

Тема 9.3: ПРОЕКТИРОВАНИЕ БД. СВЯЗИ, НОРМАЛИЗАЦИЯ БД. 3НФ.

Цель занятия:

Ознакомиться с основами проектирования БД.



План занятия:

1. Связи между отношениями:
 - а. Один к одному.
 - б. Один ко многим.
 - с. Многие ко многим.
2. Нормальные формы:
 - а. Первая нормальная форма.
 - б. Вторая нормальная форма.
 - с. Третья нормальная форма.



```
create table if not exists Student (  
    id serial primary key,  
    name varchar(40) not null,  
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5)  
);
```

Атрибут

Кортеж →

id	name	gpa
1	Егор	4.82
2	Егор	4.11
3	Егор	3.88

Пример отношения «Успеваемость студентов»

Проектирование схем БД

Проектирование схем БД

Зачем лишний атрибут?

name	gpa
Егор	4.82
Егор	4.11
Егор	3.88

Как отличить одного Егора от другого?

Primary key

Первичный ключ — это отдельное поле или комбинация полей, которые однозначно определяют запись — кортеж.

```
create table if not exists Student (  
    id serial primary key,  
    name varchar(40) not null,  
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5)  
);
```

```
create table if not exists Student (  
    name varchar(40) primary key,  
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5)  
); # какой недостаток?
```

```
create table if not exists Student (  
    name varchar(40),  
    gpa numeric(3, 2) check (gpa >= 0 and gpa <= 5),  
    constraint student_pk primary key (name, gpa)  
); # здесь все ок?
```

Primary key

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

<u>id</u>	name	gpa
1	Егор	4.25
2	Егор	3.82
3	Егор	4.25

Составной ключ

В реляционных базах данных, составной ключ (composite key) представляет собой комбинацию двух или более атрибутов (полей), которые вместе обеспечивают уникальность каждой записи в таблице.

```
1 CREATE TABLE users(  
2     name VARCHAR(50) NOT NULL,  
3     surname VARCHAR(50) NOT NULL,  
4     CONSTRAINT users_pk PRIMARY KEY(name, surname)  
5 );
```

Связи между отношениями

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

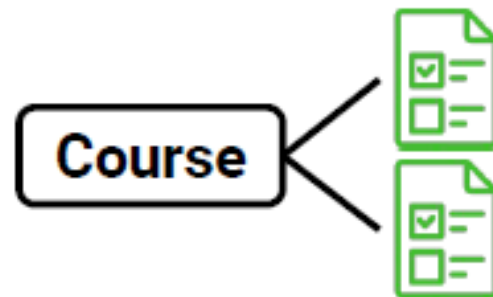
один к одному

пользователь и дополнительная информация о нём



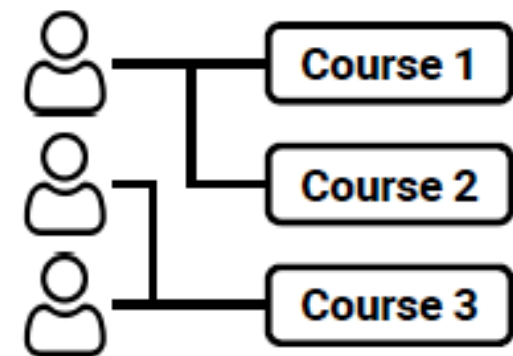
один ко многим

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**.

1.Один к Одному (One-to-One):

1.Один к одному отношение между двумя таблицами означает, что каждая запись в одной таблице связана с одной и только одной записью (не больше) в другой таблице.

2.Один ко Многим (One-to-Many):

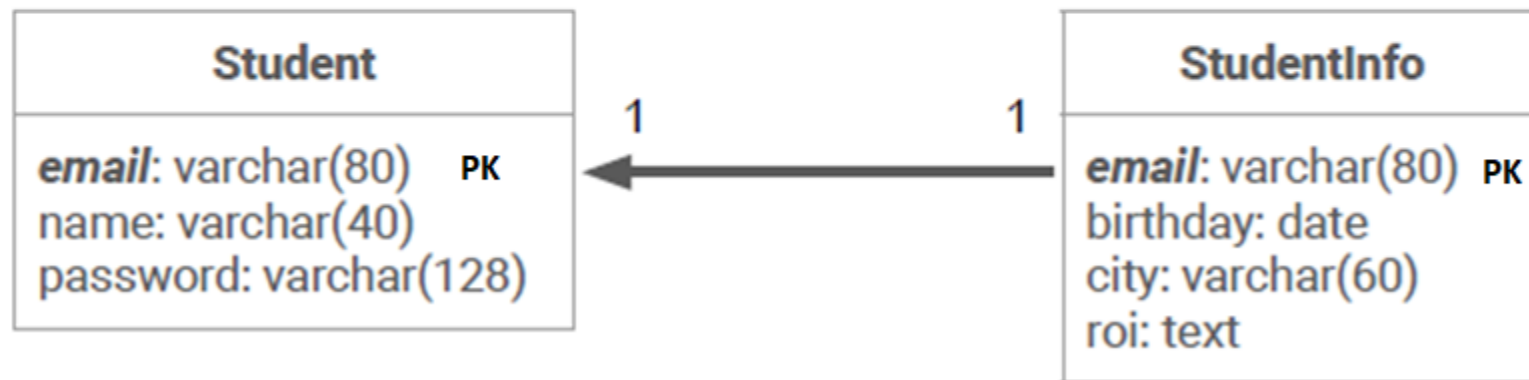
1.Один ко многим отношение означает, что каждая запись в одной таблице может быть связана с несколькими записями в другой таблице, но каждая запись во второй таблице связана только с одной записью в первой таблице.

3.Многие к Многим (Many-to-Many):

1.Многие к многим отношение означает, что множество записей в одной таблице может быть связано с множеством записей в другой таблице. Для реализации таких связей, обычно, требуется дополнительная таблица-связь (таблица-промежуточная).

Один к одному. Вариант 1

Связываем студента и дополнительную информацию о нём.



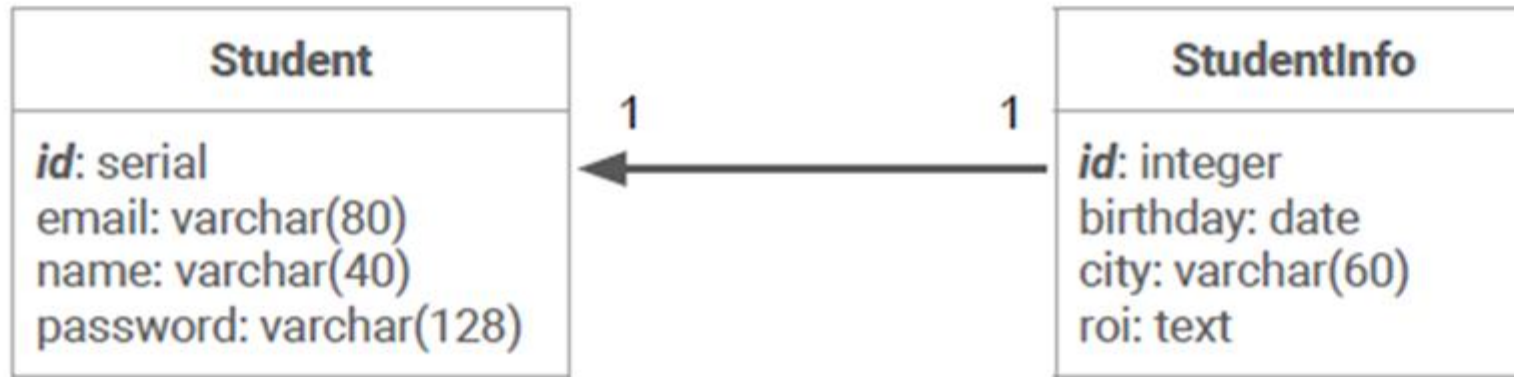
```
create table if not exists Student (  
    email varchar(80) primary key,  
    name varchar(40) not null,  
    password varchar(128) not null  
);
```

```
create table if not exists StudentInfo (  
    email varchar(80) primary key references Student(email),  
    birthday date,  
    city varchar(60),  
    roi text  
);
```

PK - не позволит создать более одной записи

references - не позволит создать информацию о несуществующем студенте

Один к одному. Вариант 2



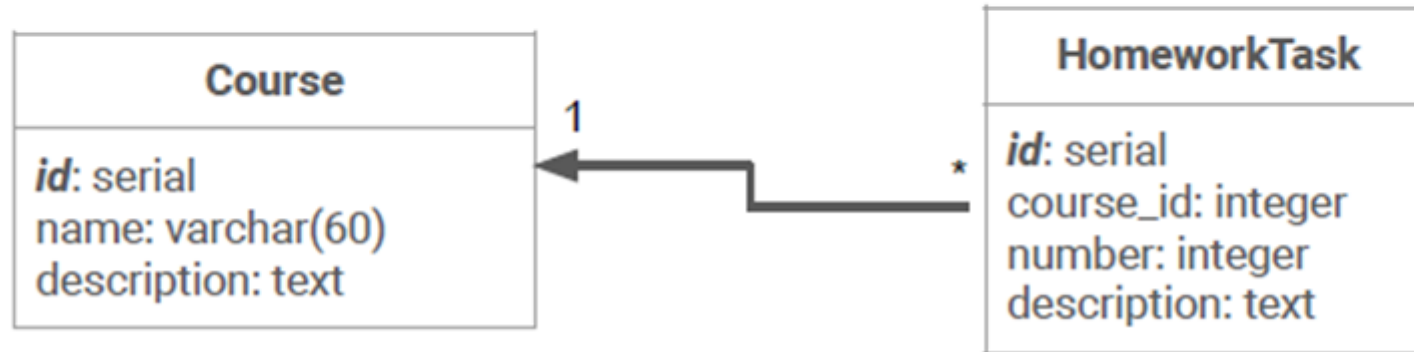
```
create table if not exists Student (  
    id serial primary key,  
    email varchar(80) unique not null,  
    name varchar(40) not null,  
    password varchar(128) not null  
);
```

```
create table if not exists StudentInfo (  
    id integer primary key references Student(id),  
    birthday date,  
    city varchar(60),  
    roi text  
);
```

- 1. экономия памяти
- 2. немного быстрее (вставка)
- 3. меньше проблем, если пользователь сменит email

Один ко многим

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



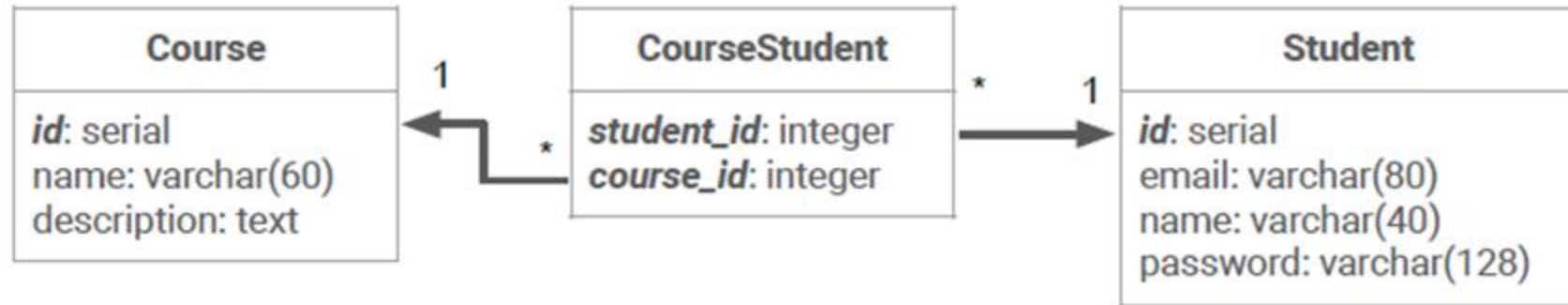
```
create table if not exists Course (  
    id serial primary key,  
    name varchar(60) not null,  
    description text  
);
```

```
create table if not exists HomeworkTask (  
    id serial primary key,  
    course id integer not null references Course(id),  
    number integer not null,  
    description text not null  
);
```

Ограничение **references** проверяет, существует ли такой курс.
А домашних заданий можно создать сколько угодно

Многие ко многим. Вариант 1

Связываем студентов и курсы, на которые они записаны.

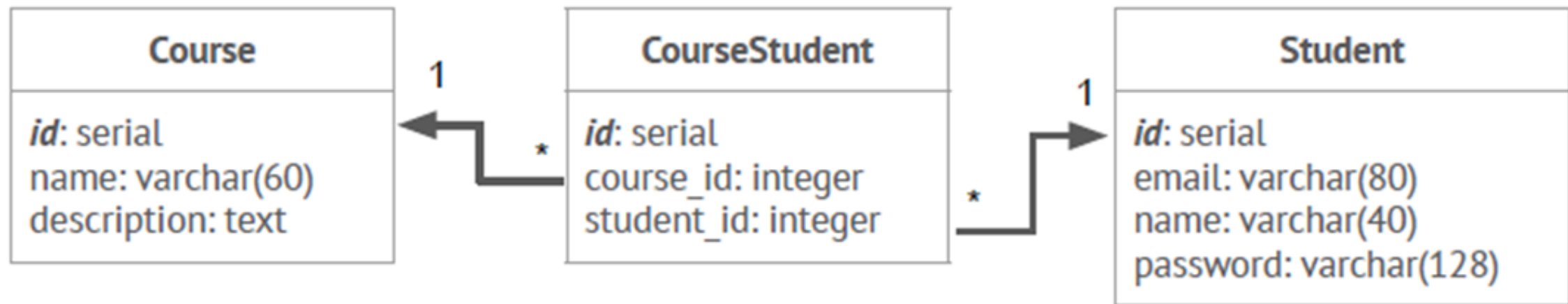


```
create table if not exists CourseStudent (  
    course id integer references Course(id),  
    student id integer references Student(id),  
    constraint pk primary key (course_id, student_id)  
);
```

pk - имя ограничения, уникальное в рамках всей БД (course_student_pk)

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

Многие ко многим. Вариант 2

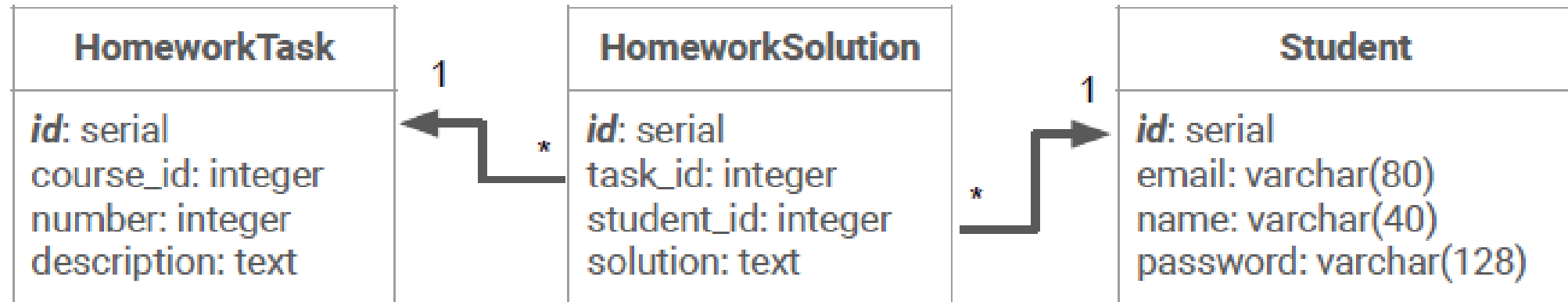


1. дополнительная трата памяти
2. возможность создания дублирующих записей

```
create table if not exists CourseStudent (  
    id serial primary key,  
    course_id integer not null references Course(id),  
    student_id integer not null references Student(id)  
);
```

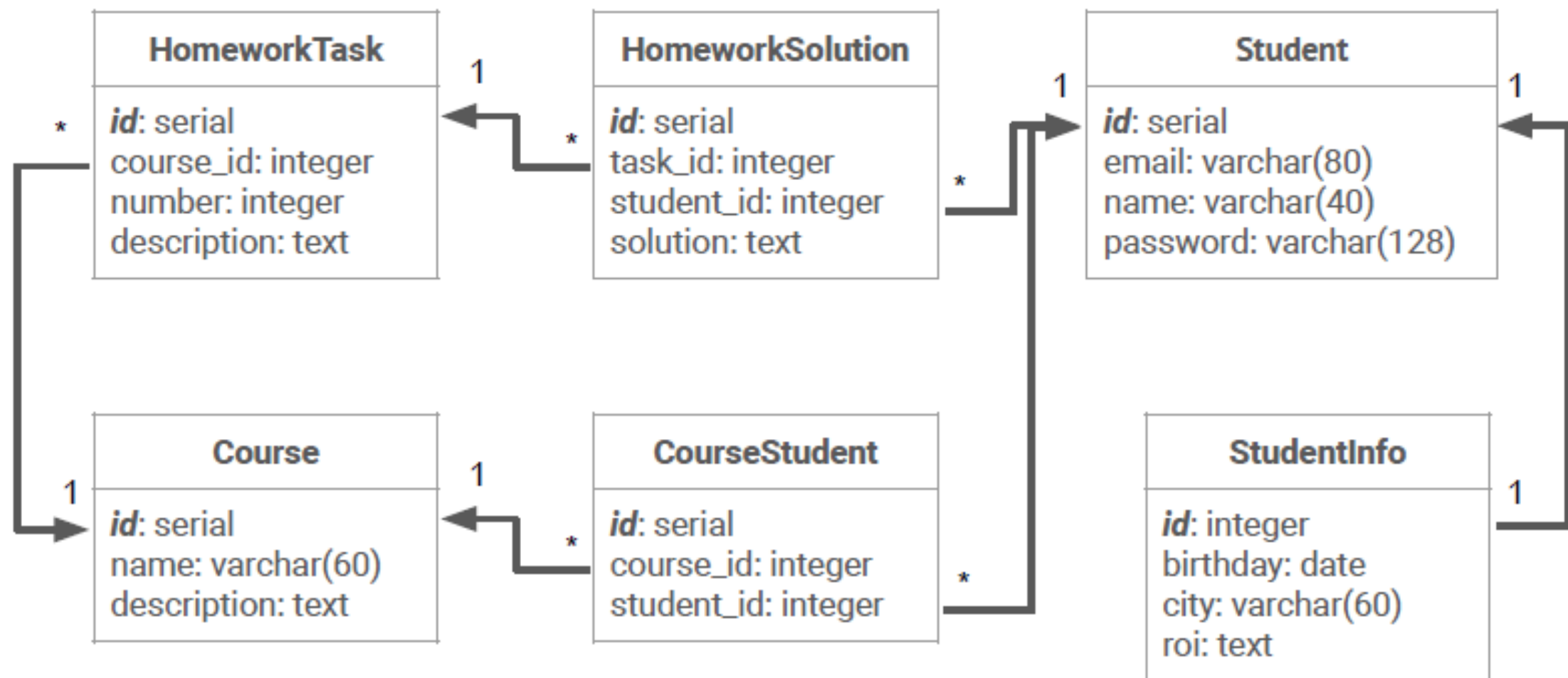
Многие ко многим

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

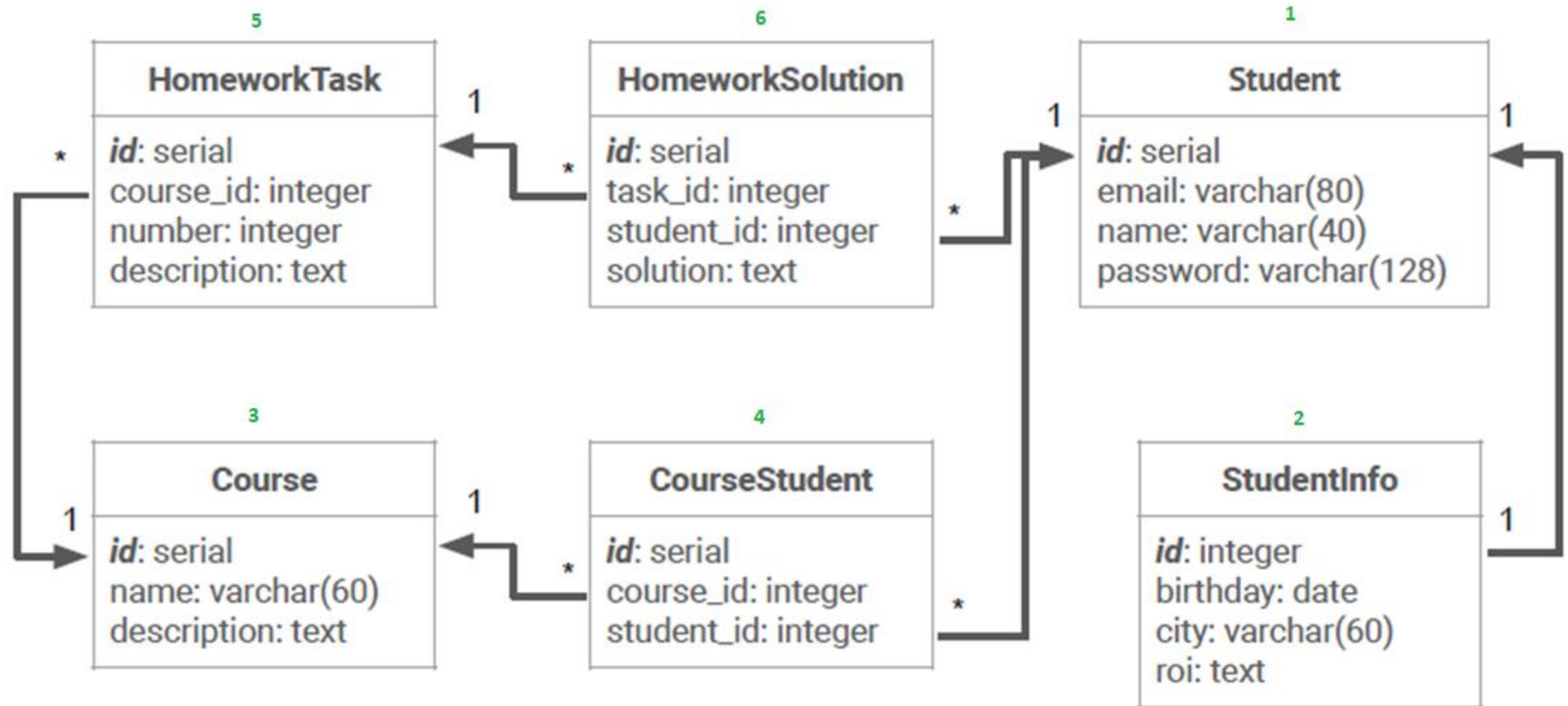


```
create table if not exists HomeworkSolution (  
    id serial primary key,  
    task_id integer not null references HomeworkTask(id),  
    student_id integer not null references Student(id),  
    solution text not null  
);
```

Итоговая схема



Итоговая схема



Нормальные формы

Нормальные формы (НФ)

Нормализация — процесс постепенного преобразования отношений (таблиц) для того, чтобы убрать дублирование данных. Это помогает уменьшить потенциальную противоречивость в БД.

HomeworkTask

<u>id</u>	course	number	description
1	Python	1	...
2	Python	2	...
3	JavaScript	1	...
4	Python	3	...

Первая нормальная форма (1НФ)

Сохраняемые данные на пересечении строк и столбцов должны представлять скалярное значение, а таблицы не должны содержать повторяющихся строк.

<u>Сотрудник</u>	Номер телефона
Иванов И. И.	283-56-82, 390-57-34
Петров П. П.	708-62-34



<u>Сотрудник</u>	<u>Номер телефона</u>
Иванов И. И.	283-56-82
Иванов И. И.	390-57-34
Петров П. П.	708-62-34

Вторая нормальная форма (2НФ)

1НФ + каждый столбец, который не является ключом, должен зависеть от первичного ключа.

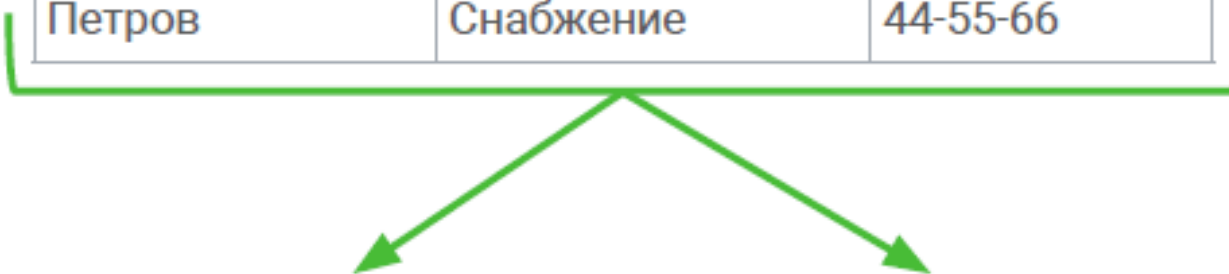
<u>Филиал компании</u>	<u>Должность</u>	Зарплата	Наличие графического планшета
Филиал в Томске	Дизайнер	40000	Есть
Филиал в Москве	Программист	60000	Нет
Филиал в Томске	Программист	40000	Нет

<u>Филиал компании</u>	<u>Должность</u>	Зарплата	<u>Должность</u>	Наличие графического планшета
Филиал в Томске	Дизайнер	40000		
Филиал в Томске	Программист	60000	Дизайнер	Есть
Филиал в Москве	Программист	40000	Программист	Нет

Третья нормальная форма (3НФ)

2НФ + каждый столбец, который не является ключом, должен зависеть от первичного ключа **напрямую**.

<u>Сотрудник</u>	Отдел	Телефон
Гришин	Бухгалтерия	11-22-33
Васильев	Бухгалтерия	11-22-33
Петров	Снабжение	44-55-66



<u>Сотрудник</u>	Отдел
Гришин	Бухгалтерия
Васильев	Бухгалтерия
Петров	Снабжение

<u>Отдел</u>	Телефон
Бухгалтерия	11-22-33
Снабжение	44-55-66

Остальные нормальные формы

- Нормальная форма Бойса-Кодда (НФБК)
- Четвертая нормальная форма (4НФ)
- Пятая нормальная форма (5НФ)
- Шестая нормальная форма (6НФ)

Статья с примерами на Хабре:

<https://habr.com/ru/articles/254773/>

В большинстве случаев достаточно нормализации базы данных до третьей нормальной формы (3НФ). Это означает, что таблицы разделены таким образом, чтобы избежать повторения данных и избыточности, а также все неключевые атрибуты функционально зависят от ключа целиком, а не от его части.

Третья нормальная форма (3НФ) предоставляет баланс между эффективностью и структурной чистотой данных. Она позволяет избежать многих типов аномалий при манипуляции данными, обеспечивает хорошую читаемость и обслуживаемость базы данных.

Четвертая нормальная форма (4НФ) и выше, такие как нормальная форма Бойса-Кодда (BCNF) и пятая нормальная форма (5НФ), обычно требуются в более сложных случаях, когда схема базы данных содержит сложные зависимости, многозначные зависимости, или когда высокая степень надежности и избежание всех возможных аномалий становится абсолютно необходимыми.

Плюсы и минусы нормализации

Плюсы	Минусы
Декомпозиция информации	Время на приведение к нормальным формам
Строгое хранение данных без возможности хранить дублированную и противоречивую информацию	Накладные расходы при извлечении информации на объединение таблиц

Выводы

Нормализация позволяет избегать дублирования данных и упрощать редактирование сущностей.

Есть несколько видов нормальных форм. Все их знать и помнить не обязательно, но важно всегда задавать себе вопрос:

«Должны ли эти данные быть в отдельной таблице или достаточно дополнительной колонки?».