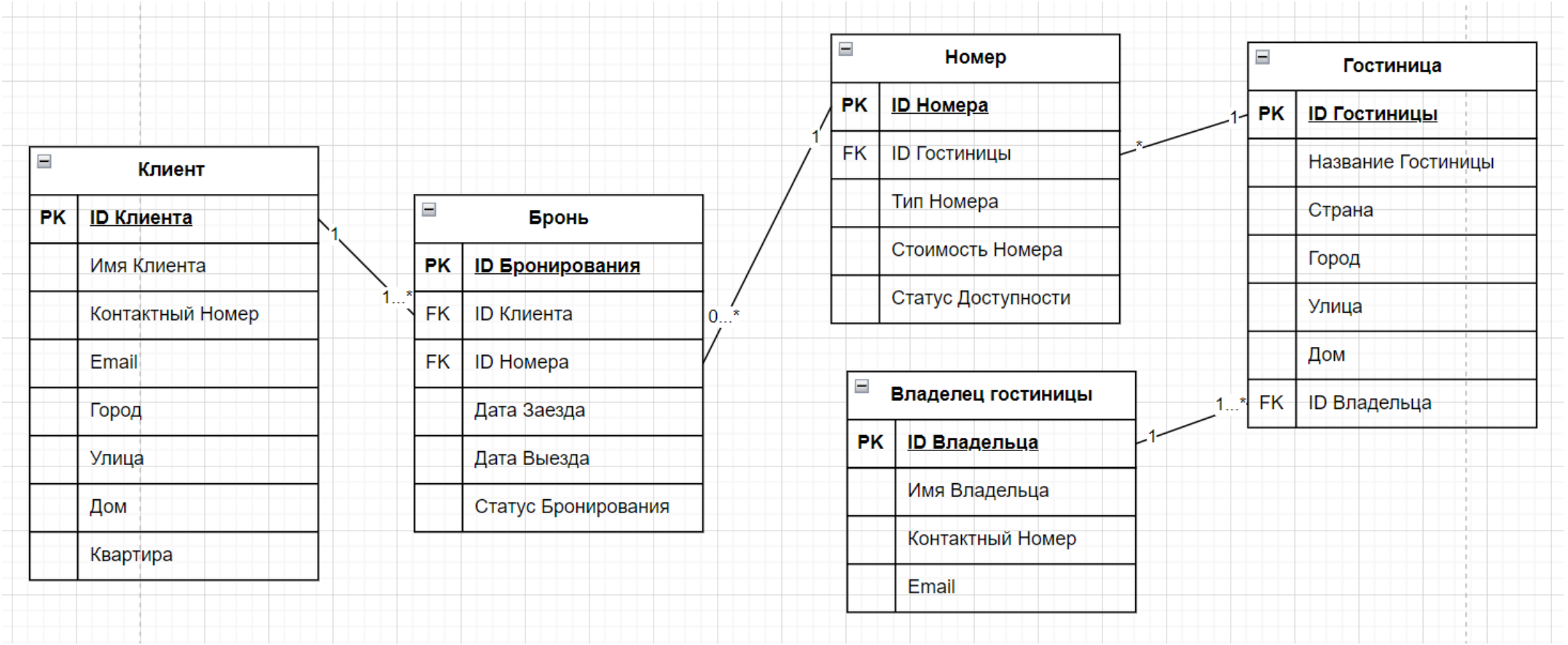
Моделирование. Физическая модель

Отлично! Вот такая получилась логическая модель данных, где мы еще провели нормализацию.



Идём далее, к физической модели.

После разработки логической модели данных следующим шагом является преобразование её в физическую модель данных. Физическая модель - это уже конкретное воплощение вашей модели данных (например, в SQL коде и хранимых данных), поэтому она зависит от выбранной системы управления базами данных (СУБД). В нашем случае будет PostgreSQL, поэтому физическая модель будет адаптирована под особенности и возможности этой СУБД.

Что нужно, чтобы логическую модель превратить в физическую?

- Переименовать все сущности и атрибуты в соответствии с нормами наименований данных
- Определить типы данных для каждого атрибута
- Указать обязательность и уникальность, значения по умолчанию для атрибутов

Сначала немного теории

1. Правила наименования данных

При наименовании элементов в физической модели баз данных, важно придерживаться четких и последовательных правил, чтобы обеспечить читаемость, поддержку и масштабируемость вашей базы данных. Вот общепринятые правила для наименования таблиц, атрибутов и других элементов:

1. Использование существительных во множественном числе: Таблицы часто называют во множественном числе, так как они содержат множество записей одного типа. Например, `Users`, `Orders`, `Products`.
2. Для атрибутов - избегание специальных символов и пробелов (используйте snake_case): Используйте подчеркивания для разделения слов, например, `customer_orders`, вместо пробелов или специальных символов.
3. Ясность и конкретика: Название должно точно описывать содержимое поля. Например, `email_address`, `order_date`.
4. Префиксы для уникальности и ясности: В случае внешних ключей используйте префиксы для указания на связанную таблицу, например, `customer_id` для ссылки на таблицу `Customers`.

2. Типы данных

Типы данных определяют, какой вид информации может храниться в каждом столбце таблицы и как эта информация может быть обработана. Очень важно выбирать оптимальные типы данных для каждого атрибута.

- Указание типа данных помогает обеспечить, что данные, вводимые в базу данных, соответствуют ожидаемому формату. Например, если поле предназначено для хранения дат (тип данных `DATE`), система базы данных будет отклонять любые некорректные даты или строки.

- Разные типы данных занимают разное количество места в памяти. Правильный выбор типа данных помогает оптимизировать использование дискового пространства. Например, тип данных `INT` для целых чисел занимает меньше места, чем `BIGINT`.
- Оптимизированные типы данных могут ускорить выполнение запросов, так как обработка и сортировка данных становятся более эффективными. Например, поиск по числовому полю обычно быстрее, чем по текстовому полю большого размера.
- Правильные типы данных обеспечивают совместимость при интеграции разных систем и обмене данными между ними. Например, формат даты может отличаться в разных системах, и корректный тип данных поможет избежать ошибок при обмене данными.

Есть общие типы данных:

1. Числовые: `INT` (целые числа), `DECIMAL` (десятичные числа), `FLOAT` (числа с плавающей точкой).
2. Строковые: `VARCHAR` (переменная длина строки), `CHAR` (фиксированная длина строки).
3. Дата и Время: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`.
4. Логические: `BOOLEAN` (истина/ложь).

Но есть и отличия между СУБД. Разные СУБД, такие как MySQL, PostgreSQL, Oracle, и SQL Server, могут иметь различия в поддержке и точных определениях типов данных. Например, тип данных `TEXT` в одной СУБД может поддерживать другую максимальную длину, чем в другой.

3. Обязательность, уникальность, значения по умолчанию для атрибутов

Важность указания ограничений для каждого атрибута в физической модели данных заключается в том, что они обеспечивают правильность, целостность и предсказуемость данных в базе данных.

Обязательность (NOT NULL)

- Атрибут должен всегда иметь значение; нулевые значения (NULL) не допускаются.

- Пример: В таблице `Пользователи`, атрибут `email` должен быть обязательным. Это означает, что каждый пользователь должен иметь адрес электронной почты. Тогда нам нужно отметить что он NOT NULL, и СУБД не позволит добавить нам строку, где атрибут email будет отсутствовать.

***внимание, все первичные ключи всегда NOT NULL

Уникальность (UNIQUE)

- Значения в этом атрибуте должны быть уникальными среди всех записей(строк) в таблице.
- Пример: В той же таблице `Пользователи`, атрибут `идентификационный номер` должен быть уникальным. Это гарантирует, что каждый пользователь имеет уникальный идентификатор, исключая путаницу или ошибки при идентификации пользователей. Если у атрибута мы проставим ограничение UNIQUE, то СУБД не даст нам добавить такую строку в таблицу, где уже есть похожее значение у данного атрибута.

***внимание, все первичные и внешние ключи всегда UNIQUE

Значения по умолчанию (DEFAULT)

- Если для атрибута не указано значение, будет использоваться значение по умолчанию(оно указывается в СУБД).
- Пример: В таблице `Заказы`, атрибут `статус заказа` может иметь значение по умолчанию `В обработке`. Это означает, что если при создании нового заказа статус не указан, он автоматически устанавливается как `В обработке`, что помогает в управлении потоком заказов. То есть, при добавлении строки в таблицу без указания атрибута статус заказа - СУБД сама заполнит это поле.

Эти ограничения помогают гарантировать, что данные в базе данных остаются последовательными, надежными и легко управляемыми. Они предотвращают появление неполных или некорректных данных.

Давайте начнём превращать логическую модель в физическую

Шаг 1. Переименовать все сущности и атрибуты в соответствии с нормами наименований данных

Сущность "Клиент" -> Таблица `clients`

- ID Клиента (PK) -> `client_id`
- Имя Клиента -> `client_name`
- Контактный Номер -> `contact_number`

- Email -> `email`
- Город -> `city`
- Улица -> `street`
- Дом -> `house_number`
- Квартира -> `apartment_number`