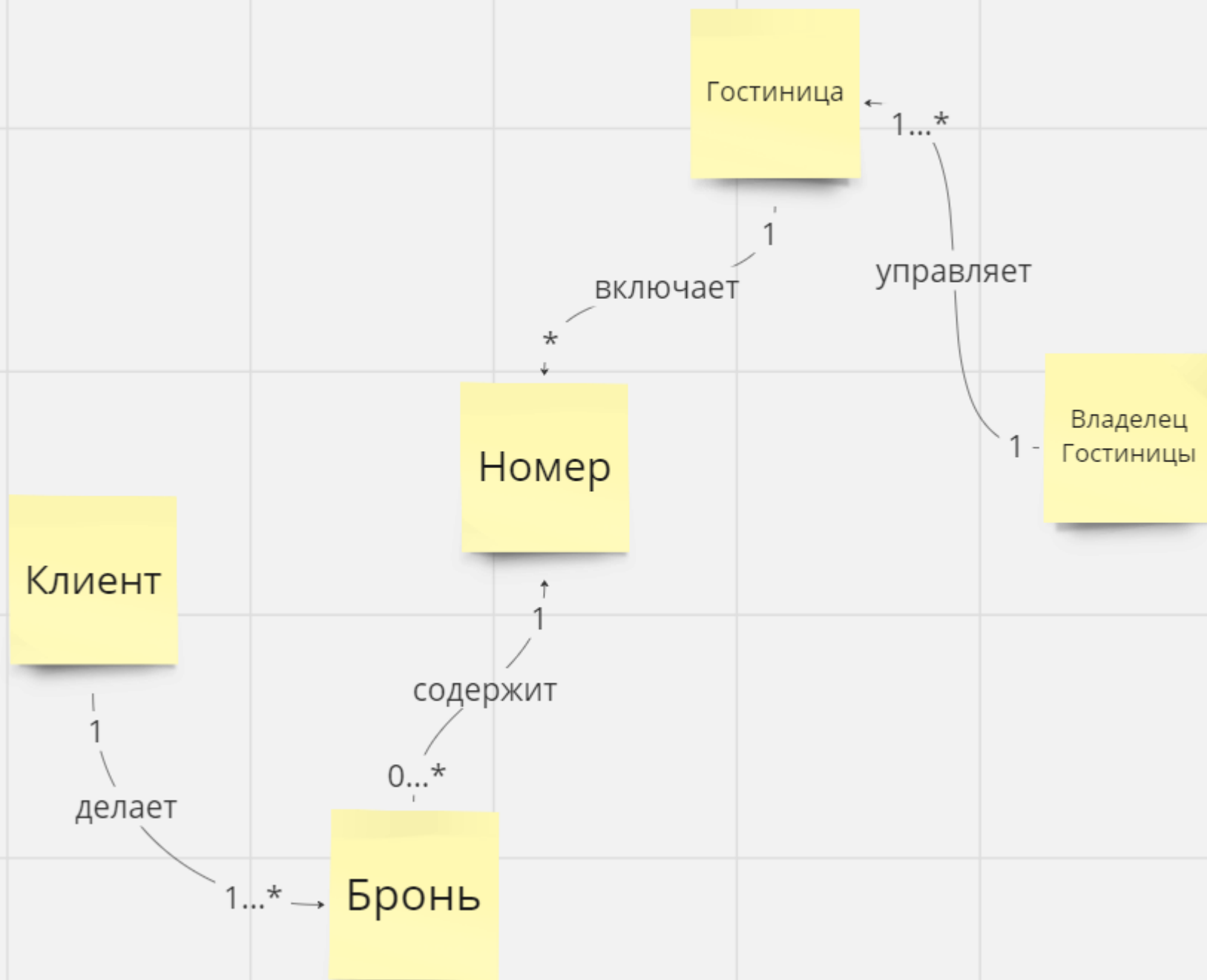


**Моделирование. Логический уровень**

Вы молодцы! Вот получившаяся за 3 шага проектирования концептуальная модель:



Мы:

- выделили сущности
- определили связи
- отметили кратность(мощность) связей

\*\*\*Помните, что концептуальную модель важно согласовать именно с бизнесом (стейкхолдерами) и архитектурой, это КОМАНДНЫЙ этап работы, тут нет правильного и неправильного решения. Есть только наиболее подходящее для вашей системы (в контексте вашей предметной области).

**Что нужно, чтобы концептуальную модель превратить в логическую?**

- Определить атрибуты для каждой сущности
- Преобразовать связи многие-ко-многим (если такие есть) в обычные
- Определить первичные и внешние ключи, по которым сущности будут связаны между собой

## Сначала немного теории

**Атрибуты** — это основные единицы данных, которые описывают характеристики сущностей, представленных в базе данных. Они также часто называются полями, свойствами или характеристиками. Каждый атрибут хранит определенный тип информации, который специфичен для сущности(объекта) в базе данных.

## Атрибуты



ФИО	Должность	Телефон
Смирнов Иван Петрович	начальник	53-22
Кузнецов Алексей Владимирович	главный инженер	53-12
Морозов Сергей Александрович	инженер	53-33
Воронов Дмитрий Николаевич	инженер	53-34

- Атрибуты — это столбцы в таблице базы данных. Каждый атрибут хранит один конкретный тип информации о сущности.
- Атрибуты могут называться полями (особенно в контексте форм или записей), свойствами (в объектно-ориентированном программировании) или характеристиками (в более общем контексте).

Допустим, у нас есть база данных "Библиотека" с таблицей "Книги". В этой таблице каждая запись (строка) представляет отдельную книгу(экземпляр сущности/объекта). Атрибуты(столбцы) в этой таблице могут включать:

1. **Название книги:** Хранит название каждой книги (например, "Война и мир").
2. **Автор:** Хранит имя автора книги (например, "Лев Толстой").
3. **Год издания:** Хранит год издания книги (например, 1869).
4. **Жанр:** Хранит жанр книги (например, "Роман").

В этом примере, каждый из этих атрибутов представляет различную характеристику книги и хранится в отдельном столбце таблицы. Каждая строка в таблице "Книги" представляет одну книгу с уникальным набором значений этих атрибутов.

*\*\*\*Внимание! Вам могут попадаться люди, которые любят научные термины и позанудствовать, мы не против, делимся с вами вот такой картинкой, чтобы вы понимали что есть что:*

Отношение	целое	строка		целое		Типы данных
	номер	имя	должность	деньги		Домены
	Табельный номер	Имя	Должность	Оклад	Премия	Атрибуты
	2934	Иванов	инженер	112	40	Кортежи
	2935	Петров	вед. инженер	144	50	
	2936	Сидоров	бухгалтер	92	35	

↑  
Ключ

таблица с данными

Связи "многие-ко-многим" в реляционных базах данных требуют специального подхода, так как напрямую в реляционной модели они не поддерживаются. Давайте разберем, почему это так, и как осуществляется преобразование этих связей.

- 1. Проблемы целостности данных:** Напрямую реализовать связь "многие-ко-многим" в реляционной БД сложно из-за потенциальных проблем с целостностью данных. Если бы разрешить такие связи, это могло бы привести к созданию избыточных и повторяющихся данных, что усложнило бы поддержку целостности и согласованности данных.

**Пример:** Предположим, у нас есть две таблицы в базе данных: "Студенты" и "Курсы". Студент может записаться на множество курсов, а на каждый курс может записаться множество студентов.

Если мы попытаемся напрямую связать эти две таблицы без промежуточной таблицы, то нам придется добавлять записи для

каждой комбинации студента и курса в обе таблицы (в таблице Студенты будет запись вида Студент1 - Курс1, а в таблице Курсы будет запись Курс1 - Студент1). Это приводит к дублированию данных и увеличению риска несогласованности.

2. **Сложность обновления и удаления:** Прямые связи "многие-ко-многим" усложняют операции обновления и удаления данных, так как изменения в одной таблице могут потребовать множественных изменений в другой.

**Пример:** Рассмотрим ту же ситуацию со связью между студентами и курсами.

Обновление: Если студент изменяет свой курс, нам нужно будет обновить информацию во множестве записей в обеих таблицах, что усложняет процесс обновления.

Удаление: Представим, что один из курсов отменяется. Нам нужно будет найти и удалить все записи, связанные с этим курсом, у каждого студента, который на него записался, в двух таблицах. Это трудоёмко, особенно если записей много, и увеличивает риск случайного удаления важной информации.

Преобразование связей "многие-ко-многим" обычно включает создание промежуточной таблицы.

1. **Создание промежуточной таблицы:** Для каждой связи "многие-ко-многим" создается отдельная промежуточная таблица, которая разделяет эту связь на две обычные связи "один-ко-многим".
2. **Добавление внешних ключей:** Промежуточная таблица включает в себя внешние ключи, которые соединяются с первичными ключами обеих исходных таблиц. Эти внешние ключи вместе формируют составной первичный ключ промежуточной таблицы.
3. **Представление связей:** Промежуточная таблица фактически представляет все уникальные комбинации связей между записями в двух исходных таблицах.

Допустим, у нас есть две сущности: "Книга" и "Читатель". Читатель может может взять множество книг, и у каждой книги может быть множество читателей (то есть связь между сущностям - многие-ко-многим, которую нужно "развязать").

- Создается промежуточная таблица, например, "Выдача".
- В таблице "Выдача" будут столбцы "ID Книга" и "ID Читатель", каждый из которых является внешним ключом на соответствующие записи в таблицах "Код Книга" и "N\_чит\_билета Читатель".
- Каждая запись в таблице "Выдача" представляет уникальную комбинацию Книги, выданной Читателю.



Первичные и внешние ключи в реляционных базах данных нужны для управления и поддержания целостности данных.

1. **Обеспечение уникальности и идентификации:** Ключи гарантируют, что каждая запись(строка) в таблице уникальна и может быть легко идентифицирована. Это особенно важно для операций поиска, обновления и удаления данных.
2. **Связывание данных между таблицами:** Они обеспечивают логическую связь между различными таблицами, позволяя интегрировать и сопоставлять данные из разных источников.
3. **Поддержка целостности данных:** Ключи помогают поддерживать целостность данных, убеждаясь, что связи между таблицами основаны на действительных и существующих данных (что существует читатель который взял существующую книгу, и наоборот).

Первичный ключ

- **Что это:** Уникальный идентификатор для каждой записи в таблице.
- **Зачем:** Для обеспечения уникальности каждой строки в таблице и упрощения её идентификации(отличия от других строк).
- **Пример:** В таблице "Руководители" первичным ключом может быть "Номер". Каждый руководитель имеет уникальный номер, который отличает его от других руководителей.



### Внешний ключ

- **Что это:** Уникальная "ссылка" на первичный ключ в другой таблице.
- **Зачем:** Для создания и поддержания связей между таблицами и обеспечения целостности данных.
- **Пример:** В таблице "Сотрудники" поле "Номер руководителя" является внешним ключом, который ссылается на "Номер" в таблице "Руководители". Это означает, что каждая запись в "Сотрудники" ассоциирована с конкретным руководителем из таблицы "Руководители".

Первичный ключ может быть таким:



## Первичный ключ

```
graph TD; A[Первичный ключ] --> B[Естественные ключи (Natural Keys)]; A --> C[Суррогатные ключи (Surrogate Keys)];
```

### Естественные ключи (Natural Keys):

- Основаны на естественных, уже существующих данных, которые уникальны для каждой записи в таблице.
- Пример: Национальный идентификационный номер, номер паспорта.

### Суррогатные ключи (Surrogate Keys):

- Искусственно созданные ключи, не имеющие естественной бизнес-значимости.
- Пример: Автоинкрементные идентификаторы, такие как порядковый номер записи или UUID.

А И первичный И внешние, могут быть такими:

## Первичный ключ

### Простые ключи (Simple Keys):

- Состоят из одного атрибута.
- Пример: "ID курса".

### Составные ключи (Composite Keys):

- Состоят из двух или более атрибутов, которые вместе образуют уникальный идентификатор записи.
- Пример: Комбинация "ID курса" и "Дата начала" для уникальной идентификации конкретного занятия в учебном расписании.

## Внешний ключ

### Простые внешние ключи (Simple Foreign Keys):

- Ссылка на первичный ключ в другой таблице, состоящая из одного столбца.
- Пример: "ID преподавателя" в таблице "Курс", ссылка на "ID преподавателя" в таблице "Преподаватели".

### Составные внешние ключи (Composite Foreign Keys):

- Ссылки, состоящие из нескольких столбцов, каждый из которых ссылается на часть составного первичного ключа в другой таблице.
- Пример: В таблице "Регистрации" внешние ключи "ID студента" и "ID курса", вместе указывающие на уникальную запись в таблице "Студенты и Курсы".

факультет
ФАП
ФАТС
ФИРТ
ИНЭК

простой ключ

факультет	специальность
ФАП	220200
ФАТС	140100
ФАТС	140200
ФИРТ	071800
ФИРТ	220200
ФИРТ	010500
ИНЭК	080100
ИНЭК	080500

составной ключ

специальность
220200
140100
140200
071800
010500
080100
080500

простой ключ