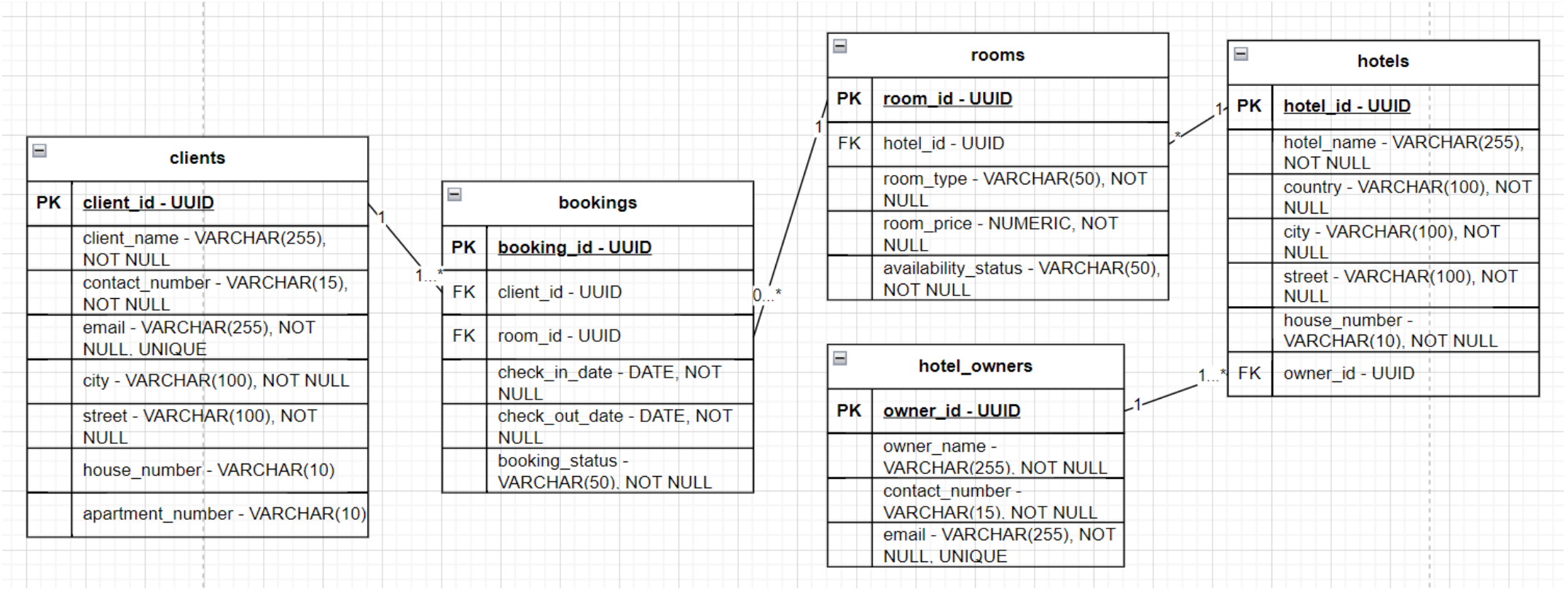
Моделирование. Физическая модель. SQL.

Практикуемся

В итоге получилась такая физическая модель данных:



Мы молодцы!

Прошли большой путь от требований к физической модели данных.

Остался последний штрих - проверить, что наша структура данных отвечает всем требованиям и сценариям использования системы (то есть, что все запросы SQL выполнимы и т.д.)

Но прежде давайте поймём принципы запросов к данным (основы SQL)

SQL (Structured Query Language) — это мощный декларативный язык запросов, используемый для управления и манипуляции данными в реляционных базах данных. Основные принципы SQL:

1. Универсальность: SQL является стандартизированным языком, принятым для работы с реляционными базами данных. Он позволяет пользователям создавать, читать, обновлять и удалять данные (CRUD-операции).

2. Декларативность: В SQL вы описываете, *что* хотите сделать с данными, а не *как* это сделать. Система управления базами данных (СУБД) определяет наиболее эффективный способ выполнения вашего запроса.

3. Работа с данными: SQL позволяет выполнять сложные запросы для обработки больших объемов данных, включая фильтрацию, сортировку, группировку и объединение данных из разных таблиц.

4. Транзакционность: SQL поддерживает концепцию транзакций, которые обеспечивают надежность и целостность данных, позволяя выполнять группу операций как единое целое.

Виды SQL

1. DDL (Data Definition Language): Язык определения данных, используется для создания, изменения и удаления объектов базы данных (то есть для работы с таблицами и другими элементами СУБД). Примеры команд DDL: `CREATE`, `ALTER`, `DROP`. Так выглядит запрос на создание таблицы customers:
   ```
   CREATE TABLE customers (
      customer_id INT PRIMARY KEY,
      name VARCHAR(100),
      email VARCHAR(100)
   );
   ```

2. DML (Data Manipulation Language): Язык для непосредственной работы с данными, используется для вставки, обновления, удаления и выборки данных. Вы точно уже видели команды DML: `INSERT`, `UPDATE`, `DELETE`, `SELECT`. Пример запроса добавления данных в таблицу:

```
INSERT INTO customers (customer_id, name, email)
VALUES (1, 'John Doe', 'john@example.com');
```

3. DCL (Data Control Language): Язык контроля данных, включает команды для управления правами доступа к данным. Примеры команд DCL: `GRANT`, `REVOKE`. Предоставление прав пользователям выполняется так:
   GRANT SELECT ON customers TO user1;

4. TCL (Transaction Control Language): Язык управления транзакциями, используется для управления транзакциями в базе данных. Примеры команд TCL: `COMMIT`, `ROLLBACK`. Применение изменений в базе данных выполняется командой:

```
COMMIT;
```

Мы не будем с вами тут непосредственно ИЗУЧАТЬ SQL, для этого есть много бесплатных онлайн курсов и так далее.

Мы рассмотрим основные кейсы использования, чтобы понять, как сценарии использования SQL влияют на структуру данных.

***вы всегда можете изучить SQL по документации, например - https://postgrespro.ru/docs/postgrespro/15/sql

Давайте попробуем использовать SQL DDL для быстрого создания нашей базы данных в PostgreSQL. Иногда разработчики ждут от аналитика не диаграмму физической модели данных, а как раз готовый код SQL DDL, при выполнении которого в СУБД будет создана структура данных по физической модели.

***основы по командам можете посмотреть здесь - https://postgrespro.ru/docs/postgresql/15/ddl-basics

***используя этот сервис https://dbdiagram.io/, вы можете создавать ER-диаграммы, а потом экспортировать их в виде .sql файла для разных СУБД (в т.ч. PostgreSQL). В этом файле будет SQL DDL код для создания таблиц по ER-диаграмме. Также можно импортировать туда SQL DDL код или .sql файлы

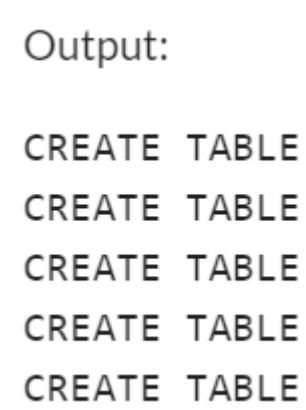
Вот такой получится у нас код:

-- Создание таблицы clients
CREATE TABLE clients (
    client_id UUID PRIMARY KEY,

```
    client_name VARCHAR(255) NOT NULL,
    contact_number VARCHAR(15) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    city VARCHAR(100) NOT NULL,
    street VARCHAR(100) NOT NULL,
    house_number VARCHAR(10),
    apartment_number VARCHAR(10)
);

-- Создание таблицы hotel_owners
CREATE TABLE hotel_owners (
    owner_id UUID PRIMARY KEY,
    owner_name VARCHAR(255) NOT NULL,
    contact_number VARCHAR(15) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE
);

-- Создание таблицы hotels
CREATE TABLE hotels (
    hotel_id UUID PRIMARY KEY,
    hotel_name VARCHAR(255) NOT NULL,
    country VARCHAR(100) NOT NULL,
    city VARCHAR(100) NOT NULL,
    street VARCHAR(100) NOT NULL,
    house_number VARCHAR(10) NOT NULL,
    owner_id UUID REFERENCES hotel_owners(owner_id)
);

-- Создание таблицы rooms
CREATE TABLE rooms (
    room_id UUID PRIMARY KEY,
    hotel_id UUID REFERENCES hotels(hotel_id),
    room_type VARCHAR(50) NOT NULL,
    room_price NUMERIC NOT NULL,
    availability_status VARCHAR(50) NOT NULL
);
```

```
-- Создание таблицы bookings
CREATE TABLE bookings (
    booking_id UUID PRIMARY KEY,
    client_id UUID REFERENCES clients(client_id),
    room_id UUID REFERENCES rooms(room_id),
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    booking_status VARCHAR(50) NOT NULL
);
```

Вы можете использовать онлайн сервис https://onecompiler.com/postgresql, вставить туда этот код, выполнить, и увидеть справа, что таблицы были успешно созданы:



Давайте попробуем проверить, действительно ли они созданы? Добавляем команды из SQL DML.

Снова используем онлайн сервис https://onecompiler.com/postgresql. Он позволяет, к сожалению, использовать только за один раз все нужные команды, поэтому добавим к коду выше, в самый конец через строку, команды выборки данных из таблиц:

```
Select * from clients;
Select * from hotel_owners;
Select * from hotels;
Select * from rooms;
Select * from bookings;
```

И в итоге увидим, что у нас произошла выборка данных из таблиц, и сервис верно указывает, что везде у нас 0 строк, потому что мы не выполняли никаких команд по НАПОЛНЕНИЮ данными таблиц. Но зато мы видим названия столбцов:

```
CREATE TABLE
 client_id | client_name | contact_number | email | city |
-----------+-------------+----------------+-------+------+-
(0 rows)

 owner_id | owner_name | contact_number | email
----------+------------+----------------+-------
(0 rows)

 hotel_id | hotel_name | country | city | street | house_nu
----------+------------+---------+------+--------+---------
(0 rows)

 room_id | hotel_id | room_type | room_price | availability
---------+----------+-----------+------------+-------------
(0 rows)

 booking_id | client_id | room_id | check_in_date | check_o
------------+-----------+---------+---------------+--------
(0 rows)
```

Давайте попробуем наполнить таблицы данными. Добавляем команды из SQL DML.

Добавим вот такой код, между кодом создания таблиц, и кодом выборки данных из таблиц:

```
-- Добавление записей в таблицу clients
INSERT INTO clients (client_id, client_name, contact_number, email, city, street, house_number, apartment_number) VALUES
('a0e1b4e8-3f9b-11ec-8d3d-0242ac130003', 'John Doe', '123-456-7890', 'john.doe@example.com', 'New York', '5th Avenue', '101', '20'),
('a0e1b7d2-3f9b-11ec-8d3d-0242ac130003', 'Jane Smith', '987-654-3210', 'jane.smith@example.com', 'Los Angeles', 'Sunset Boulevard', '202', '15');

-- Добавление записей в таблицу hotel_owners
INSERT INTO hotel_owners (owner_id, owner_name, contact_number, email) VALUES
('a0e1ba9c-3f9b-11ec-8d3d-0242ac130003', 'Alice Johnson', '555-123-4567', 'alice.johnson@example.com'),
```

('a0e1bcb6-3f9b-11ec-8d3d-0242ac130003', 'Bob Williams', '555-765-4321', 'bob.williams@example.com');

-- Добавление записей в таблицу hotels
INSERT INTO hotels (hotel_id, hotel_name, country, city, street, house_number, owner_id) VALUES
('a0e1bf0a-3f9b-11ec-8d3d-0242ac130003', 'Grand Plaza', 'USA', 'New York', 'Central St', '50', 'a0e1ba9c-3f9b-11ec-8d3d-0242ac130003'),
('a0e1c138-3f9b-11ec-8d3d-0242ac130003', 'Sunset Resort', 'USA', 'Los Angeles', 'Beach Ave', '100', 'a0e1bcb6-3f9b-11ec-8d3d-0242ac130003');

-- Добавление записей в таблицу rooms
INSERT INTO rooms (room_id, hotel_id, room_type, room_price, availability_status) VALUES
('a0e1c368-3f9b-11ec-8d3d-0242ac130003', 'a0e1bf0a-3f9b-11ec-8d3d-0242ac130003', 'Deluxe', 200.00, 'Available'),
('a0e1c4d2-3f9b-11ec-8d3d-0242ac130003', 'a0e1c138-3f9b-11ec-8d3d-0242ac130003', 'Standard', 150.00, 'Available');

-- Добавление записей в таблицу bookings
INSERT INTO bookings (booking_id, client_id, room_id, check_in_date, check_out_date, booking_status) VALUES
('a0e1c6dc-3f9b-11ec-8d3d-0242ac130003', 'a0e1b4e8-3f9b-11ec-8d3d-0242ac130003', 'a0e1c368-3f9b-11ec-8d3d-0242ac130003', '2021-12-01',
'2021-12-10', 'Confirmed'),
('a0e1c85e-3f9b-11ec-8d3d-0242ac130003', 'a0e1b7d2-3f9b-11ec-8d3d-0242ac130003', 'a0e1c4d2-3f9b-11ec-8d3d-0242ac130003', '2021-12-05',
'2021-12-15', 'Confirmed');
Выполняем код и видим, что наши таблицы "ожили"! Там есть теперь данные (по 2 строки в каждой таблице), которые позволяют уже понять, как это всё выглядит в реальной жизни.

```
              client_id               | client_name | contact
--------------------------------------+-------------+--------
 a0e1b4e8-3f9b-11ec-8d3d-0242ac130003 | John Doe    | 123-456
 a0e1b7d2-3f9b-11ec-8d3d-0242ac130003 | Jane Smith  | 987-654
(2 rows)

               owner_id               |  owner_name   | conta
--------------------------------------+---------------+------
 a0e1ba9c-3f9b-11ec-8d3d-0242ac130003 | Alice Johnson | 555-1
 a0e1bcb6-3f9b-11ec-8d3d-0242ac130003 | Bob Williams  | 555-7
(2 rows)

               hotel_id               |  hotel_name   | count
--------------------------------------+---------------+------
```

Давайте наконец попробуем выполнить запрос, который отдаст все данные по клиенту с id=a0e1b4e8-3f9b-11ec-8d3d-0242ac130003. Снова SQL DML.

Заменим все строчки кода в конце вида select * ..... на одну строку:

Select* from clients where client_id = 'a0e1b4e8-3f9b-11ec-8d3d-0242ac130003';

И увидим результат:

```
INSERT 0 2
              client_id               | client_name | contact_num
--------------------------------------+-------------+------------
 a0e1b4e8-3f9b-11ec-8d3d-0242ac130003 | John Doe    | 123-456-789
(1 row)
```

Хоть в таблице clients и две строки, но мы поставили условие, client_id = 'a0e1b4e8-3f9b-11ec-8d3d-0242ac130003', поэтому СУБД возвращает нам только подходящую строку.

Такого типа запросы - самые частые к базе данных, потому что это чтение данных, чтобы потом их отобразить в мобильном приложении или на сайте, или отдать по API.

Теперь, когда мы поняли основные принципы и логику SQL, делаем последние проверки нашей физической модели данных

1. Доступность данных для выборки

- Убедитесь, что ваша модель данных может легко предоставлять необходимую информацию для отображения данных, отчетов и аналитики.
- Напишите основные типичные SQL запросы к вашей будущей базе данных. Убедитесь, что их можно выполнить и что они не слишком сложные.

Например. Вы получили требование от бизнеса: получить отчет о всех бронированиях в отеле "Grand Plaza" с 1 декабря 2021 года по 31 декабря 2021 года, включая информацию о клиентах (имя, контактный номер, email) и о номерах (тип номера, стоимость). Вы пробуете написать SQL запрос и у вас получается без проблем:

```
SELECT
   c.client_name,
   c.contact_number,
   c.email,
   r.room_type,
   r.room_price,
   b.check_in_date,
   b.check_out_date
FROM
   bookings b
JOIN
   clients c ON b.client_id = c.client_id
JOIN
   rooms r ON b.room_id = r.room_id
JOIN
   hotels h ON r.hotel_id = h.hotel_id
WHERE
   h.hotel_name = 'Grand Plaza' AND
   b.check_in_date >= '2021-12-01' AND
   b.check_out_date <= '2021-12-31';
```

Тогда вы делаете вывод, что структуру данных не нужно менять, требование выполнимо, можно идти дальше.

2. Историчность данных

- Необходимо удостовериться, что вы учли все требования к системе по отслеживанию истории изменений важных данных(если это необходимо).
- Если вы обнаружили требования к историчности - нужно его реализовать, либо добавив новые атрибуты, либо новую таблицу.

Например, есть требование "сохранять историю изменений данных клиента - телефона и адреса почты".

Как его можно реализовать?

Если бы нужно было сохранять только одно прошлое изменение (например, прошлый телефон) - мы могли бы добавить просто новый атрибут old_contact_number. Но у нас требование - сохранять все изменения. Вариант решения: отдельная таблица истории изменений данных клиента.

Создайте дополнительную таблицу, например `client_changes`, где будут храниться записи об изменениях телефонных номеров и емейлов клиентов. Пример таблицы `client_changes`:

```
CREATE TABLE client_changes (
    change_id UUID PRIMARY KEY,
    client_id UUID REFERENCES clients(client_id),
    old_contact_number VARCHAR(15) NOT NULL,
    new_contact_number VARCHAR(15) NOT NULL,
    old_email VARCHAR(255) NOT NULL,
    new_email VARCHAR(255) NOT NULL,
    change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Проверка возможности CRUDL операций над вашими сущностями

- CRUDL (Создание, Чтение, Обновление, Удаление, Список) — это расширенная версия CRUD, включающая операции получения списка записей из таблицы. Рассмотрим, как проверить каждую из этих операций на основе вашей физической модели данных.
- Создав физическую модель данных в СУБД, попробуйте выполнить эти операции по очереди, убедившись, что всё работает корректно. Проверьте, соблюдаются ли ограничения целостности данных и правила валидации при выполнении операций.

На примере сущности clients. Создание (Create). Добавление новых записей в таблицу.

INSERT INTO clients (client_id, client_name, contact_number, email, city, street, house_number, apartment_number)
VALUES ('a0e1c8e4-3f9b-11ec-8d3d-0242ac130003', 'Alex Brown', '321-654-0987', 'alex.brown@example.com', 'Chicago', 'North Avenue', '303', '12');
Чтение (Read) Извлечение конкретной записи из таблицы.

SELECT * FROM clients WHERE client_id = 'a0e1b4e8-3f9b-11ec-8d3d-0242ac130003';
Обновление (Update) Изменение существующих записей в таблице.

UPDATE clients
SET contact_number = '555-123-4567', email = 'new.email@example.com'
WHERE client_id = 'a0e1b4e8-3f9b-11ec-8d3d-0242ac130003';
Удаление (Delete) Удаление записей из таблицы.

DELETE FROM clients WHERE client_id = 'a0e1c8e4-3f9b-11ec-8d3d-0242ac130003';

****обычно мало кто делает реальные операции удаления данных из баз данных. Обычно используют атрибут is_deleted вместо удаления строк. То есть, у строки, которую нужно "удалить" ставится значение этого атрибута true. А у "не удалённых" строк этот атрибут заполнен как false. Этот подход позволяет сохранять все данные для возможного аудита или восстановления. Он также помогает избежать дорогостоящих операций удаления, которые могут негативно сказаться на производительности. Конечно, со временем таблица может разрастаться, что потенциально может замедлить запросы, особенно если индексы не оптимизированы под такой подход.*
Список (List) Получение списка записей, соответствующих определенным критериям или всех записей из таблицы.

SELECT * FROM bookings WHERE check_in_date BETWEEN '2021-12-01' AND '2021-12-31';

4. Безопасность и контроль доступа (совместно с вашим администратором СУБД)

- Важно гарантировать, что данные защищены и доступ к ним строго контролируется.
- Проверьте, что в системе реализованы различные уровни доступа к данным (например, администраторы, операторы, обычные пользователи).
- Попытайтесь выполнить операции с данными от имени пользователя с ограниченными правами, чтобы убедиться, что ограничения доступа работают корректно.