Оглавление

[1. Проектирование базы данных 2](#_Toc11257)

[1.1 Определение таблиц и их атрибутов 2](#_Toc23674)

[1.2 ER – диаграмма проектируемой базы данных 3](#_Toc11382)

[1.3. Нормализация данных и создание таблиц 5](#_Toc12792)

[1.4. SQL – запросы 7](#_Toc13119)

[1.5. Вставьте записи о новом студенте с его личной информацией:ФИО, дата рождения, контактные данные и др. 14](#_Toc25462)

[Приложение А – Связи таблиц базы данных 16](#_Toc14938)

1. **Проектирование базы данных**

Мы будем использовать MySQL для создания базы данных.

## **1.1 Определение таблиц и их атрибутов**

Для учёта студентов, преподавателей, предметов и выставленных оценок нам понадобятся следующие основные таблицы:

1. Students (Студенты)
2. Teachers (Преподаватели)
3. Courses (Предметы/Курсы)
4. Enrollments (Записи о прохождении курсов студентами)
5. Grades (Оценки)
6. Departments (Кафедры/Отделы) *(опционально, для лучшей организации преподавателей и курсов)*

Таблица 2.1 – Таблица Students

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| StudentID | INT AUTO\_INCREMENT | Первичный ключ |
| FirstName | VARCHAR(50) | Имя студента |
| LastName | VARCHAR(50) | Фамилия студента |
| DateOfBirth | DATE | Дата рождения |
| Email | VARCHAR(100) | Электронная почта (уникальная) |
| Phone | VARCHAR(20) | Номер телефона |
| EnrollmentYear | YEAR | Год поступления |

Таблица 2.2 – Таблица Students

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| TeacherID | INT AUTO\_INCREMENT | Первичный ключ |
| FirstName | VARCHAR(50) | Имя преподавателя |
| LastName | VARCHAR(50) | Фамилия преподавателя |
| DateOfBirth | DATE | Дата рождения |
| Email | VARCHAR(100) | Электронная почта (уникальная) |
| Phone | VARCHAR(20) | Номер телефона |
| DepartmentID | INT | Внешний ключ на Departments |

Таблица 2.3 – Таблица Departments *(опционально)*

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| DepartmentID | INT AUTO\_INCREMENT | Первичный ключ |
| DepartmentName | VARCHAR(100) | Название кафедры/отдела |

Таблица 2.4 – Таблица Courses

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| CourseID | INT AUTO\_INCREMENT | Первичный ключ |
| CourseName | VARCHAR(100) | Название курса/предмета |
| DepartmentID | INT | Внешний ключ на Departments |
| Credits | INT | Количество кредитов/часов |

Таблица 2.5 – Таблица Enrollments

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| EnrollmentID | INT AUTO\_INCREMENT | Первичный ключ |
| StudentID | INT | Внешний ключ на Students |
| CourseID | INT | Внешний ключ на Courses |
| Year | YEAR | Год обучения |
| Semester | ENUM('Spring', 'Summer', 'Fall', 'Winter') | Семестр обучения |

Таблица 2.6 – Таблица Grades

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| GradeID | INT AUTO\_INCREMENT | Первичный ключ |
| EnrollmentID | INT | Внешний ключ на Enrollments |
| TeacherID | INT | Внешний ключ на Teachers |
| Grade | TINYINT | Оценка (1-5) |
| GradeDate | DATE | Дата выставления оценки |

## **1.2 ER – диаграмма проектируемой базы данных**

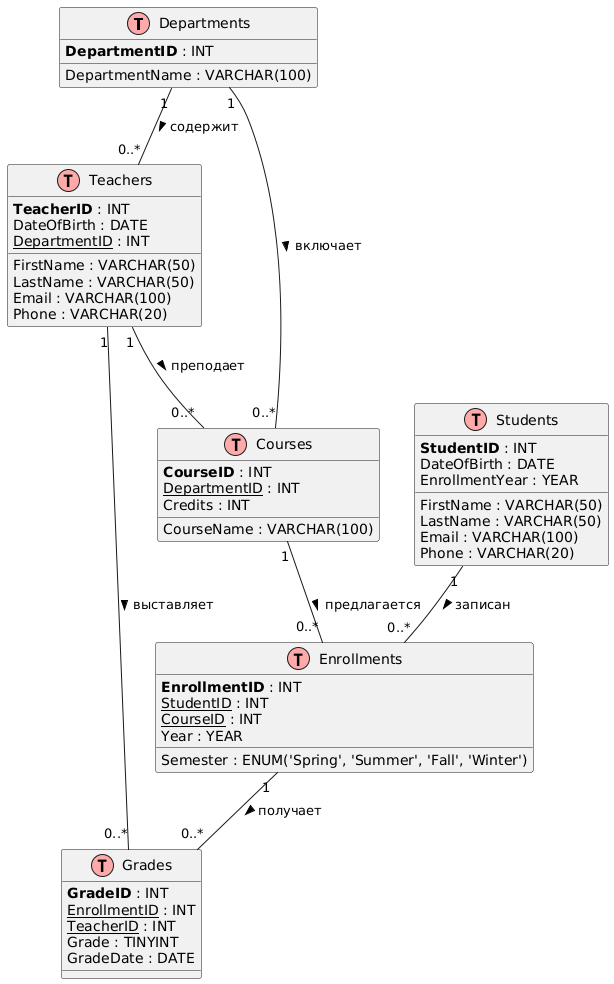


Рисунок 1.2. – ER-диаграмма базы данных

* Студенты могут быть записаны на курсы через регистрацию.
* Teachers преподают Courses и выставляют Grades.
* Departments могут объединять Teachers и Courses.

## **1.3. Нормализация данных и создание таблиц**

CREATE DATABASE UniversityDB CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

USE UniversityDB;

CREATE TABLE Departments (

DepartmentID INT AUTO\_INCREMENT PRIMARY KEY,

DepartmentName VARCHAR(100) NOT NULL UNIQUE

) ENGINE=InnoDB;

CREATE TABLE Teachers (

TeacherID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DateOfBirth DATE,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(20),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

ON DELETE SET NULL

ON UPDATE CASCADE

) ENGINE=InnoDB;

CREATE TABLE Students (

StudentID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

DateOfBirth DATE,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(20),

EnrollmentYear YEAR

) ENGINE=InnoDB;

CREATE TABLE Courses (

CourseID INT AUTO\_INCREMENT PRIMARY KEY,

CourseName VARCHAR(100) NOT NULL UNIQUE,

DepartmentID INT,

Credits INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

ON DELETE SET NULL

ON UPDATE CASCADE

) ENGINE=InnoDB;

CREATE TABLE Enrollments (

EnrollmentID INT AUTO\_INCREMENT PRIMARY KEY,

StudentID INT NOT NULL,

CourseID INT NOT NULL,

Year YEAR NOT NULL,

Semester ENUM('Spring', 'Summer', 'Fall', 'Winter') NOT NULL,

UNIQUE (StudentID, CourseID, Year, Semester),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

ON DELETE CASCADE

ON UPDATE CASCADE

) ENGINE=InnoDB;

CREATE TABLE Grades (

GradeID INT AUTO\_INCREMENT PRIMARY KEY,

EnrollmentID INT NOT NULL,

TeacherID INT NULL, -- Исправлено на NULL

Grade TINYINT NOT NULL CHECK (Grade BETWEEN 1 AND 5),

GradeDate DATE NOT NULL,

FOREIGN KEY (EnrollmentID) REFERENCES Enrollments(EnrollmentID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (TeacherID) REFERENCES Teachers(TeacherID)

ON DELETE SET NULL

ON UPDATE CASCADE

) ENGINE=InnoDB;

);

* 1NF (Первая нормальная форма): все таблицы имеют атомарные (неделимые) значения.
* 2NF (Вторая нормальная форма): все неключевые атрибуты полностью функционально зависят от первичного ключа.
* 3NF (Третья нормальная форма): нет транзитивных зависимостей между неключевыми атрибутами.

**Создание Ограничений на Атрибуты**

Ограничение на Оценки

В таблице Grades уже установлен CHECK для поля Grade, которое ограничивает значения от 1 до 5:

Уникальные Ограничения

* Электронная почта в таблицах Students и Teachers уникальна, чтобы предотвратить дублирование.
* В таблице Enrollments уникально сочетание StudentID, CourseID, Year и Semester, чтобы один студент не был записан на один и тот же курс более одного раза в одном семестре.

**Механизмы для предотвращения дублирования и некорректных данных**

Уникальные Индексы

Как было указано выше, уникальные ограничения для полей, таких как Email и комбинации полей в Enrollments, предотвращают дублирование записей.

Ограничения NOT NULL и CHECK

Использование NOT NULL гарантирует, что обязательные поля не будут пустыми, а CHECK ограничивает допустимые значения для определённых полей.

Триггеры (опционально)

Если требуется более сложная проверка, можно использовать триггеры. Например, чтобы предотвратить удаление преподавателя, если у него есть выставленные оценки.

DELIMITER $$

CREATE TRIGGER before\_teacher\_delete

BEFORE DELETE ON Teachers

FOR EACH ROW

BEGIN

IF EXISTS (SELECT 1 FROM Grades WHERE TeacherID = OLD.TeacherID) THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Нельзя удалить преподавателя с существующими оценками';

END IF;

END$$

DELIMITER ;

* 1. **SQL – запросы**

**Выводить список студентов по определённому предмету**

SELECT

s.StudentID,

CONCAT(s.FirstName, ' ', s.LastName) AS StudentName

FROM

Students s

JOIN

Enrollments e ON s.StudentID = e.StudentID

JOIN

Courses c ON e.CourseID = c.CourseID

WHERE

c.CourseName = 'Алгебра';

****

**Выводить список предметов, которые преподаёт конкретный преподаватель**

SELECT

DISTINCT c.CourseID,

c.CourseName

FROM

Courses c

JOIN

Enrollments e ON c.CourseID = e.CourseID

JOIN

Grades g ON e.EnrollmentID = g.EnrollmentID

WHERE

g.TeacherID = (SELECT TeacherID FROM Teachers WHERE CONCAT(FirstName, ' ', LastName) = 'Алексей Смирнов');



**Выводить средний балл студента по всем предметам**

SELECT

s.StudentID,

CONCAT(s.FirstName, ' ', s.LastName) AS StudentName,

AVG(g.Grade) AS AverageGrade

FROM

Students s

JOIN

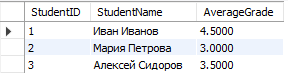
Enrollments e ON s.StudentID = e.StudentID

JOIN

Grades g ON e.EnrollmentID = g.EnrollmentID

GROUP BY

s.StudentID, s.FirstName, s.LastName;



**Выводить рейтинг преподавателей по средней оценке студентов**

SELECT

t.TeacherID,

CONCAT(t.FirstName, ' ', t.LastName) AS TeacherName,

AVG(g.Grade) AS AverageGrade

FROM

Teachers t

JOIN

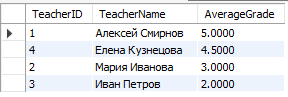
Grades g ON t.TeacherID = g.TeacherID

GROUP BY

t.TeacherID, t.FirstName, t.LastName

ORDER BY

AverageGrade DESC;



**Выводить список преподавателей, которые вели более 3 предметов в прошлом году**

SELECT

t.TeacherID,

CONCAT(t.FirstName, ' ', t.LastName) AS TeacherName,

COUNT(DISTINCT c.CourseID) AS CoursesTaught

FROM

Teachers t

JOIN

Grades g ON t.TeacherID = g.TeacherID

JOIN

Enrollments e ON g.EnrollmentID = e.EnrollmentID

JOIN

Courses c ON e.CourseID = c.CourseID

WHERE

e.Year = YEAR(CURDATE()) - 1

GROUP BY

t.TeacherID, t.FirstName, t.LastName

HAVING

COUNT(DISTINCT c.CourseID) > 3;

****

**Вывести список студентов, которые имеют средний балл выше 4 по математическим предметам, но ниже 3 по гуманитарным**

SELECT

s.StudentID,

CONCAT(s.FirstName, ' ', s.LastName) AS StudentName

FROM

Students s

JOIN

Enrollments e ON s.StudentID = e.StudentID

JOIN

Courses c ON e.CourseID = c.CourseID

JOIN

Grades g ON e.EnrollmentID = g.EnrollmentID

WHERE

c.DepartmentID = (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'Алгебра')

GROUP BY

s.StudentID, s.FirstName, s.LastName

HAVING

AVG(g.Grade) > 4

AND

s.StudentID IN (

SELECT

s2.StudentID

FROM

Students s2

JOIN

Enrollments e2 ON s2.StudentID = e2.StudentID

JOIN

Courses c2 ON e2.CourseID = c2.CourseID

JOIN

Grades g2 ON e2.EnrollmentID = g2.EnrollmentID

WHERE

c2.DepartmentID = (SELECT DepartmentID FROM Departments WHERE DepartmentName = 'Математика')

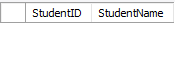
GROUP BY

s2.StudentID

HAVING

AVG(g2.Grade) < 3

**);**

****

**Определить предметы, по которым больше всего двоек в текущем семестре**

SELECT

c.CourseID,

c.CourseName,

COUNT(g.Grade) AS NumberOfTwos

FROM

Courses c

JOIN

Enrollments e ON c.CourseID = e.CourseID

JOIN

Grades g ON e.EnrollmentID = g.EnrollmentID

WHERE

g.Grade = 2

AND e.Year = YEAR(CURDATE())

AND e.Semester = 'ТЕКУЩИЙ\_СЕМЕСТР'

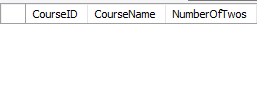
GROUP BY

c.CourseID, c.CourseName

ORDER BY

NumberOfTwos DESC

LIMIT 1;



**Выводить студентов, получивших высший балл по всем своим экзаменам, и преподавателей, которые вели эти предметы**

**-**- Студенты, которые получили высший балл по всем экзаменам

SELECT

s.StudentID,

CONCAT(s.FirstName, ' ', s.LastName) AS StudentName

FROM

Students s

JOIN

Enrollments e ON s.StudentID = e.StudentID

JOIN

Grades g ON e.EnrollmentID = g.EnrollmentID

GROUP BY

s.StudentID, s.FirstName, s.LastName

HAVING

MIN(g.Grade) = 5;

-- Преподаватели, которые вели эти предметы

SELECT DISTINCT

t.TeacherID,

CONCAT(t.FirstName, ' ', t.LastName) AS TeacherName

FROM

Teachers t

JOIN

Grades g ON t.TeacherID = g.TeacherID

JOIN

Enrollments e ON g.EnrollmentID = e.EnrollmentID

JOIN

Students s ON e.StudentID = s.StudentID

WHERE

s.StudentID IN (

SELECT

s2.StudentID

FROM

Students s2

JOIN

Enrollments e2 ON s2.StudentID = e2.StudentID

JOIN

Grades g2 ON e2.EnrollmentID = g2.EnrollmentID

GROUP BY

s2.StudentID

HAVING

MIN(g2.Grade) = 5

);

**Просматривать изменение среднего балла студента по годам обучения**

SELECT

s.StudentID,

CONCAT(s.FirstName, ' ', s.LastName) AS StudentName,

e.Year,

AVG(g.Grade) AS AverageGrade

FROM

Students s

JOIN

Enrollments e ON s.StudentID = e.StudentID

JOIN

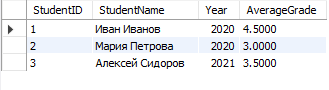
Grades g ON e.EnrollmentID = g.EnrollmentID

GROUP BY

s.StudentID, s.FirstName, s.LastName, e.Year

ORDER BY

s.StudentID, e.Year;



**Определить группы, в которых средний балл выше, чем в других, по аналогичным предметам**

SELECT

e.Semester,

c.CourseID,

c.CourseName,

AVG(g.Grade) AS AverageGrade,

RANK() OVER (PARTITION BY c.CourseID ORDER BY AVG(g.Grade) DESC) AS Rank

FROM

Enrollments e

JOIN

Courses c ON e.CourseID = c.CourseID

JOIN

Grades g ON e.EnrollmentID = g.EnrollmentID

GROUP BY

e.Semester, c.CourseID, c.CourseName

HAVING

AVG(g.Grade) > (

SELECT

AVG(g2.Grade)

FROM

Enrollments e2

JOIN

Grades g2 ON e2.EnrollmentID = g2.EnrollmentID

WHERE

e2.CourseID = c.CourseID

AND e2.Semester = e.Semester

)

ORDER BY

c.CourseID, AverageGrade DESC;

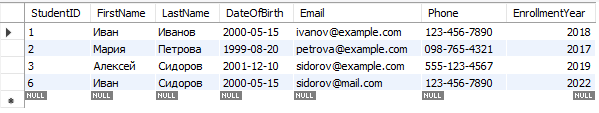
**1.5. Вставьте записи о новом студенте с его личной информацией:ФИО, дата рождения, контактные данные и др.**

**Вставка записи о новом студенте**

SELECT \* FROM universitydb.students;

INSERT INTO Students (FirstName, LastName, DateOfBirth, Email, Phone, EnrollmentYear)

VALUES ('Иван', 'Сидоров', '2000-05-15', 'sidorov@mail.com', '123-456-7890', 2022);



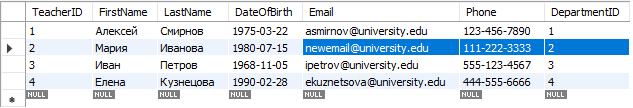
**Обновление контактной информации преподавателя**

Пример: обновить email и телефон преподавателя с TeacherID = 2.

UPDATE Teachers

SET Email = 'newemail@university.edu', Phone = '111-222-3333'

WHERE TeacherID = 2;



или используя ФИО:

UPDATE Teachers

SET Email = 'newemail@university.edu', Phone = '111-222-3333'

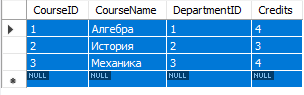
WHERE CONCAT(FirstName, ' ', LastName) = 'Мария Иванова';

**Удаление записи о предмете**

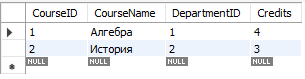
DELETE FROM Courses

WHERE CourseID = 2;

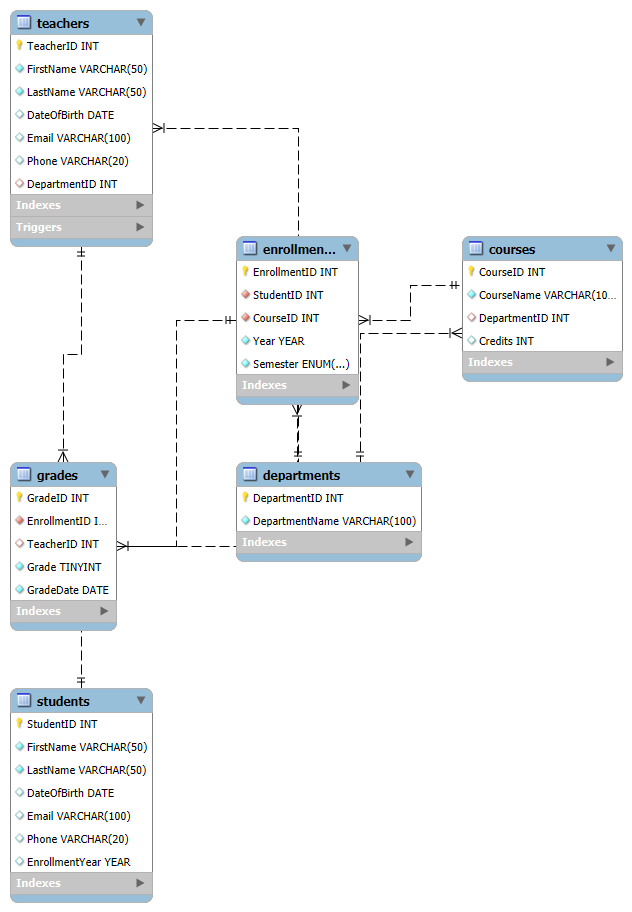
до



после



Приложение А – Связи таблиц базы данных



**1. Departments (Кафедры/Отделы)**

**Описание:** Таблица Departments содержит информацию о различных кафедрах или факультетах учебного заведения. Каждая кафедра может иметь несколько преподавателей и связанных с ней курсов.

**Поля:**

* DepartmentID (INT, ПЕРВИЧНЫЙ КЛЮЧ, АВТО\_ИНКРЕМЕНТ)
* DepartmentName (VARCHAR(100), УНИКАЛЬНЫЙ, НЕ NULL)

**Связи:**

* **Отделы → Учителя:** **один ко многим (1:N)**
  + **Описание:** Одна кафедра может иметь множество преподавателей. Связь устанавливается через внешний ключ DepartmentID в таблице Teachers, который ссылается на Departments.DepartmentID.
* **Отделы → Курсы:** **один ко многим (1:N)**
  + **Описание:** одна кафедра может предлагать несколько курсов. Связь устанавливается через внешний ключ DepartmentID в таблице Courses, который ссылается на Departments.DepartmentID.

**2. Teachers (Преподаватели)**

**Описание:** Таблица Teachers содержит информацию о преподавателях, работающих в учебном заведении. Каждый преподаватель относится к одной кафедре и может вести несколько курсов, а также выставлять оценки студентам.

**Поля:**

* TeacherID (INT, ПЕРВИЧНЫЙ КЛЮЧ, АВТО\_ИНКРЕМЕНТ)
* FirstName (VARCHAR(50), НЕ NULL)
* LastName (VARCHAR(50), НЕ NULL)
* DateOfBirth (ДАТА)
* Email (VARCHAR(100), УНИКАЛЬНЫЙ)
* Phone (VARCHAR(20))
* DepartmentID (INT, ВНЕШНИЙ КЛЮЧ, ОБНУЛЯЕМЫЙ)

**Связи:**

* **Многие-ко-многим (M** **:N)Учителя → Курсы:** (реализовано через дополнительную таблицу TeacherCourses)
  + **Описание:** Один преподаватель может вести несколько курсов, и один курс может вестись разными преподавателями. Для явного представления этой связи создается вспомогательная таблица TeacherCourses.
* **Учителя → Классы:** **один ко многим (1:N)**
  + **Описание:** Один преподаватель может выставлять множество оценок. Связь устанавливается через внешний ключ TeacherID в таблице Grades, который ссылается на Teachers.TeacherID.

**3. Students (Студенты)**

**Описание:** Таблица Students содержит информацию о студентах, включая личные данные и год поступления. Каждый студент может быть записан на несколько курсов через таблицу Enrollments и получать оценки за эти курсы через таблицу Grades.

**Поля:**

* StudentID (INT, ПЕРВИЧНЫЙ КЛЮЧ, АВТО\_ИНКРЕМЕНТ)
* FirstName (VARCHAR(50), НЕ NULL)
* LastName (VARCHAR(50), НЕ NULL)
* DateOfBirth (ДАТА)
* Email (VARCHAR(100), УНИКАЛЬНЫЙ)
* Phone (VARCHAR(20))
* EnrollmentYear (ГОД)

**Связи:**

* **Студенты → Зачисления:** **один ко многим (1:N)**
  + **Описание:** Один студент может быть записан на множество курсов через записи в таблице Enrollments. Связь устанавливается через внешний ключ StudentID в таблице Enrollments, который ссылается на Students.StudentID.

**4. Courses (Курсы/Предметы)**

**Описание:** Таблица Courses содержит информацию о курсах или предметах, предлагаемых учебным заведением. Каждый курс относится к одной кафедре и может иметь множество записей о зачислении студентов через таблицу Enrollments.

**Поля:**

* CourseID (INT, ПЕРВИЧНЫЙ КЛЮЧ, АВТО\_ИНКРЕМЕНТ)
* CourseName (VARCHAR(100), УНИКАЛЬНЫЙ, НЕ NULL)
* DepartmentID (INT, ВНЕШНИЙ КЛЮЧ, ОБНУЛЯЕМЫЙ)
* Credits (INT)

**Связи:**

* **Курсы → Зачисления:** **один ко многим (1:N)**
  + **Описание:** Один курс может иметь множество записей о зачислении студентов через таблицу Enrollments. Связь устанавливается через внешний ключ CourseID в таблице Enrollments, который ссылается на Courses.CourseID.
* **Курсы → Отделы:** **многие ко многим (N:1)**
  + **Описание:** Каждый курс принадлежит одной кафедре. Связь устанавливается через внешний ключ DepartmentID в таблице Courses, который ссылается на Departments.DepartmentID.

**5. Enrollments (Записи о Зачислении)**

**Описание:** Таблица Enrollments связывает студентов с курсами, на которые они записаны. Каждая запись представляет зачисление конкретного студента на конкретный курс в определённый год и семестр.

**Поля:**

* EnrollmentID (INT, ПЕРВИЧНЫЙ КЛЮЧ, АВТО\_ИНКРЕМЕНТ)
* StudentID (INT, ВНЕШНИЙ КЛЮЧ, НЕ NULL)
* CourseID (INT, ВНЕШНИЙ КЛЮЧ, НЕ NULL)
* Year (ГОД, НЕ НУЛЕВОЙ)
* Semester (ЦВЕТ('Весна', 'Лето', 'Осень', 'Зима'), НЕ NULL)

**Связи:**

* **Зачисление → Студенты:** **Многие-к-Одному (N:1)**
  + **Описание:** Каждая запись в Enrollments связана с одним студентом через внешний ключ StudentID, который ссылается на Students.StudentID.
* **Зачисления → Курсы:** **Многие-к-Одному (N:1)**
  + **Описание:** каждая запись в Enrollments связана с одним курсом через внешний ключ CourseID, который ссылается на Courses.CourseID.
* **Зачисление → Классы:** **один ко многим (1:N)**
  + **Описание:** каждая запись в Enrollments может иметь множество оценок в таблице Grades. Связь устанавливается через внешний ключ EnrollmentID в таблице Grades, который ссылается на Enrollments.EnrollmentID.

**6. Grades (Оценки)**

**Описание:** Таблица Grades содержит оценки, выставленные преподавателями студентам за конкретные курсы. Каждая оценка связана с записью о зачислении через EnrollmentID и с преподавателем через TeacherID.

**Поля:**

* GradeID (INT, ПЕРВИЧНЫЙ КЛЮЧ, АВТО\_ИНКРЕМЕНТ)
* EnrollmentID (INT, ВНЕШНИЙ КЛЮЧ, НЕ NULL)
* TeacherID (INT, ВНЕШНИЙ КЛЮЧ, ОБНУЛЯЕМЫЙ)
* Grade (TINYINT, НЕ NULL, ПРОВЕРКА (Оценка от 1 до 5))
* GradeDate (ДАТА, НЕ НУЛЕВАЯ)

**Связи:**

* **Оценки → Зачисление:** **Многие-к-Одному (N:1)**
  + **Описание:** каждая оценка связана с одной записью о зачислении через внешний ключ EnrollmentID, который ссылается на Enrollments.EnrollmentID.
* **Оценки → Учителя:** **многие к одному (N:1)**
  + **Описание:** Каждая оценка выставляется одним преподавателем с помощью внешнего ключа TeacherID, который ссылается на Teachers.TeacherID. При удалении преподавателя поле TeacherID устанавливается в NULL (из-за ON DELETE SET NULL).

**7. TeacherCourses (Преподаватели-Курсы) *(Дополнительная таблица для M:N Связи)***

**Описание:** Таблица TeacherCourses явно связывает преподавателей и курсы, которые они ведут, позволяя реализовать связь «многие ко многим» (M:N) между таблицами Teachers и Courses.

**Поля:**

* TeacherID (INT, ВНЕШНИЙ КЛЮЧ, НЕ NULL)
* CourseID (INT, ВНЕШНИЙ КЛЮЧ, НЕ NULL)

**Связи:**

* **Курсы для преподавателей → Преподаватели:** **многие ко многим (N:1)**
  + **Описание:** каждая запись в TeacherCourses связана с одним преподавателем через внешний ключ TeacherID, который ссылается на Teachers.TeacherID.
* **Учительские курсы → Курсы:** **многие ко многим (N:1)**
  + **Описание:** каждая запись в TeacherCourses связана с одним курсом через внешний ключ CourseID, который ссылается на Courses.CourseID.

**Поля с Композитным Первичным Ключом:**

* PRIMARY KEY (TeacherID, CourseID)

**Пояснение:**

* Таблица TeacherCourses не содержит собственного первичного ключа, а использует сочетание TeacherID и CourseID в качестве составного первичного ключа, что гарантирует уникальность каждой пары преподаватель-курс.

**Общее Описание Связей**

1. **Кафедры ↔ Преподаватели (1:N)**
   * **Описание:** Одна кафедра может иметь множество

преподавателей. Каждый преподаватель принадлежит только одной кафедре.

1. **Факультеты ↔ Курсы (1:N)**
   * **Описание:** Одна кафедра может предлагать множество курсов. Каждый курс принадлежит только одной кафедре.
2. **Курсы для преподавателей (M:N)**
   * **Описание:** Один преподаватель может вести множество курсов, и один курс может вестись множеством преподавателей. Это реализовано с помощью таблицы TeacherCourses.
3. **Учителя ↔ Оценки (1:N)**
   * **Описание:** Один преподаватель может выставить множество оценок. Каждая оценка выставляется только одним преподавателем.
4. **Студенты ↔ Количество зачисленных (1:N)**
   * **Описание:** Один студент может быть зачислен на множество курсов с помощью записей в таблице Enrollments. Каждое зачисление относится только к одному студенту.
5. **Курсы ↔ Количество зачисленных (1:N)**
   * **Описание:** Один курс может иметь множество зачётов. Каждый зачёт относится только к одному курсу.
6. **Зачисление ↔ Оценки (1:N)**
   * **Описание:** Одно зачисление может иметь множество оценок. Каждая оценка относится только к одному зачислению.
7. **УчительКурсы ↔ Учителя и курсы (M:N)**
   * **Описание:** Таблица TeacherCourses связывает преподавателей с курсами, которые они ведут, обеспечивая связь «многие ко многим» между Teachers и Courses