Тема 13. Проектирование БД, связи между таблицами, нормальные формы.

Цель занятия:

Ознакомиться с концепциями работы с таблицами в базах данных, с видами связей между таблицами, понятиями первичных и внешних ключей.

Учебные вопросы:

1. Этапы проектирования базы данных.
2. ER-диаграммы.
3. Первичные и внешние ключи.
4. Связи между отношениями
5. Нормализация данных. Нормальные формы.
6. DML-запросы. SELECT-запросы, выборки из одной таблицы.

1. Этапы проектирования базы данных.

Этапы проектирования базы данных:

1. Анализ требований
2. Проектирование концептуальной модели
3. Проектирование логической модели
4. Физическое проектирование

1. Анализ требований:

Цель: понять бизнес-процессы, для которых создается база данных, и определить, какие данные необходимо хранить и обрабатывать.

Действия:

- Общение с заказчиками и пользователями для сбора информации о том, какие задачи должна решать система.
- Определение объема данных, частоты их обновления и основных операций (чтение, запись, обновление).
- Формулирование целей системы и ключевых требований, например, объем данных, скорость обработки запросов, безопасность и масштабируемость.

Результат: четкое понимание, как база данных будет поддерживать бизнес-процессы, какие данные будут обрабатываться и какие требования к ней предъявляются.

2. Проектирование концептуальной модели:

Цель: создание наглядной модели данных, которая отражает ключевые сущности и их связи без учета специфики СУБД.

Действия:

- Определение сущностей (например, "Клиент", "Заказ", "Продукт"), атрибутов (свойства сущностей, такие как имя клиента, дата заказа) и связей между сущностями.
- Создание ER-диаграммы для отображения всех сущностей, их атрибутов и связей между ними.
- Определение типов связей (один к одному, один ко многим, многие ко многим).

Результат: ER-диаграмма, которая является концептуальной моделью базы данных, предоставляющей общее представление о структуре данных.

3. Проектирование логической модели:

Цель: преобразование концептуальной модели в логическую модель, которая уже учитывает требования нормализации и особенности реляционных баз данных.

Действия:

- Преобразование сущностей из ER-диаграммы в таблицы, где каждая сущность становится таблицей, а атрибуты — полями таблицы.
- Применение нормализации для устранения избыточности данных (приведение к 1NF, 2NF, 3NF, при необходимости — BCNF).
- Определение первичных и внешних ключей для таблиц.

Результат: логическая модель базы данных, включающая структуры таблиц, ключи и связи между таблицами.

4. Физическое проектирование:

Цель: реализация логической модели на уровне конкретной СУБД с учетом производительности, безопасности и будущих изменений.

Действия:

- Выбор конкретной СУБД (например, MySQL, PostgreSQL, SQL Server), учитывая требования системы (масштабируемость, стоимость, доступные функции).
- Проектирование таблиц, включая выбор типов данных для каждого поля, с учетом их объема и типа информации (например, INT для чисел, VARCHAR для текста).
- Создание индексов для ускорения поиска и выполнения сложных запросов.
- Определение стратегии управления транзакциями и обеспечения безопасности данных.

Результат: готовая к реализации физическая структура базы данных, которая оптимизирована для выбранной СУБД и учитывает требования по производительности и надежности.

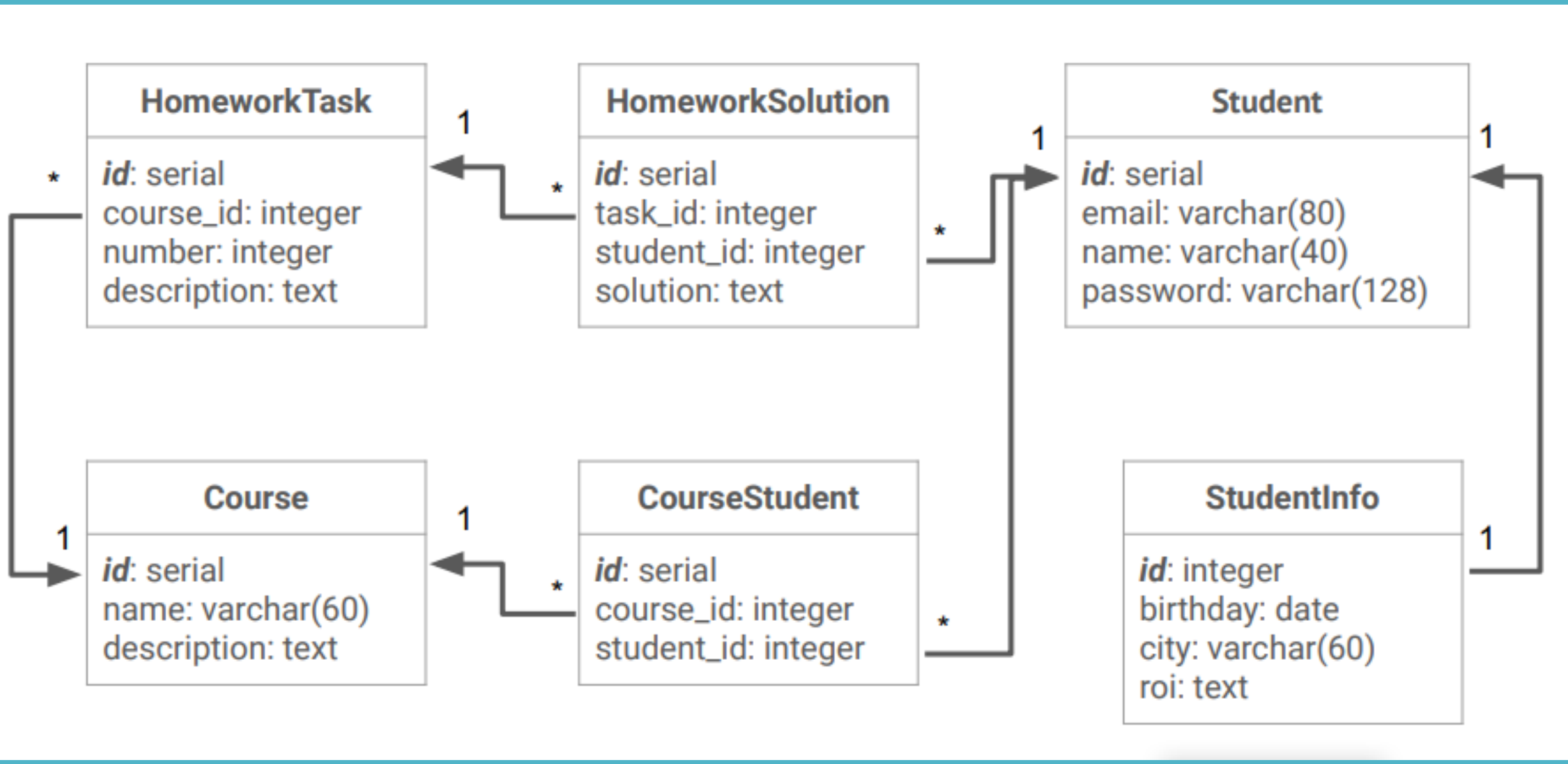
2. ER-диаграммы.

ER-диаграмма (Entity-Relationship диаграмма) — это графическая модель, которая используется для представления данных и взаимосвязей между ними.

Она помогает проектировщикам баз данных наглядно представить структуру данных, их взаимосвязь и основные атрибуты сущностей.

Описывает структуру данных в базе данных, фокусируясь на сущностях (таблицах), их атрибутах (полях) и связях между ними.

Пример. ER-диаграмма.



Основные элементы ER-диаграммы:

Сущности (таблицы): Объекты, для которых необходимо хранить данные. Сущности обычно представлены прямоугольниками. Примеры сущностей: "Студент", "Курс", "Преподаватель".

Атрибуты (поля): Свойства или характеристики сущности, которые содержат данные. Атрибуты находятся внутри прямоугольника класса. Примеры атрибутов для сущности "Студент": "Имя", "Дата рождения", "Номер студенческого билета".

Связи (отношения): Отражают взаимодействие или ассоциации между сущностями. Связи изображаются линиями, которые соединяют сущности. Например, связь между сущностями "Клиент" и "Заказ" может означать, что один клиент может разместить несколько заказов.

Популярные инструменты для построения ER-диаграмм:

- **MySQL Workbench**: Интегрированная среда разработки для MySQL, которая поддерживает создание ER-диаграмм и автоматическое преобразование их в схемы базы данных. Поддерживает реверс-инжиниринг для анализа существующих баз данных.
- **Microsoft Visio**: Мощный инструмент для создания диаграмм, включая ER-диаграммы. Предлагает шаблоны для моделирования баз данных с возможностью настройки сущностей, атрибутов и связей.
- **Lucidchart**: Онлайн-инструмент для создания различных диаграмм, включая ER-диаграммы. Поддерживает совместную работу в реальном времени, шаблоны и удобный интерфейс для моделирования баз данных.
- **Draw.io** (diagrams.net): Бесплатный онлайн-инструмент для создания диаграмм, включая ER-диаграммы. Поддерживает интеграцию с Google Drive, OneDrive и другими облачными сервисами для хранения проектов.
- **dbdiagram.io**: Простое и удобное онлайн-решение для создания ER-диаграмм с поддержкой синтаксиса для описания таблиц и связей. Предлагает экспорт моделей в различные форматы, такие как SQL и PDF.
- **и другие**.

3. Первичные и внешние ключи.

Первичный ключ (Primary Key) — это **один** или **несколько** столбцов в таблице, которые однозначно идентифицируют каждую запись (строку) в этой таблице.

Каждый первичный ключ должен содержать уникальные значения, и ни одно из значений в столбце(ах) первичного ключа не может быть **NULL**.

Зачем лишний атрибут?

name	gpa
Егор	4.82
Егор	4.11
Егор	3.88

Как отличить одного Егора от другого?

Primary key (первичный ключ) на схемах-таблицах обозначается с помощью подчеркивания. На схеме ниже атрибут **id** — это первичный ключ.

id	**name**	**gpa**
1	Егор	4.25
2	Егор	3.82
3	Егор	4.25

Основные характеристики первичного ключа:

Уникальность: Значения первичного ключа должны быть уникальными для каждой строки в таблице, что позволяет однозначно идентифицировать каждую запись.

Не допускается NULL: Первичный ключ не может содержать значения NULL, так как это нарушает принцип уникальности.

Один первичный ключ на таблицу: В каждой таблице может быть только один первичный ключ, который может состоять из одного столбца (простой первичный ключ) или из нескольких столбцов (составной первичный ключ).

Первичные ключи в базе данных бывают двух типов: **простой** и **составной**. Оба типа выполняют одну и ту же основную функцию — однозначно идентифицируют записи в таблице, но различаются по структуре.

- **Простой первичный ключ** — это первичный ключ, состоящий из одного столбца таблицы. Значения в этом столбце уникальны для каждой строки, и они не могут быть NULL.
- **Составной первичный ключ** (Composite Primary Key) — это первичный ключ, который состоит из нескольких столбцов. В совокупности значения этих столбцов должны быть уникальными для каждой строки. Составной первичный ключ используется, когда уникальность строки должна определяться комбинацией нескольких атрибутов.

Простой первичный ключ.

В таблице Employees (Сотрудники) столбец EmployeeID (ID сотрудника) может быть объявлен первичным ключом, так как он содержит уникальный идентификатор для каждого сотрудника.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    HireDate DATE
);
```

Составной первичный ключ.

В таблице OrderDetails (Детали заказа), которая хранит информацию о товарах в заказе, можно использовать составной первичный ключ, состоящий из столбцов OrderID (ID заказа) и ProductID (ID товара), чтобы гарантировать уникальность записи о каждом товаре в каждом заказе.

```
CREATE TABLE OrderDetails (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    PRIMARY KEY (OrderID, ProductID)
);
```

```sql
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(100)
);

CREATE TABLE EmployeeProjects (
    EmployeeID INT,
    ProjectID INT,
    PRIMARY KEY (EmployeeID, ProjectID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
    FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID)
);
```

Пример: база данных системы управления проектами с таблицами Employees (Сотрудники) и Projects (Проекты).

Один сотрудник может участвовать в нескольких проектах, и один проект может включать нескольких сотрудников.

Для реализации такой связи создаётся промежуточная таблица EmployeeProjects.

Primary key

Primary key (первичный ключ) на схемах-таблицах обозначается с помощью подчеркивания. На схеме ниже атрибут **id** – это первичный ключ.

id	name	gpa
1	Егор	4.25
2	Егор	3.82
3	Егор	4.25

Зачем нужен первичный ключ?

- **Идентификация**: Первичный ключ позволяет однозначно идентифицировать каждую запись в таблице, что необходимо для корректного выполнения операций вставки, обновления, удаления и поиска данных.
- **Обеспечение целостности данных**: Наличие первичного ключа предотвращает появление дубликатов в таблице и обеспечивает корректное построение связей между таблицами через внешние ключи.
- **Оптимизация**: Первичные ключи помогают оптимизировать работу базы данных, так как многие системы управления базами данных (СУБД) автоматически создают индексы на первичные ключи, что ускоряет операции поиска.

Внешний ключ (Foreign Key)

Внешний ключ (Foreign Key) — это столбец или набор столбцов в таблице, значения которых ссылаются на первичный ключ (или уникальный ключ) другой таблицы.

Внешний ключ создаёт связь между двумя таблицами, обеспечивая целостность данных и предотвращая несовместимость данных.

Роль внешнего ключа в установлении связей между таблицами:

- **Создание связи между таблицами**: Внешний ключ связывает строки одной таблицы с соответствующими строками другой таблицы. Это позволяет структурировать данные так, чтобы информация, которая относится к одной сущности, была организована и связана с другой сущностью.
- **Поддержание ссылочной целостности**: Внешние ключи гарантируют, что значения в дочерней таблице (таблице, содержащей внешний ключ) всегда соответствуют существующим значениям в родительской таблице (таблице, на которую указывает внешний ключ). Это предотвращает создание записей, которые ссылаются на несуществующие данные, и защищает от ошибок при удалении или обновлении данных.
- **Обеспечение целостности данных**: Внешний ключ помогает поддерживать целостность базы данных. Например, при попытке удаления строки в родительской таблице, которая имеет связанные строки в дочерней таблице, СУБД может запретить это действие или автоматически удалить связанные строки (каскадное удаление).

```sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

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

Пример: база данных, где есть две таблицы: Customers (Клиенты) и Orders (Заказы).
Один клиент может сделать несколько заказов, но каждый заказ относится только к одному клиенту.

Связь "Один-ко-многим"

CustomerID в таблице Orders — это внешний ключ, который ссылается на столбец CustomerID в таблице Customers.

```sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100)
);

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

Пример: система управления университетом, где есть таблицы Students (Студенты) и Courses (Курсы).

Один студент может быть записан на несколько курсов, и один курс может включать несколько студентов.

Для реализации такой связи используется промежуточная таблица StudentCourses.

Связь "Многие-ко-многим"

Таблица StudentCourses содержит два внешних ключа: StudentID, ссылающийся на Students, и CourseID, ссылающийся на Courses.

Ограничения и поддержка целостности данных

Принудительное выполнение ссылочной целостности

Ссылочная целостность — это свойство базы данных, обеспечивающее, что отношения между таблицами остаются логически связными. Она гарантирует, что каждый внешний ключ в дочерней таблице ссылается на существующую запись в родительской таблице.

Система управления базами данных (СУБД) обеспечивает принудительное выполнение ссылочной целостности за счёт использования внешних ключей и связанных с ними ограничений. Вот основные механизмы:

- **Ограничение внешнего ключа (Foreign Key Constraint):**

Это ограничение заставляет базу данных проверять, что любое значение внешнего ключа соответствует значению в первичном ключе (или уникальном ключе) в другой таблице.

Если при вставке или обновлении данных обнаруживается нарушение этого ограничения, операция будет отклонена, и данные не будут сохранены.

- **Обеспечение уникальности данных:**

СУБД гарантирует, что записи в родительской таблице, на которые ссылаются внешние ключи, будут уникальными и неповторимыми.

- **Запрет на "висячие" ссылки:**

Это предотвращает ситуации, когда запись в дочерней таблице ссылается на несуществующую запись в родительской таблице.

Поведение при удалении и обновлении данных (каскадные действия)

При удалении или обновлении записей в родительской таблице, на которые ссылаются записи в дочерней таблице, СУБД может применять различные стратегии для обеспечения целостности данных.

Эти стратегии задаются с помощью каскадных действий (ON DELETE и ON UPDATE) в определении внешнего ключа.

Каскадные действия при удалении:

CASCADE:

При удалении записи в родительской таблице автоматически удаляются все связанные записи в дочерней таблице.

Пример: Если удаляется заказ в таблице Orders, то все связанные позиции в таблице OrderDetails также будут удалены.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE
```

SET NULL:

При удалении записи в родительской таблице соответствующие внешние ключи в дочерней таблице устанавливаются в NULL.

Пример: Если удаляется клиент в таблице Customers, то поле CustomerID в связанных записях в таблице Orders будет установлено в NULL.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE SET NULL
```

SET DEFAULT:

При удалении записи в родительской таблице соответствующие внешние ключи в дочерней таблице устанавливаются в значение по умолчанию, если оно определено.

Пример: Если удаляется запись из таблицы Projects, то связанные записи в таблице EmployeeProjects могут быть установлены в значение по умолчанию.

```
FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID) ON DELETE SET DEFAULT
```

RESTRICT / NO ACTION:

Удаление записи в родительской таблице запрещено, если существуют связанные записи в дочерней таблице.

Пример: Если попытаться удалить клиента, который имеет заказы, операция удаления будет отклонена.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE RESTRICT
```

Каскадные действия при обновлении:

CASCADE:

При изменении значения первичного ключа в родительской таблице автоматически изменяются все соответствующие внешние ключи в дочерней таблице.

Пример: Если обновляется CustomerID в таблице Customers, то все связанные записи в таблице Orders также обновят своё значение CustomerID.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON UPDATE CASCADE
```

SET NULL:

При изменении значения первичного ключа в родительской таблице соответствующие внешние ключи в дочерней таблице устанавливаются в NULL.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON UPDATE SET NULL
```

SET DEFAULT:

При обновлении значения первичного ключа в родительской таблице внешние ключи в дочерней таблице могут быть установлены в значение по умолчанию.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON UPDATE SET DEFAULT
```

RESTRICT / NO ACTION:

Обновление значения первичного ключа в родительской таблице запрещено, если существуют связанные записи в дочерней таблице.

```
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON UPDATE RESTRICT
```

Поддержка целостности данных с помощью внешних ключей и каскадных действий позволяет СУБД гарантировать, что все связи между таблицами остаются корректными и непротиворечивыми.

Это важно для предотвращения ошибок и сохранения целостности данных в реляционной базе данных.

4. Связи между отношениями.

В реляционных базах данных связи между таблицами реализуются через внешние ключи (FOREIGN KEY).

В зависимости от логики данных, связи могут быть следующих типов:

- Один-к-одному (1:1)
- Один-ко-многим (1:M)
- Многие-ко-многим (M:N)

Типы связей:

- **Один к одному**. Каждая сущность A может быть связана только с одной сущностью B, и наоборот. Пример: каждый человек имеет один паспорт, и каждый паспорт принадлежит только одному человеку.
- **Один ко многим.** Одна сущность A может быть связана с несколькими сущностями B, но каждая сущность B может быть связана только с одной сущностью A. Пример: один преподаватель ведет несколько курсов, но каждый курс преподается только одним преподавателем.
- **Многие ко многим.** Несколько сущностей A могут быть связаны с несколькими сущностями B. Пример: студенты могут посещать несколько курсов, и каждый курс может быть посещён несколькими студентами.

Смоделируем ситуацию:

Есть система онлайн обучения.

В ней есть пользователи — студенты. У каждого пользователя есть почта (она же является логином), пароль и имя.

Также у пользователя есть возможность указать дополнительную информацию: дату рождения, город проживания, свои интересы.

Пользователи могут записываться на курсы.

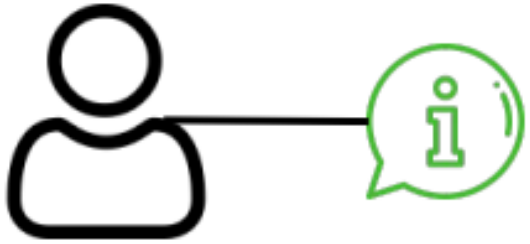
В рамках курса пользователи должны выполнять домашние задания и загружать их в систему.

Связи между отношениями

Типы связей

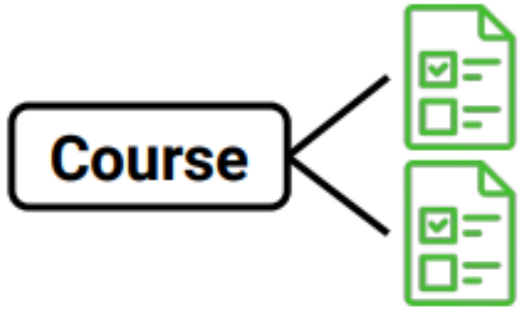
один к одному

пользователь и дополнительная информация о нём



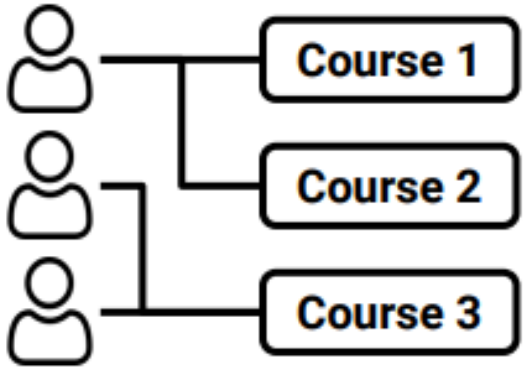
один ко многим

домашние задания на курсе



многие ко многим

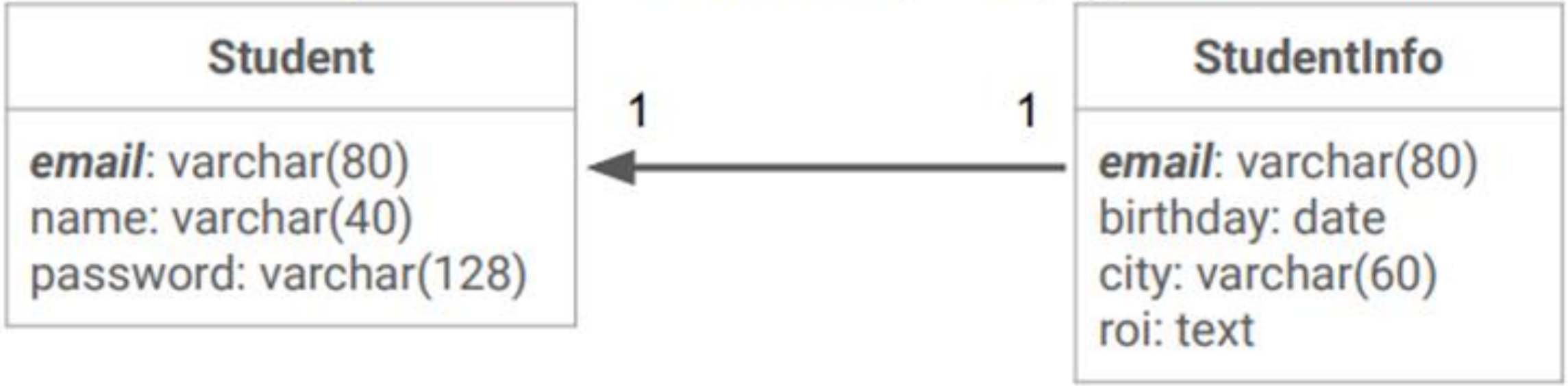
пользователи и курсы



> Связи, как и первичный ключ, можно попросить контролировать СУБД.
> Для этого используется ограничение **foreign key**.

Один к одному. Вариант 1

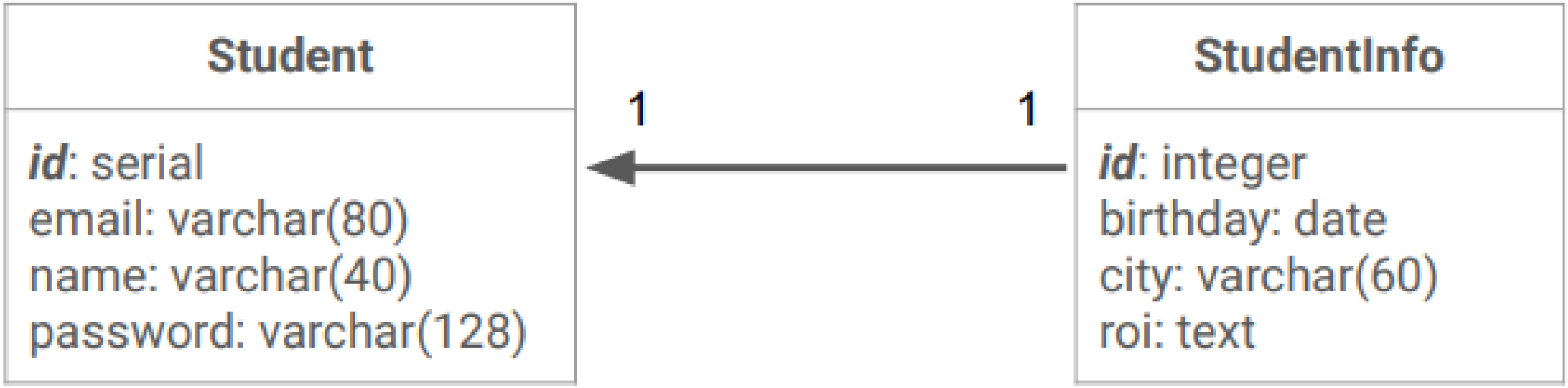
Связываем студента и дополнительную информацию о нём.



```
mysql> CREATE TABLE IF NOT EXISTS StudentInfo (
    ->     email VARCHAR(80) PRIMARY KEY,
    ->     birthday DATE,
    ->     city VARCHAR(60),
    ->     roi TEXT,
    ->     CONSTRAINT fk_student_email FOREIGN KEY (email) REFERENCES Student(email)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> CREATE TABLE IF NOT EXISTS StudentInfo (
    ->     email VARCHAR(80) PRIMARY KEY,
    ->     birthday DATE,
    ->     city VARCHAR(60),
    ->     roi TEXT,
    ->     CONSTRAINT fk_student_email FOREIGN KEY (email) REFERENCES Student(email)
    -> );
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

Один к одному. Вариант 2

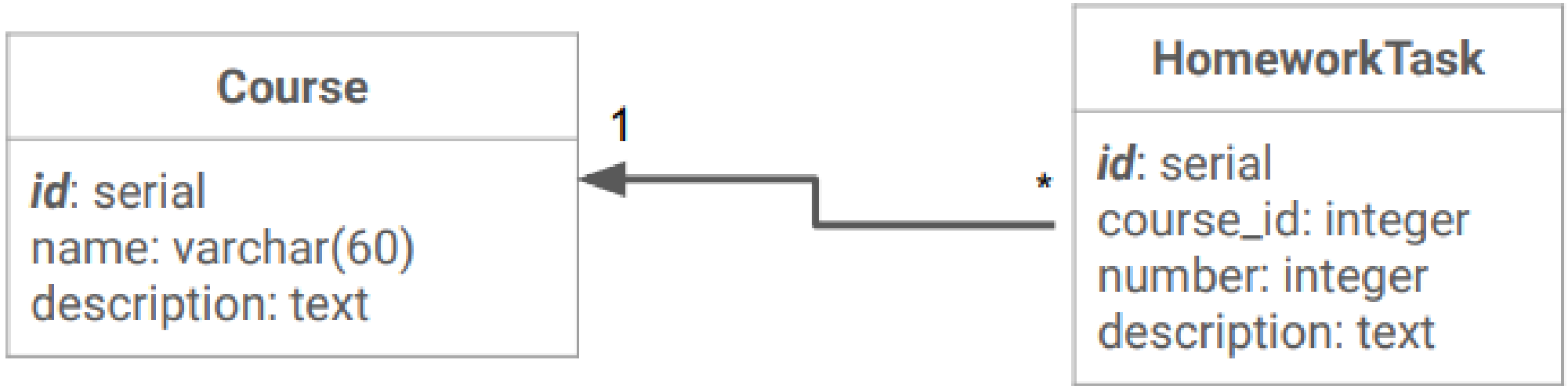


```sql
CREATE TABLE IF NOT EXISTS Student (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(80) UNIQUE NOT NULL,
    name VARCHAR(40) NOT NULL,
    password VARCHAR(128) NOT NULL
);
```

```sql
CREATE TABLE IF NOT EXISTS StudentInfo (
    id INT PRIMARY KEY,
    birthday DATE,
    city VARCHAR(60),
    roi TEXT,
    FOREIGN KEY (id) REFERENCES Student(id) ON DELETE CASCADE
);
```

Один ко многим

Связываем описание домашних заданий с курсами, к которым они относятся.

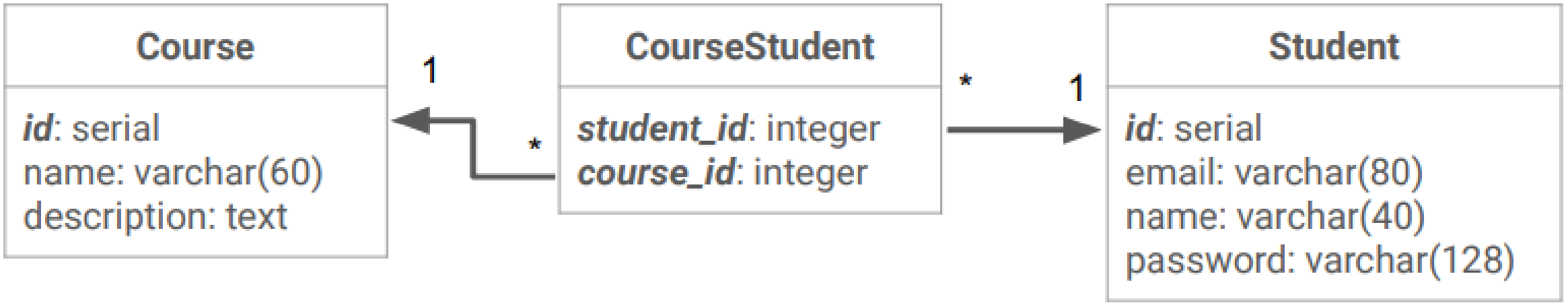


```sql
CREATE TABLE IF NOT EXISTS Course (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(60) NOT NULL,
    description TEXT
);

CREATE TABLE IF NOT EXISTS HomeworkTask (
    id INT AUTO_INCREMENT PRIMARY KEY,
    course_id INT NOT NULL,
    number INT NOT NULL,
    description TEXT NOT NULL,
    FOREIGN KEY (course_id) REFERENCES Course(id) ON DELETE CASCADE
);
```

Многие ко многим. Вариант 1

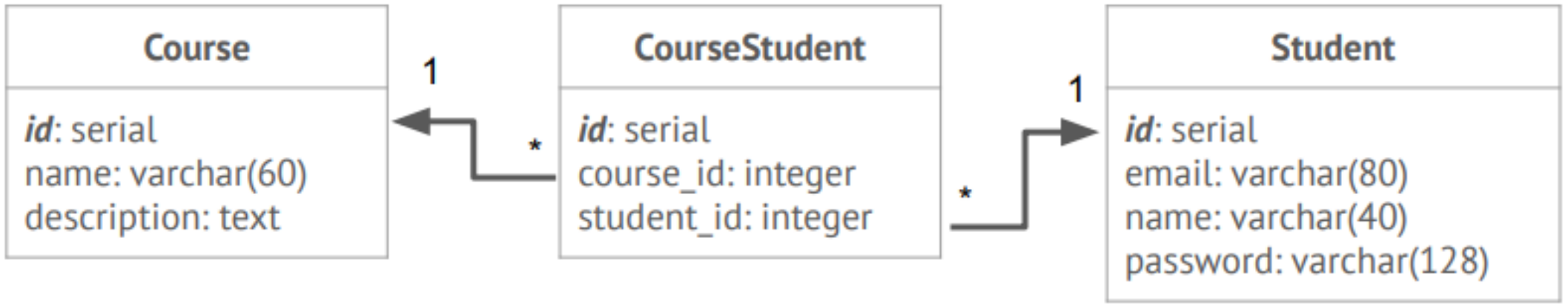
Связываем студентов и курсы, на которые они записаны.



```sql
CREATE TABLE IF NOT EXISTS CourseStudent (
    course_id INT NOT NULL,
    student_id INT NOT NULL,
    PRIMARY KEY (course_id, student_id),
    FOREIGN KEY (course_id) REFERENCES Course(id) ON DELETE CASCADE,
    FOREIGN KEY (student_id) REFERENCES Student(id) ON DELETE CASCADE
);
```

student_id	course_id
1 (Вова)	1 (Python)
1 (Вова)	2 (Java)
2 (Дима)	1 (Python)

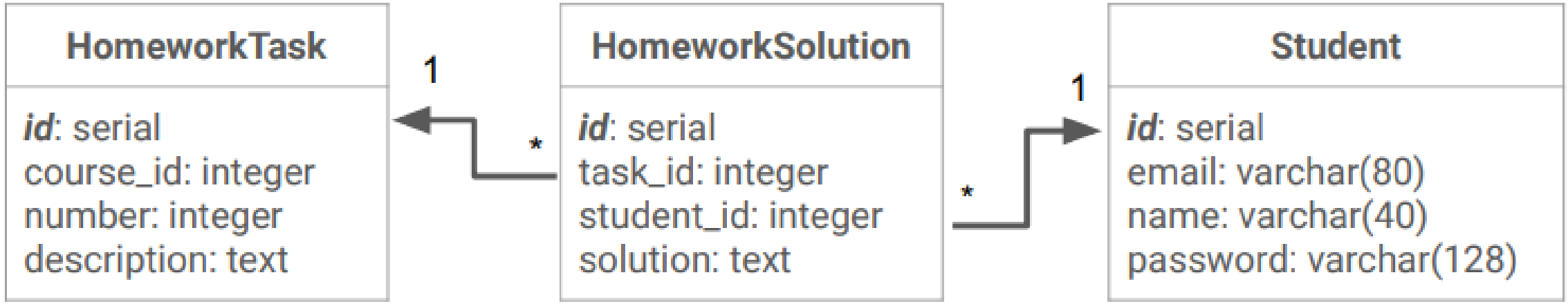
Многие ко многим. Вариант 2




```
CREATE TABLE IF NOT EXISTS CourseStudent (
    id INT AUTO_INCREMENT PRIMARY KEY,
    course_id INT NOT NULL,
    student_id INT NOT NULL,
    FOREIGN KEY (course_id) REFERENCES Course(id) ON DELETE CASCADE,
    FOREIGN KEY (student_id) REFERENCES Student(id) ON DELETE CASCADE
);
```

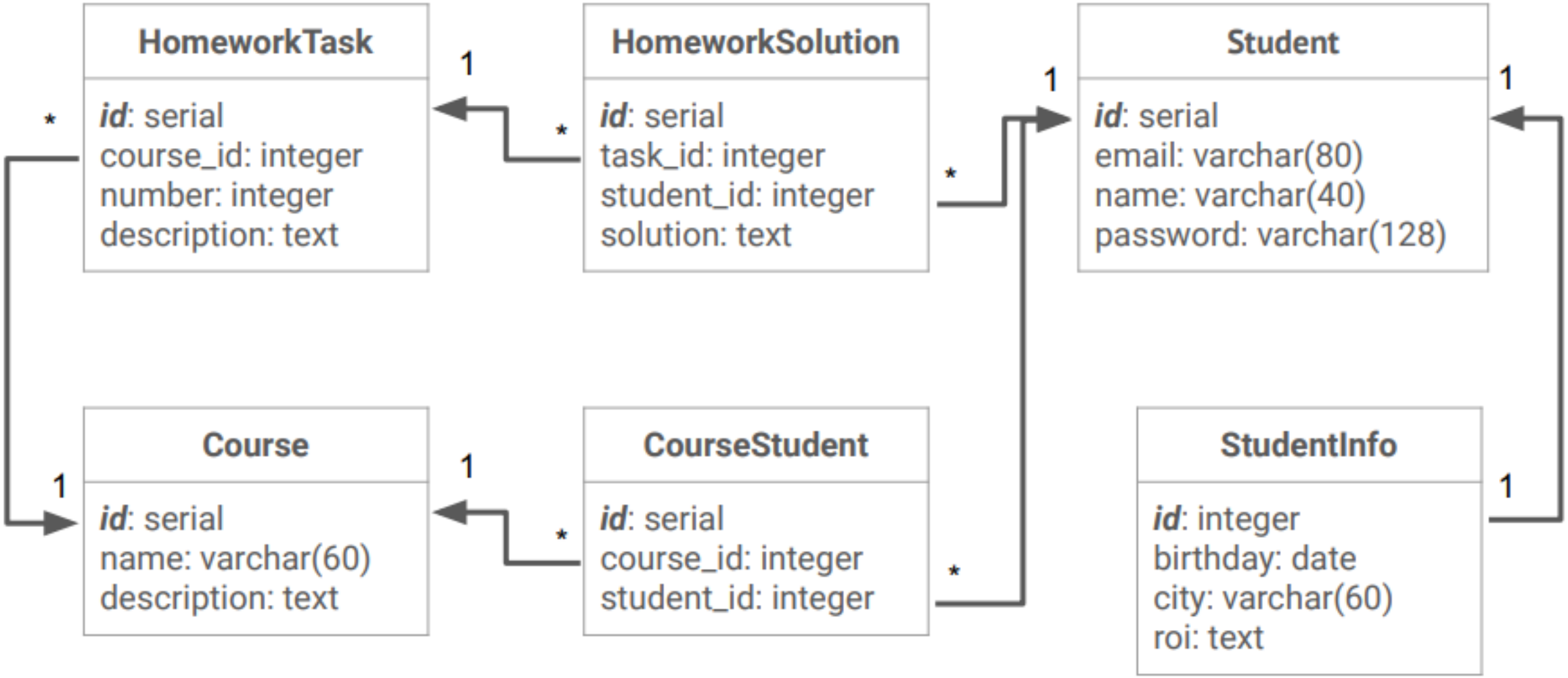
Многие ко многим

Связываем студента и домашние работы, которые он отправил в систему.



```
CREATE TABLE IF NOT EXISTS HomeworkSolution (
    id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    student_id INT NOT NULL,
    solution TEXT NOT NULL,
    FOREIGN KEY (task_id) REFERENCES HomeworkTask(id) ON DELETE CASCADE,
    FOREIGN KEY (student_id) REFERENCES Student(id) ON DELETE CASCADE
);
```

Итоговая схема



5. Нормализация данных.

Нормализация — это процесс организации структуры базы данных, направленный на минимизацию избыточности данных и предотвращение аномалий при обновлении, удалении или добавлении данных.

Нормализация разбивает большие, дублирующиеся таблицы на более маленькие связанные таблицы, что делает базу данных более логичной и управляемой.

Роль нормализации: Нормализация играет ключевую роль в проектировании базы данных, обеспечивая, что структура данных будет соответствовать принципам правильного хранения и организации. Она помогает **предотвратить избыточное хранение данных** и ошибки при обновлении информации, делая базы данных гибкими и согласованными.

Цели нормализации:

- **Устранение избыточности данных.** Избыточные данные занимают лишнее место и могут привести к несогласованным записям. Нормализация помогает избежать дублирования, распределяя данные по нескольким связанным таблицам. Например, вместо хранения информации о клиенте в каждой записи о заказе, создается отдельная таблица клиентов, на которую ссылаются заказы.
- **Обеспечение целостности данных.** Нормализация позволяет поддерживать целостность данных, то есть гарантирует, что данные всегда будут корректными и непротиворечивыми. Это достигается за счет четкой структуры и разделения данных по смысловым категориям. Например, если номер клиента изменяется, достаточно обновить информацию в одной таблице, а не во всех связанных таблицах.
- **Улучшение производительности запросов.** Хотя нормализация изначально направлена на структурную целостность, в некоторых случаях она также может способствовать улучшению производительности запросов, особенно при работе с обновлениями и удалением данных. Меньшие таблицы с четкими связями позволяют быстрее выполнять выборки данных и избегать лишних операций.

Пример избыточных и несогласованных данных:

Customers			
id	**email**	**name**	**city**
1	ivanoff@bk.ru	Иванов	Алматы
2	sidorov@gmail.com	Сидоров	Алма-Ата
3	dtepanof@ya.ru	Степанова	Алмата
4	egorova@list.ru	Егорова	Almaty
5	petrovv@gmail.com	Петров	Астана
6	zuev@bk.com	Зуев	Актау

Пример без избыточных и несогласованных данных:

Customers			
id	**email**	**name**	**city_id**
1	ivanoff@bk.ru	Иванов	1
2	sidorov@gmail.com	Сидоров	1
3	dtepanof@ya.ru	Степанова	1
4	egorova@list.ru	Егорова	1
5	petrovv@gmail.com	Петров	2
6	zuev@bk.com	Зуев	3

Cities	
id	**title**
1	Алматы
2	Астана
3	Актау

Нормальные формы.

1. Первая нормальная форма (1NF) — Гарантирует, что все данные в таблице атомарны (неделимы).
2. Вторая нормальная форма (2NF) — Устраняет частичные зависимости от первичного ключа, обеспечивая, что каждый неключевой атрибут полностью зависит от всего первичного ключа.
3. Третья нормальная форма (3NF) — Устраняет транзитивные зависимости, обеспечивая, что все неключевые атрибуты зависят только от первичного ключа и не зависят друг от друга.
4. Бойс-Кодд нормальная форма (BCNF) — Дополняет 3NF, устраняя случаи, когда определение функциональных зависимостей нарушает целостность данных, хотя она уже находится в 3NF.
5. Четвертая нормальная форма (4NF) — Устраняет многозначные зависимости, обеспечивая, что каждая независимая многозначная зависимость должна находиться в отдельной таблице.
6. Пятая нормальная форма (5NF) — Устраняет случаи, когда таблица может быть разделена на несколько таблиц, не теряя информацию о связях между данными, и предотвращает проблемы с проекцией данных.
7. Шестая нормальная форма (6NF) — Используется для обработки временных зависимостей и изменения данных, которые могут быть разделены по времени.

На практике чаще всего рассматривают первые три нормальные формы. Почему?

- **Практическая применимость.** 1NF, 2NF и 3NF покрывают большинство случаев в проектировании баз данных и решают основные проблемы избыточности и целостности данных, что делает их достаточными для большинства бизнес-приложений.
- **Сложность и избыточность.** BCNF и 4NF и последующие нормальные формы предназначены для более специализированных случаев и могут вводить дополнительную сложность, которая не всегда оправдана в реальных приложениях. Например, 4NF и 5NF часто применяются в теоретических исследованиях или очень сложных системах, где многозначные зависимости и проектирование данных играют ключевую роль.

Первая нормальная форма (1NF).

Определение: Таблица находится в первой нормальной форме, если все её поля содержат только **атомарные** (неделимые) значения и каждая ячейка содержит только одно значение.

Требования:

- Каждое поле таблицы должно содержать только **одно значение** (например, не должно быть списков или наборов значений в одной ячейке).
- Каждая строка таблицы должна быть **уникальной**, что достигается использованием первичного ключа.

Пример. Таблица, содержащая информацию о студентах и их курсах:

Students_Courses		
id	**name**	**courses**
1	Иванов	Python, Java
2	Сидоров	Python, CSharp

Students_Courses		
id	**name**	**courses**
1	Иванов	Python
2	Иванов	Java
3	Сидоров	Python
4	Сидоров	CSharp

Вторая нормальная форма (2NF).

Определение: Таблица находится во второй нормальной форме, если она уже находится в 1NF и все неключевые атрибуты полностью функционально зависят от всего первичного ключа.

Требования:

- Таблица должна быть в 1NF.
- Все атрибуты, не входящие в первичный ключ, должны **зависеть** от всего первичного ключа, а не от его части.

Пример. Таблица с данными о студентах и их факультетах:

Students_faculties			
Student_ID	StudentName	faculty	Dean_of_th_Faculty
1	Иванов	программирование	ДТН Егорова
2	Сидоров	кибербезопасность.	КТН Петров
3	Степанова	робототехника	КТН Зуев

Поле **Dean_of_th_Faculty** (декан факультета) зависит только от **Faculty** (Факультет), а не от первичного ключа. Это означает, что знание **Faculty** позволяет определить **Dean_of_th_Faculty**, независимо от значения StudentID.

Преобразование в 2NF. Для приведения таблицы в 2NF нужно устранить частичные зависимости и разделить данные на несколько таблиц:

Students_faculties		
Student_ID	StudentName	faculty
1	Иванов	программирование
2	Сидоров	кибербезопасность.
3	Степанова	робототехника

faculties	
faculty	Dean_of_th_Faculty
программирова	ДТН Егорова
кибербезопасно	КТН Петров
робототехника	КТН Зуев

Пример. Таблица с данными о заказах:

Orders			
OrderID	**ProductID**	**ProductName**	**Quantity**
1	101	Widget	10
1	102	Gadget	5
2	101	Widget	20

В данной таблице первичный ключ составной, состоящий из двух полей: OrderID и ProductID.

Поле ProductName зависит только от ProductID, а не от всего первичного ключа (OrderID и ProductID).

Это означает, что знание ProductID позволяет определить ProductName, независимо от значения OrderID.

Из-за этой частичной зависимости данные о продукте (например, ProductName) повторяются для каждого заказа, в котором этот продукт появляется.

Если название продукта изменится, его нужно будет обновить в каждой строке таблицы, что может привести к ошибкам и несогласованности.

Чтобы привести таблицу в 2NF, нужно устранить частичные зависимости и разделить данные на несколько таблиц:

Orders		
OrderID	ProductID	Quantity
1	101	10
1	102	5
2	101	20

Products	
ProductID	ProductName
101	Widget
102	Gadget

Третья нормальная форма (3NF).

Определение: Таблица находится в третьей нормальной форме, если она уже находится во 2NF и все неключевые атрибуты не зависят транзитивно от первичного ключа.

Требования:

- Таблица должна быть в 2NF.
- Никакой неключевой атрибут не должен зависеть от другого неключевого атрибута.

Предположим, у нас есть таблица с информацией о сотрудниках и их проектах:

Projects				
EmployeeID	EmployeeName	ProjectID	ProjectName	ProjectManager
1	John Doe	101	Alpha	Sarah Lee
1	John Doe	102	Beta	Mike Brown
2	Jane Smith	101	Alpha	Sarah Lee
3	Alice Johnson	103	Gamma	Emma Davis

Нарушение 3NF: Третья нормальная форма требует, чтобы все неключевые атрибуты зависели только от первичного ключа и не имели транзитивных зависимостей.

В этом примере, ProjectManager транзитивно зависит от EmployeeID, через ProjectID и ProjectName.

Чтобы привести таблицу в 3NF, нужно устранить транзитивные зависимости и разделить данные на несколько таблиц:

EmployeeProjects	
EmployeeID	ProjectID
1	101
1	102
2	101
3	103

Projects		
ProjectID	ProjectName	ProjectManager
101	Alpha	Sarah Lee
102	Beta	Mike Brown
103	Gamma	Emma Davis

Пошаговое преобразование базы данных от 1NF до 3NF на конкретном примере базы данных магазина:

Orders					
OrderID	CustomerName	Product	Quantity	Price	CustomerAddress
1	Иван Иванов	Товар1	2	200	Адрес1
1	Иван Иванов	Товар2	1	100	Адрес1
2	Петр Петров	Товар1	1	200	Адрес2

Приведение к 2NF. 2NF требует устранения частичных зависимостей. Мы разделяем таблицу на две:

Orders		
OrderID	CustomerName	CustomerAddress
1	Иван Иванов	Адрес1
2	Петр Петров	Адрес2

OrderDetails			
OrderID	Product	Quantity	Price
1	Товар1	2	200
1	Товар2	1	100
2	Товар1	1	200

3NF требует устранения транзитивных зависимостей. Добавляем таблицы для клиентов и продуктов:

Customers		
CustomerID	CustomerName	CustomerAddress
1	Иван Иванов	Адрес1
2	Петр Петров	Адрес2

Products		
ProductID	ProductName	Price
1	Товар1	200
2	Товар2	100

Orders	
OrderID	CustomerID
1	1
2	2

OrderDetails		
OrderID	ProductID	Quantity
1	1	2
1	2	1
2	1	1

Домашнее задание:

1. Задача: Построить ER-диаграмму для простой системы управления библиотекой.

Сущности:

- Книга: id (первичный ключ), название, автор, год издания, жанр, количество экземпляров, статус (доступна/выдана).
- Читатель: id (первичный ключ), имя, фамилия, номер телефона, адрес, дата рождения.
- Заказ: id (первичный ключ), дата выдачи, дата возврата, читатель_id (внешний ключ), книга_id (внешний ключ).

Связи:

- Книга и Заказ связаны отношением "один ко многим": одна книга может быть выдана многим читателям.
- Читатель и Заказ связаны отношением "один ко многим": один читатель может взять много книг.

Список литературы:

1. Руководство по MySQL.

Материалы лекций:

https://github.com/ShViktor72/Education2025