type VARCHAR(50) NOT NULL UNIQUE,

Amount DECIMAL(10, 2) NOT NULL

);

-- Таблица Disciplines

CREATE TABLE IF NOT EXISTS Disciplines (

DisciplineID SERIAL PRIMARY KEY,

DisciplineName VARCHAR(100) NOT NULL UNIQUE,

Description TEXT NOT NULL UNIQUE,

ExamType VARCHAR(50) NOT NULL,

Hours INT NOT NULL

);

-- Связь многие ко многим: Teachers\_Disciplines

CREATE TABLE IF NOT EXISTS Teachers\_Disciplines (

TeacherID INT NOT NULL,

DisciplineID INT NOT NULL,

PRIMARY KEY (TeacherID, DisciplineID),

CONSTRAINT FK\_TeacherDiscipline\_Teacher FOREIGN KEY (TeacherID) REFERENCES Teachers(TeacherID) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FK\_TeacherDiscipline\_Discipline FOREIGN KEY (DisciplineID) REFERENCES Disciplines(DisciplineID) ON UPDATE CASCADE

);

-- Связь многие ко многим: Disciplines\_Specialties

CREATE TABLE IF NOT EXISTS Disciplines\_Specialties (

DisciplineID INT NOT NULL,

SpecialtyID INT NOT NULL,

PRIMARY KEY (DisciplineID, SpecialtyID),

CONSTRAINT FK\_DisciplineSpecialty\_Discipline FOREIGN KEY (DisciplineID) REFERENCES Disciplines(DisciplineID) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FK\_DisciplineSpecialty\_Specialty FOREIGN KEY (SpecialtyID) REFERENCES Specialties(SpecialtyID) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Связь многие ко многим: Students\_Scholarships

CREATE TABLE IF NOT EXISTS Students\_Scholarships (

StudentID INT NOT NULL,

ScholarshipID INT NOT NULL,

StartDate DATE NOT NULL,

EndDate DATE NOT NULL,

PRIMARY KEY (StudentID, ScholarshipID),

CONSTRAINT FK\_StudentScholarship\_Student FOREIGN KEY (StudentID) REFERENCES Students(StudentID) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FK\_StudentScholarship\_Scholarship FOREIGN KEY (ScholarshipID) REFERENCES Scholarships(ScholarshipID) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT CHK\_ScholarshipDates CHECK (EndDate >= StartDate)

);

-- Получить список студентов, обучающихся в заданной группе, табличная функция

CREATE FUNCTION GetStudentsByGroup(\_GroupID INT) RETURNS TABLE (

StudentID INT,

LastName VARCHAR,

FirstName VARCHAR

) AS $$

BEGIN

RETURN QUERY (

SELECT StudentID, LastName, FirstName

FROM Students

WHERE GroupID = \_GroupID

);

END;

$$ LANGUAGE plpgsql;

-- Получить преподавателей, работающих на заданной кафедре, табличная функция

CREATE FUNCTION GetTeachersByDepartment(\_DepartmentID INT) RETURNS TABLE (

TeacherID INT,

LastName VARCHAR,

FirstName VARCHAR

) AS $$

BEGIN

RETURN QUERY (

SELECT TeacherID, LastName, FirstName

FROM Teachers

WHERE DepartmentID = \_DepartmentID

);

END;

$$ LANGUAGE plpgsql;

-- Представление: студенты с их группами и специальностями

CREATE VIEW StudentsOverview AS

SELECT

s.StudentID,

s.LastName,

s.FirstName,

s.MiddleName,

s.Status,

s.Email,

g.GroupName,

sp.SpecialtyName

FROM Students s

INNER JOIN Groups g ON s.GroupID = g.GroupID

INNER JOIN Specialties sp ON g.SpecialtyID = sp.SpecialtyID;

-- Скалярная функция для подсчёта студентов в группе

CREATE FUNCTION GetStudentCountInGroup(\_GroupID INT) RETURNS INT AS $$

BEGIN

RETURN (

SELECT COUNT(\*)

FROM Students

WHERE GroupID = \_GroupID

);

END;

$$ LANGUAGE plpgsql;

-- Табличная функция для получения студентов по типу стипендии

CREATE FUNCTION GetStudentsByScholarshipType(\_ScholarshipType VARCHAR) RETURNS TABLE (

StudentID INT,

LastName VARCHAR,

FirstName VARCHAR

) AS $$

BEGIN

RETURN QUERY (

SELECT s.StudentID, s.LastName, s.FirstName

FROM Students s

JOIN Students\_Scholarships ss ON s.StudentID = ss.StudentID

JOIN Scholarships sc ON ss.ScholarshipID = sc.ScholarshipID

WHERE sc.Type = \_ScholarshipType

);

END;

$$ LANGUAGE plpgsql;

-- Получить список дисциплин для заданной специальности, представление

CREATE VIEW DisciplinesForSpecialty AS

SELECT d.DisciplineName

FROM Disciplines d

JOIN Disciplines\_Specialties ds ON d.DisciplineID = ds.DisciplineID

JOIN Specialties sp ON ds.SpecialtyID = sp.SpecialtyID;

-- Получить количество студентов в каждой группе

CREATE VIEW StudentCountByGroup AS

SELECT GroupID, COUNT(StudentID) AS StudentCount

FROM Students

GROUP BY GroupID;

-- Хранимая процедура для добавления нового студента

CREATE PROCEDURE AddNewStudent(

\_PassportNumber VARCHAR,

\_LastName VARCHAR,

\_FirstName VARCHAR,

\_MiddleName VARCHAR,

\_Status VARCHAR,

\_Phone VARCHAR,

\_Email VARCHAR,

\_GroupID INT

)

LANGUAGE plpgsql AS $$

BEGIN

INSERT INTO Students (

PassportNumber, LastName, FirstName, MiddleName, Status, Phone, Email, GroupID

) VALUES (

\_PassportNumber, \_LastName, \_FirstName, \_MiddleName, \_Status, \_Phone, \_Email, \_GroupID

);

END;

$$;

**Реализация запросов в виде программируемых объектов:**

1. Получить список студентов, обучающихся в заданной группе: запрос параметризованный (зависит от номера группы), поэтому создаётся табличная функция.

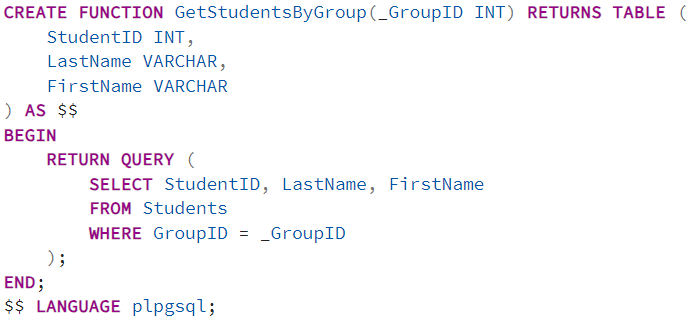


Рисунок 1. Программируемый объект для первого запроса реляционной алгебры.

1. Получить преподавателей, работающих на заданной кафедре: запрос не параметризованный, поэтому создаётся представление.

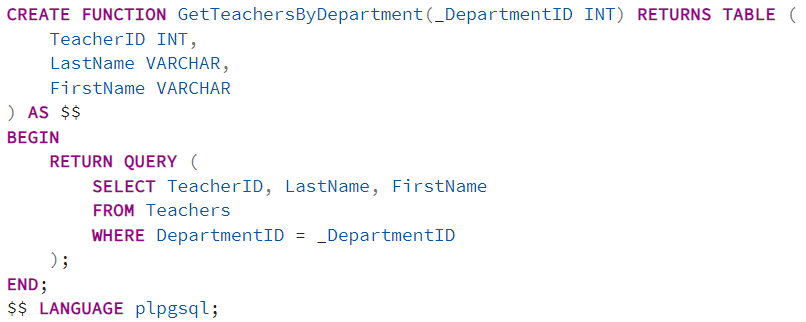


Рисунок 2. Программируемый объект для второго запроса реляционной алгебры.

1. Получить список студентов, получающих заданный тип стипендии: запрос параметризованный (зависит от типа стипендии), поэтому создаётся табличная функция.

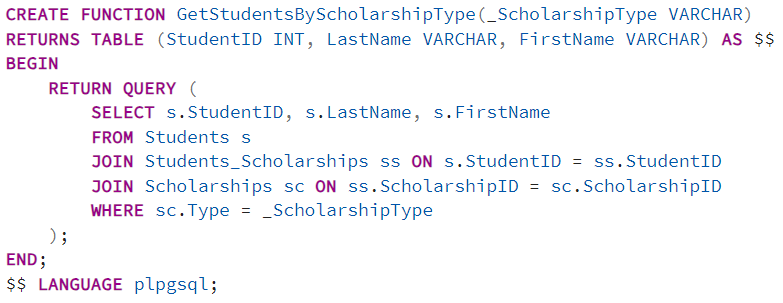


Рисунок 3. Программируемый объект для третьего запроса реляционной алгебры.

1. Получить список дисциплин для заданной специальности: запрос не параметризованный, поэтому создаётся представление.

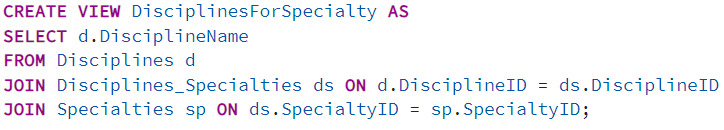


Рисунок 4. Программируемый объект для четвёртого запроса реляционной алгебры.

1. Получить количество студентов в каждой группе: запрос не параметризованный, поэтому создаётся представление:

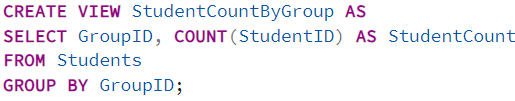


Рисунок 5. Программируемый объект для пятого запроса реляционной алгебры.

**Описание структуры базы данных**

Разработанная база данных представляет собой сложную и продуманную систему, в которой таблицы и их связи организованы таким образом, чтобы удовлетворить потребности управления учебными данными в университете.

Таблица Department хранит информацию о кафедрах, каждой из которых назначается уникальный идентификатор и название. Эта таблица связана с Teacher, что позволяет учитывать принадлежность преподавателей к кафедрам.

Таблица Specialty предоставляет информацию о специальностях, включая их уникальные идентификаторы, названия, коды и продолжительность обучения. Каждая группа, представленная в таблице Group, связана с конкретной специальностью через внешний ключ. В таблице Group также хранится информация о годе набора и уникальном названии группы, что делает её ключевым элементом в структуре данных.

Студенты, учтенные в таблице Student, ассоциируются с группами через внешний ключ. Для каждого студента записаны уникальные данные, включая паспортный номер, ФИО, контактную информацию и статус (например, «Обучается» или «В академическом отпуске»). Таблица также связана с Scholarship через промежуточную таблицу Student\_Scholarship, что позволяет учитывать назначение стипендий.

Таблица Teacher содержит сведения о преподавателях, включая стаж, паспортные данные, должности и принадлежность к кафедрам. Через таблицу Teacher\_Discipline преподаватели связываются с дисциплинами, которые они преподают, а информация о самих дисциплинах представлена в таблице Discipline. Для каждой дисциплины указаны её название, описание, форма сдачи (например, экзамен или зачет) и количество часов.

Таблица Scholarship охватывает различные виды стипендий, с подробностями о типе. Её связь с таблицей Student через Student\_Scholarship помогает отслеживать распределение стипендий среди студентов.

Связи многие ко многим, такие как Teacher\_Discipline, Discipline\_Specialty и Student\_Scholarship, эффективно организуют взаимодействие между ключевыми элементами данных, обеспечивая гибкость и масштабируемость системы.

Программируемые объекты, встроенные в базу данных, значительно упрощают управление данными и их обработку:

1. Представления:

* StudentsOverview: объединяет данные о студентах, их группах и специальностях. Это представление упрощает анализ данных и генерацию отчётов.
* DisciplinesForSpecialty: позволяет просмотреть список дисциплин, относящихся к заданной специальности.
* StudentCountByGroup: показывает количество студентов в каждой группе.

1. Функции:

* GetStudentsByGroup: возвращает список студентов, обучающихся в заданной группе.
* GetTeacherByDepartment: возвращает преподавателей, работающих в определённой кафедре.
* GetStudentCountInGroup: возвращает количество студентов, обучающихся в конкретной группе.
* GetStudentsByScholarshipType: возвращает список студентов, получающих определённый тип стипендии.

1. Хранимая процедура:

* AddNewStudent: автоматизирует процесс добавления нового студента, исключая возможность ошибок при вводе данных.

Далее я приведу две автоматически сгенерированные диаграммы базы данных в нотации Мартина и Чена, средой разработки базы данных PgAdmin4:

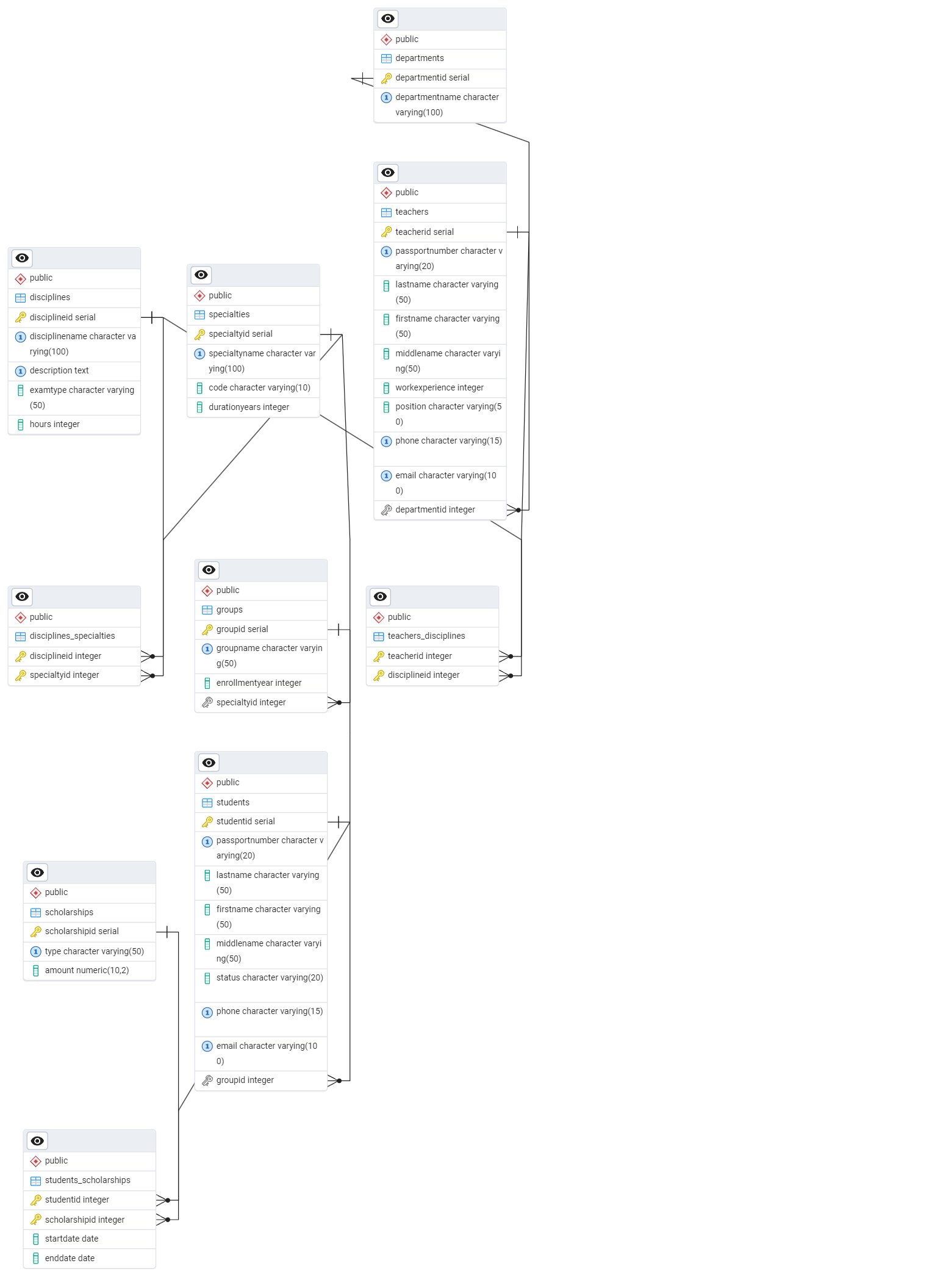


Диаграмма 1. База данных в нотации Мартина.

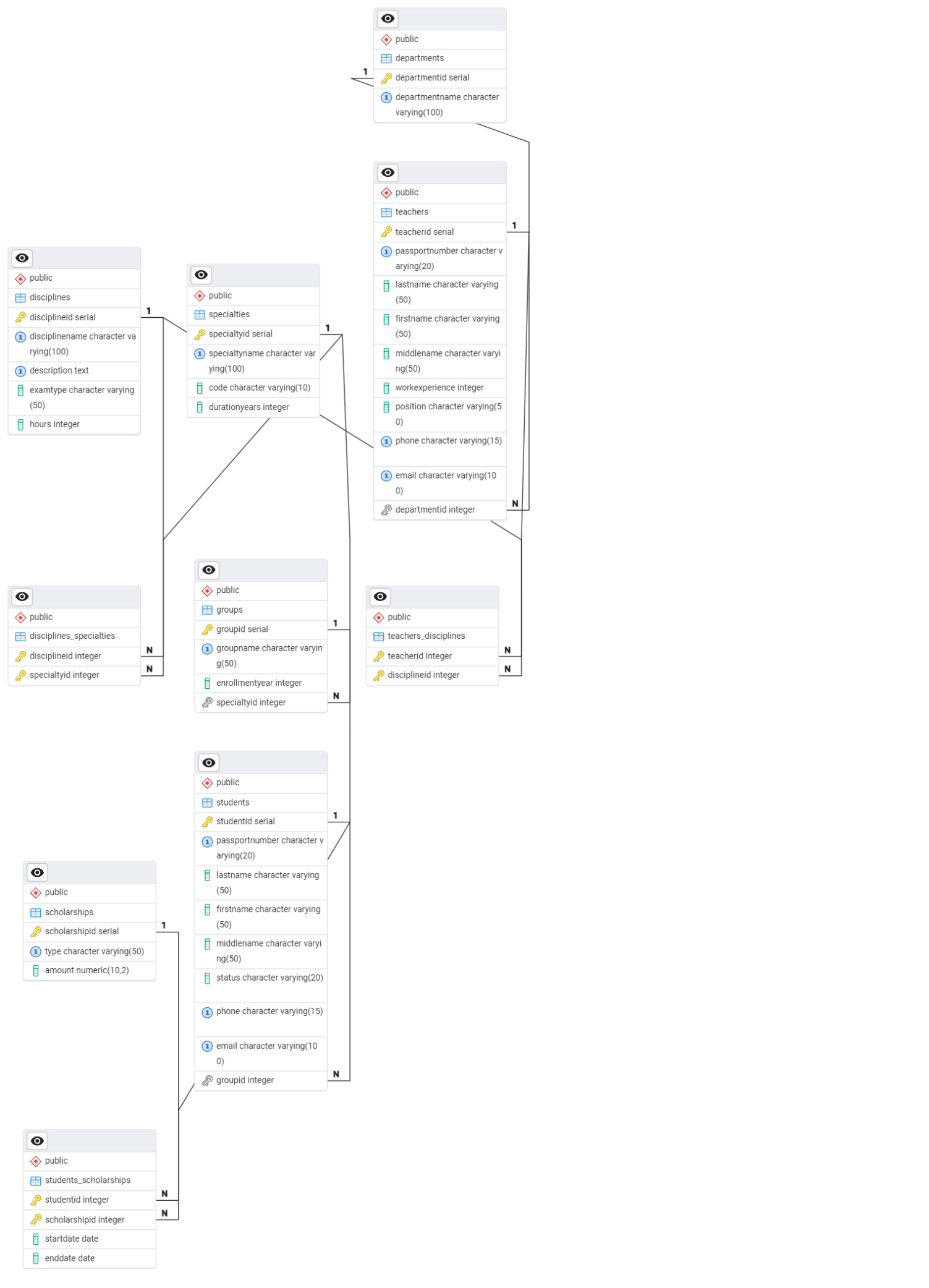


Диаграмма 2. База данных в нотации Чена.

**Выводы**

Созданная база данных полностью соответствует задачам управления учебными данными в вузе. Она организована таким образом, чтобы обеспечить надежное хранение, обработку и анализ данных. Использование первичных и внешних ключей гарантирует целостность данных, а триггеры предотвращают ошибки при их вводе. Программируемые объекты делают систему гибкой и адаптируемой, предоставляя мощные инструменты для анализа и управления.

Система продемонстрировала свою универсальность: она легко масштабируется, позволяя добавлять новые таблицы, связи и функции. Она упрощает рутинные задачи и ускоряет процессы формирования отчетов и принятия решений. Благодаря высокому уровню надежности и гибкости база данных становится важным инструментом для управления учебными процессами, что делает её ценным решением для университетов.