Домашнее задание 4

Ответ на задание 1:

Каждая строка описывает визит животного в клинику в конкретную дату. Для удобства каждому животному присваивается уникальный номер AnimalID.
По смыслу предметной области можно сформулировать следующие функциональные зависимости (ФЗ). Раз номер животного уникален и все его характеристики не меняются от визита к визиту, то:

AnimalID => Animal
AnimalID => Species
AnimalID => Age
AnimalID => Owner

То есть, зная AnimalID, мы однозначно восстанавливаем кличку, вид, возраст и владельца животного. Эти атрибуты повторяются во всех строках, соответствующих одному и тому же животному, что приводит к избыточности данных.

Рассмотрим нормальные формы. Отношение Report находится в первой нормальной форме (1НФ), так как все атрибуты атомарны: в каждой ячейке хранится одно значение (одна дата, одна кличка и т.д.). Однако для строки отчёта логический ключ является составным: чтобы отличить один визит от другого, нужно как минимум указать животное и дату визита, а в реальной базе данных дополнительно может использоваться идентификатор конкретной процедуры. При этом атрибуты Animal, Species, Age, Owner зависят не от всего составного ключа, а только от его части - AnimalID. Это частичные функциональные зависимости, которые нарушают вторую нормальную форму (2НФ): неключевые атрибуты не должны зависеть от части составного ключа.

Такая структура приводит к классическим аномалиям. Аномалия обновления: если у владельца животного с AnimalID = 505 изменилось имя, его приходится менять во всех строках с этим идентификатором, иначе данные станут противоречивыми. Аномалия вставки: информацию о новом животном нельзя занести в таблицу без визита, так как атрибуты животного хранятся только вместе с датой посещения. Аномалия удаления: при удалении всех строк с визитами животного мы одновременно теряем и информацию о самом животном и его владельце.

Чтобы устранить частичные зависимости и привести схему к третьей нормальной форме (3НФ), выполним декомпозицию исходного отношения. Все атрибуты, которые зависят только от AnimalID, выделим в отдельное отношение, описывающее животных:

Animal(pk_animal_id, animal_name, species, age, owner)

Здесь pk_animal_id - первичный ключ (это исходный AnimalID). Для этого отношения выполняются следующие функциональные зависимости:

pk_animal_id => animal_name
pk_animal_id => species
pk_animal_id => age
pk_animal_id => owner

Все неключевые атрибуты зависят только от ключа, между ними нет зависимостей, значит отношение Animal находится в 3НФ и информация о каждом животном и его владельце хранится в базе один раз.

Оставшаяся часть данных описывает сами визиты. В упрощённом варианте, соответствующем условию задачи, достаточно хранить идентификатор животного и дату визита, поэтому формируем отношение:

Visit(pk_animal_id, pk_date_of_visit)

где составной первичный ключ - (pk_animal_id, pk_date_of_visit). В этом отношении неключевых атрибутов нет, единственная нетривиальная зависимость - ключ определяет саму строку:

(pk_animal_id, pk_date_of_visit) => pk_animal_id, pk_date_of_visit

Атрибут pk_animal_id в таблице Visit является внешним ключом, ссылающимся на Animal(pk_animal_id). Отношение Visit также находится в 3НФ, так как отсутствуют частичные и транзитивные зависимости.

Таким образом, исходное отношение Report было декомпозировано в два отношения 3НФ:

Animal(pk_animal_id, animal_name, species, age, owner)
Visit(pk_animal_id, pk_date_of_visit)

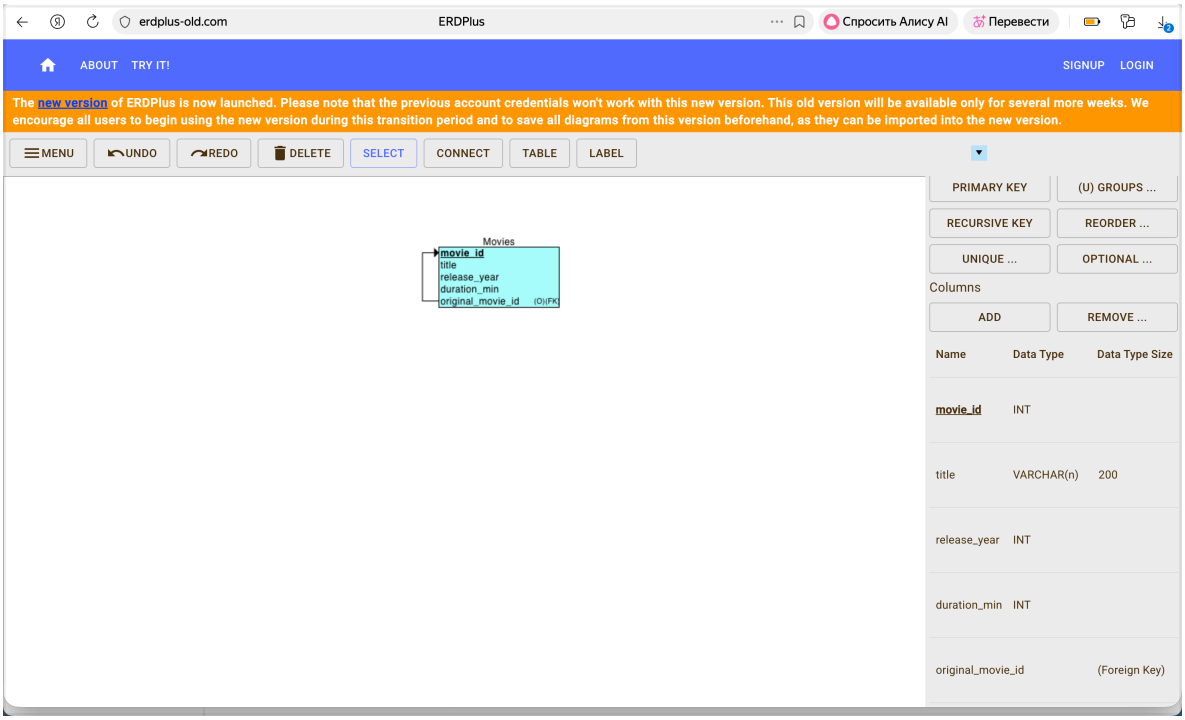
Декомпозиция выполняется без потерь: если соединить таблицы Animal и Visit по полю pk_animal_id, мы восстановим все данные исходного отчёта. При этом избыточность сведена к минимуму: информация о животном и владельце больше не дублируется в каждой строке визита, а аномалии вставки, обновления и удаления устранены.

Ответ на задание 2

В исходной концептуальной модели присутствует одна сущность **Movies** и рекурсивная связь **Sequel-of**, в которой один и тот же тип сущности участвует два раза в разных ролях: как **Original** (оригинальный фильм) и

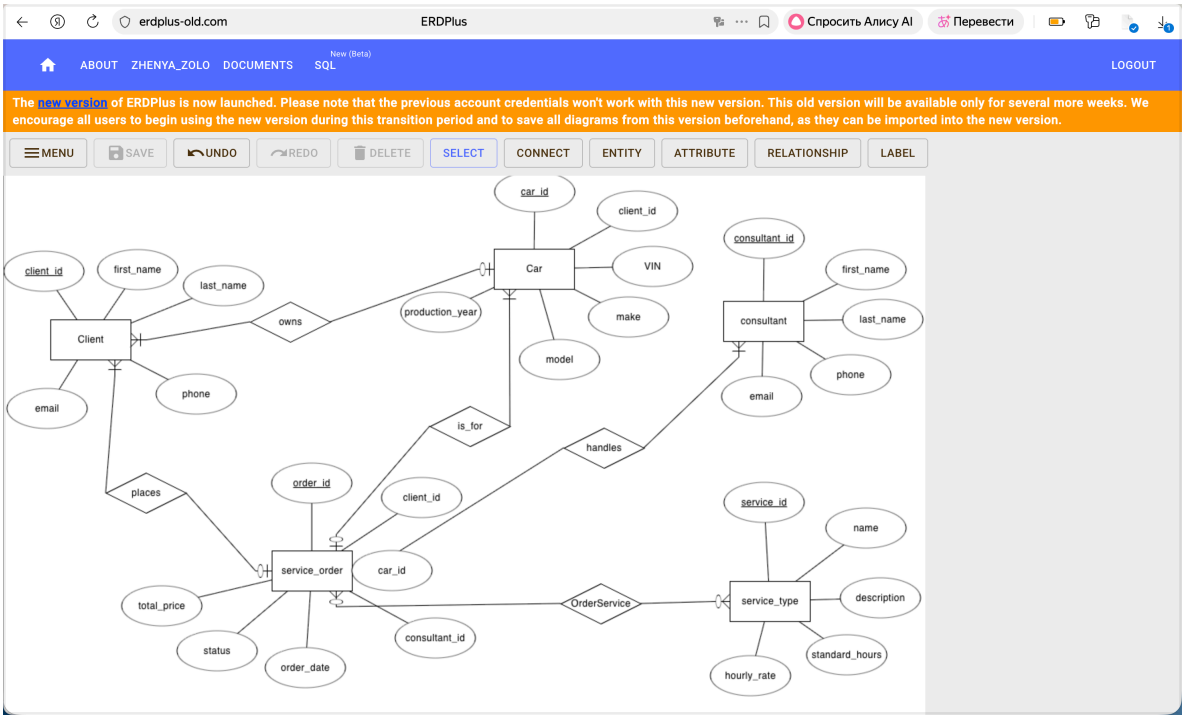
как **Sequel** (сиквел). Стрелка на стороне Original означает, что каждый сиквел может иметь только один оригинальный фильм, тогда как у оригинального фильма может быть ноль, один или несколько сиквелов. Это связь типа «один ко многим» внутри одной сущности.
При преобразовании такой модели в реляционную схему связь 1:N реализуется добавлением внешнего ключа на стороне N. Поскольку обе роли (Original и Sequel) относятся к одной сущности Movies, достаточно одной таблицы, в которой каждая запись-сиквел будет хранить ссылку на запись-оригинал. Поэтому реляционная схема содержит одну таблицу. Атрибут movie_id является первичным ключом и однозначно идентифицирует фильм. Атрибуты title, release_year и duration_min - простые описательные атрибуты, которыми «обогащена» таблица в соответствии с заданием. Атрибут original_movie_id является внешним ключом, ссылающимся на Movies(movie_id) той же таблицы, и реализует связь **Sequel-of**: для сиквела в этом поле записывается идентификатор его оригинального фильма, а для фильмов, которые не являются чьими-то сиквелами, значение может быть NULL.

Таким образом, одна таблица Movies с рекурсивным внешним ключом корректно отражает исходную концептуальную модель и выполняет требования задания по добавлению ключевых и простых атрибутов.

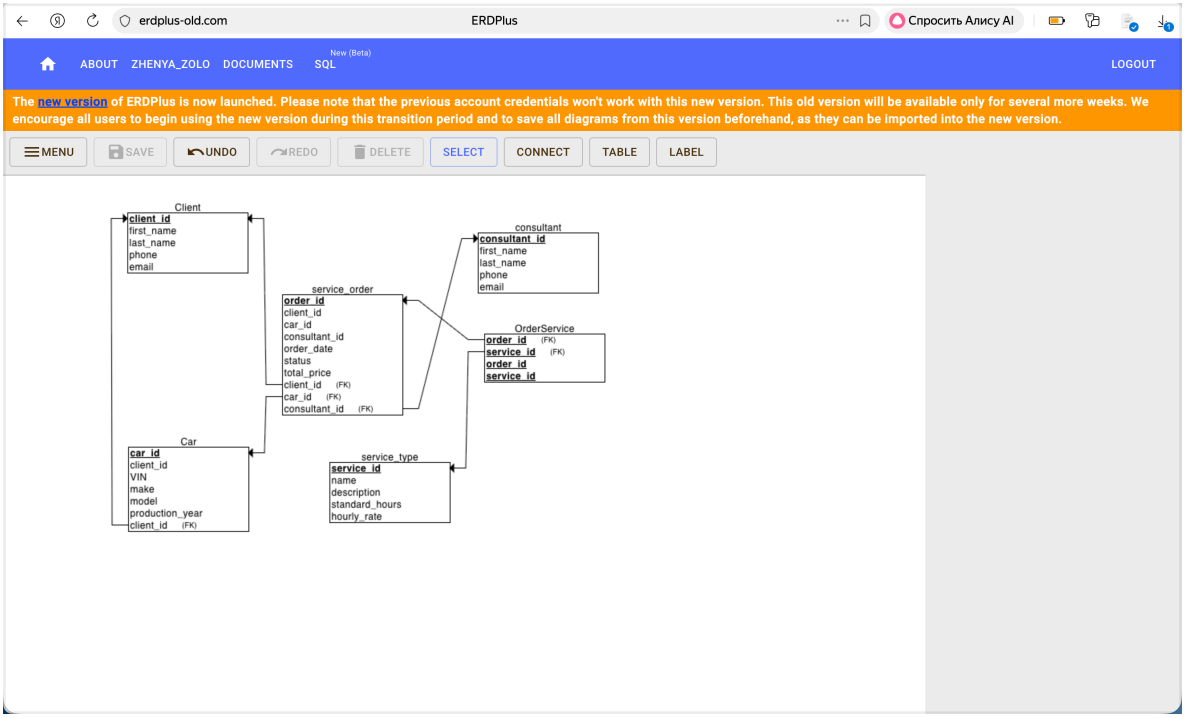


Ответ на задание 3

Концептуальная схема:



Реляционная схема:



Краткое описание модели:

Проектируемая база данных описывает работу автомастерской.

Основные сущности, выделенные на концептуальной схеме:

- **Client** - клиент сервиса (ФИО, телефон, email).
- **Car** - автомобиль клиента (VIN, марка, модель, год выпуска).
- **Consultant** - сервис-консультант, оформляющий заказ-наряд.
- **Service_type** - вид обслуживания (например, замена масла, перестановка шин) с плановым количеством часов и

почасовой ставкой.
- **Service_order** - заказ-наряд, фиксирующий клиента, автомобиль, консультанта, дату обслуживания, статус и итоговую стоимость.
- **OrderService** - строки заказ-наряда: какие именно услуги входят в конкретный заказ.

Обоснование связей:
- Один **клиент** может владеть несколькими машинами, но каждая **машина** относится ровно к одному клиенту => связь *Client-Car* типа 1:N, внешний ключ client_id находится в таблице Car.
- Один **клиент** и одна **машина** могут иметь много заказ-нарядов, но каждый **заказ** относится к одному клиенту и одной машине => связь 1:N реализована внешними ключами client_id и car_id в Service_order.
- Один **консультант** ведёт множество заказов, но каждый заказ оформляет один консультант => связь *Consultant-Service_order* также 1:N, внешний ключ consultant_id хранится в Service_order.
- Между **Service_order** и **Service_type** связь *многие ко многим*: в одном заказе может быть несколько разных видов обслуживания, и один вид обслуживания используется во многих заказах. Для реализации этой связи введена отдельная таблица OrderService с составным первичным ключом (order_id, service_id).

Расчёт стоимости:
Для каждого вида обслуживания заранее задаются standard_hours и hourly_rate.
В таблице OrderService по ключам order_id и service_id можно хранить дополнительные данные по строке заказа (например, количество и сумму), а поле total_price в Service_order содержит общую стоимость заказ-наряда, вычисленную как сумму стоимостей всех входящих в него услуг.

DDL-скрипт:

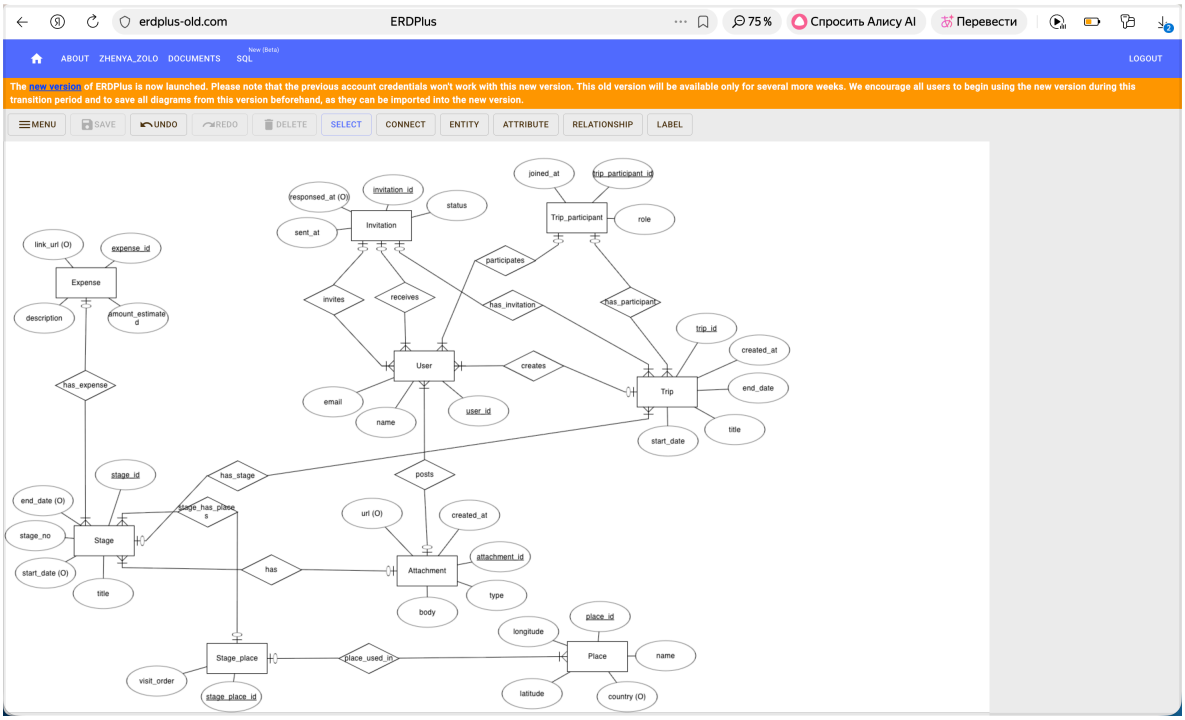
```
CREATE TABLE Client (
  client_id   INT NOT NULL,
  first_name  VARCHAR(50) NOT NULL,
  last_name   VARCHAR(50) NOT NULL,
  phone       VARCHAR(20) NOT NULL,
  email       VARCHAR(100) NOT NULL,
  PRIMARY KEY (client_id)
);
```

```
CREATE TABLE Car (
  car_id          INT NOT NULL,
  client_id       INT NOT NULL,
  VIN             VARCHAR(17) NOT NULL,
  make            VARCHAR(50) NOT NULL,
  model           VARCHAR(50) NOT NULL,
  production_year INT NOT NULL,
  PRIMARY KEY (car_id),
  FOREIGN KEY (client_id) REFERENCES Client(client_id)
);

CREATE TABLE consultant (
  consultant_id INT NOT NULL,
  first_name    VARCHAR(50) NOT NULL,
  last_name     VARCHAR(50) NOT NULL,
  phone         VARCHAR(20) NOT NULL,
  email         VARCHAR(100) NOT NULL,
  PRIMARY KEY (consultant_id)
);

CREATE TABLE service_type (
  service_id      INT NOT NULL,
  name            VARCHAR(100) NOT NULL,
  description     VARCHAR(255) NOT NULL,
  standard_hours  NUMERIC(5, 2) NOT NULL,
  hourly_rate     NUMERIC(10, 2) NOT NULL,
  PRIMARY KEY (service_id)
);

CREATE TABLE service_order (
  order_id      INT NOT NULL,
  client_id     INT NOT NULL,
  car_id        INT NOT NULL,
  consultant_id INT NOT NULL,
  order_date    DATE NOT NULL,
  status        VARCHAR(30) NOT NULL,
  total_price   NUMERIC(12, 2) NOT NULL,
  PRIMARY KEY (order_id),
  FOREIGN KEY (client_id)     REFERENCES Client(client_id),
  FOREIGN KEY (car_id)        REFERENCES Car(car_id),
  FOREIGN KEY (consultant_id) REFERENCES consultant(consultant_id)
);

CREATE TABLE OrderService (
  order_id   INT NOT NULL,
  service_id INT NOT NULL,
```

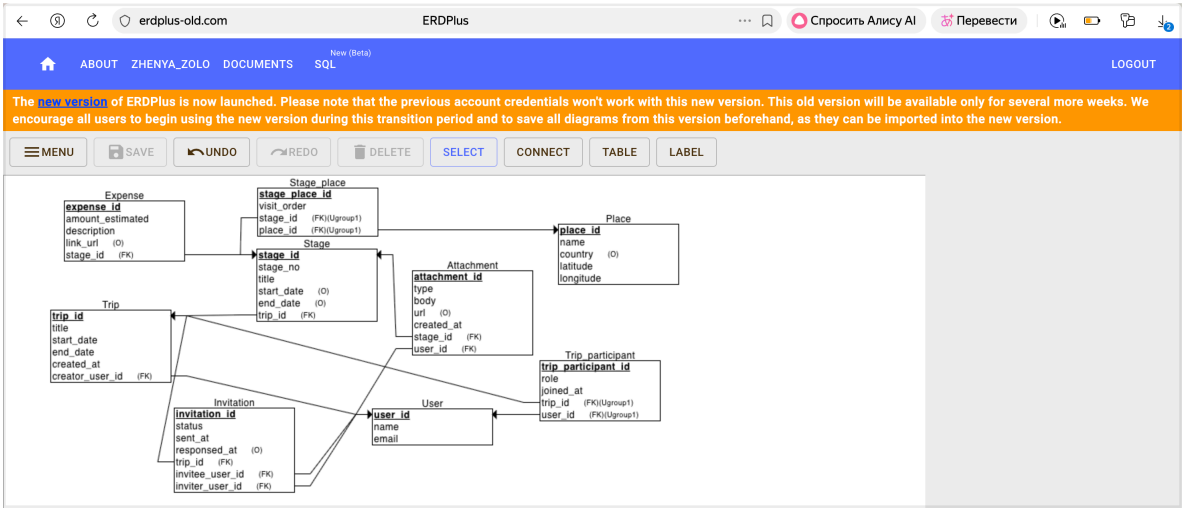
```
  PRIMARY KEY (order_id, service_id),
  FOREIGN KEY (order_id)   REFERENCES service_order(order_id),
  FOREIGN KEY (service_id) REFERENCES service_type(service_id)
);
```

Ответ на задание 4

Концептуальная схема:



Реляционная схема:



Пояснение к схеме.

В модели выделены ключевые сущности приложения: User (пользователь), Trip (поездка), Stage (этап), Place (место), Expense (расход), Invitation (приглашение), Trip_participant (участие) и Attachment (вложения к этапам). Пользователь создаёт поездку (Trip.creator_user_id => User.user_id), поездка состоит из этапов (Stage.trip_id => Trip.trip_id). Места вынесены в отдельную таблицу и переиспользуются в разных поездках, поэтому связь этап–место реализована как M:N через

Stage_place с порядком посещения (visit_order) и ограничением UNIQUE(stage_id, place_id) для исключения дублей. Расходы привязаны к этапу (Expense.stage_id), а вложения хранятся как записи разных типов (фото/ссылка/комментарий) и привязаны одновременно к этапу и автору (Attachment.stage_id, Attachment.user_id). Приглашения фиксируют отправителя/получателя и статус (Invitation), а факт участия хранится отдельно в Trip_participant (с ролью и датой присоединения) с ограничением UNIQUE(user_id, trip_id).

DDL-скрипт:

```
CREATE TABLE User
(
  user_id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL,
  PRIMARY KEY (user_id)
);

CREATE TABLE Trip
(
  trip_id INT NOT NULL,
  title VARCHAR(200) NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  created_at TIMESTAMP NOT NULL,
  creator_user_id INT NOT NULL,
  PRIMARY KEY (trip_id),
  FOREIGN KEY (creator_user_id) REFERENCES User(user_id)
);

CREATE TABLE Stage
(
  stage_id INT NOT NULL,
  stage_no INT NOT NULL,
  title VARCHAR(200) NOT NULL,
  start_date DATE,
  end_date DATE,
  trip_id INT NOT NULL,
  PRIMARY KEY (stage_id),
  FOREIGN KEY (trip_id) REFERENCES Trip(trip_id)
);

CREATE TABLE Place
(
  place_id INT NOT NULL,
  name VARCHAR(200) NOT NULL,
```

```
  country VARCHAR(100),
  latitude NUMERIC(9, 6) NOT NULL,
  longitude NUMERIC(9, 6) NOT NULL,
  PRIMARY KEY (place_id)
);

CREATE TABLE Stage_place
(
  stage_place_id INT NOT NULL,
  visit_order INT NOT NULL,
  stage_id INT NOT NULL,
  place_id INT NOT NULL,
  PRIMARY KEY (stage_place_id),
  FOREIGN KEY (stage_id) REFERENCES Stage(stage_id),
  FOREIGN KEY (place_id) REFERENCES Place(place_id),
  UNIQUE (stage_id, place_id)
);

CREATE TABLE Expense
(
  expense_id INT NOT NULL,
  amount_estimated NUMERIC(12, 2) NOT NULL,
  description VARCHAR(255) NOT NULL,
  link_url VARCHAR(2000),
  stage_id INT NOT NULL,
  PRIMARY KEY (expense_id),
  FOREIGN KEY (stage_id) REFERENCES Stage(stage_id)
);

CREATE TABLE Invitation
(
  invitation_id INT NOT NULL,
  status VARCHAR(20) NOT NULL,
  sent_at TIMESTAMP NOT NULL,
  responsed_at TIMESTAMP,
  trip_id INT NOT NULL,
  invitee_user_id INT NOT NULL,
  inviter_user_id INT NOT NULL,
  PRIMARY KEY (invitation_id),
  FOREIGN KEY (trip_id) REFERENCES Trip(trip_id),
  FOREIGN KEY (invitee_user_id) REFERENCES User(user_id),
  FOREIGN KEY (inviter_user_id) REFERENCES User(user_id)
);

CREATE TABLE Trip_participant
(
  trip_participant_id INT NOT NULL,
```

```
  role VARCHAR(200) NOT NULL,
  joined_at TIMESTAMP NOT NULL,
  trip_id INT NOT NULL,
  user_id INT NOT NULL,
  PRIMARY KEY (trip_participant_id),
  FOREIGN KEY (trip_id) REFERENCES Trip(trip_id),
  FOREIGN KEY (user_id) REFERENCES User(user_id),
  UNIQUE (user_id, trip_id)
);

CREATE TABLE Attachment
(
  attachment_id INT NOT NULL,
  type VARCHAR(20) NOT NULL,
  body TEXT NOT NULL,
  url VARCHAR(2000),
  created_at TIMESTAMP NOT NULL,
  stage_id INT NOT NULL,
  user_id INT NOT NULL,
  PRIMARY KEY (attachment_id),
  FOREIGN KEY (stage_id) REFERENCES Stage(stage_id),
  FOREIGN KEY (user_id) REFERENCES User(user_id)
);
```