📖 Общая логика БД

Эта база делится на несколько смысловых блоков:

👤 Пользователи и аккаунт

(users, profiles, account_levels, social_logins, login_history, notifications)
→ всё, что связано с регистрацией, входом, настройками и коммуникацией.

🚌 Поездки и бронирования

(routes, stops, segments, segment_transport, schedules, bookings, tickets, payments, booking_pricing, booking_details, bookings_history)
→ хранение маршрутов, расписаний, бронирований, билетов и оплат.

🎁 Лояльность и акции

(rewards, promo_rewards, loyalty_rewards, user_loyalty_balance)
→ промокоды, бонусные баллы и их использование.

📝 Логи и аудит

(audit_logs)
→ система фиксирует все важные действия (для безопасности и аналитики).

☆ Отзывы и медиа

(route_reviews, review_files, user_files)
→ отзывы на маршруты, файлы пользователей, вложения.

Как читать таблицы

Пример: `users`

- `users` — это главная таблица всех аккаунтов.
- В ней есть флаг активности (`is_active`) и верификация почты (`email_verified`).
- `profiles` хранит дополнительные данные (ФИО, день рождения, часовой пояс, уровень аккаунта).
- `account_levels` описывает статусы (Gold, Silver, Bronze).
- Если человек зашёл через Google или VK — запись попадёт в `social_logins`.
- Все его входы (успешные и неуспешные) логируются в `login_history`.
- Сообщения (уведомления, промо, напоминания) уходят в `notifications`.

Чтобы собрать профиль пользователя, делаем:
`users + profiles + account_levels + social_logins`

Пример: `bookings` (бронирование)

- Когда юзер покупает билет → создаётся запись в `bookings`.
- Финансовая часть (цены, скидки) хранится отдельно в `booking_pricing`.
- Детали билета (класс, багаж) — в `booking_details`.
- Сами билеты на пассажиров — в `tickets`.
- Оплаты проходят через `payments`.
- Все изменения статуса брони (создано, отменено, оплачено) фиксируются в `bookings_history`.

👉 Чтобы собрать историю брони:
`bookings + booking_pricing + tickets + payments + bookings_history`

Пример: `routes` (маршруты)

- `routes` — сам маршрут (например, «Москва–СПб»).
- Он состоит из `segments` (каждый отрезок пути с остановками).
- `segment_transport` уточняет перевозчика и стоимость сегмента.
- `schedules` привязывает сегменты к времени (отправление и прибытие).

👉 Если нужен полный маршрут с расписанием:
`routes + segments + segment_transport + schedules`

Пример: loyalty и promo

- `rewards` — базовая таблица наград (унифицированная).
- `promo_rewards` — условия промокодов.
- `loyalty_rewards` — правила лояльности (как начислять/списывать баллы).
- `user_loyalty_balance` — баланс пользователя по бонусам.

Пример: `audit_logs`

- Любое действие (создание брони, отмена, оплата, отзыв) фиксируется здесь.
- Указывается кто (`user_id`), что сделал (`action`), над какой сущностью (`entity_type` и `entity_id`), когда (`timestamp`), с какого устройства/IP.
- Используется для безопасности, расследований и аналитики.

Пример: отзывы и медиа

- `route_reviews` — текстовый отзыв и рейтинг.
- `review_files` — вложенные картинки/файлы к отзыву.
- `user_files` — любые загруженные файлы (привязка к брони или отзыву).

Если коротко:

- Всё, что связано с **человеком** → блок `users`
- Всё, что связано с **поездкой** → блок `routes/bookings`

- Всё, что связано с **деньгами и бонусами** → блок `payments/rewards`
- Всё, что связано с **контролем и безопасностью** → блок `audit_logs`
- Всё, что связано с **обратной связью** → блок `reviews + files`

Подробно о маршрутах:

Основные сущности

providers

- **Описание**: Перевозчики (автобусные компании, метро, такси и др.).
- **Назначение**: Определяет владельца маршрута.
- **Связи**:
  - 1 → N `routes` (один перевозчик может иметь несколько маршрутов).

---

routes

- **Описание**: Общая информация о маршруте (например, "Автобус №15").
- **Назначение**: Хранит базовые данные маршрута.
- **Связи**:
  - N → 1 `providers` (каждый маршрут принадлежит перевозчику).
  - 1 → N `segments` (маршрут состоит из сегментов).

---

stops

- **Описание**: Точки остановок/станций (с координатами).
- **Назначение**: Универсальная таблица для всех типов транспорта.
- **Связи**:
  - Используется в `segments` (начальная и конечная точка сегмента).
  - Может быть общей для разных маршрутов.

---

segments

- **Описание**: Участки маршрута от одной остановки до другой.
- **Назначение**: Делит маршрут на логические части.
- **Связи**:
  - N → 1 `routes` (каждый сегмент принадлежит маршруту).
  - N → 1 `stops` (stop_from_id — начало сегмента).
  - N → 1 `stops` (stop_to_id — конец сегмента).
  - 1 → 1 `segment_transport` (тип транспорта для сегмента).

- 1 → N `schedules` (расписание движения по сегменту).

---

segment_transport

- **Описание**: Указывает вид транспорта (bus, train, metro, taxi и т. д.) для сегмента.
- **Назначение**: Дает гибкость, так как разные сегменты одного маршрута могут выполняться разными типами транспорта.
- **Связи**:
  - 1 → 1 `segments`.

---

schedules

- **Описание**: Время отправления и прибытия для сегмента.
- **Назначение**: Определяет, когда транспорт следует по конкретному сегменту.
- **Связи**:
  - N → 1 `segments` (расписание относится к определенному сегменту).

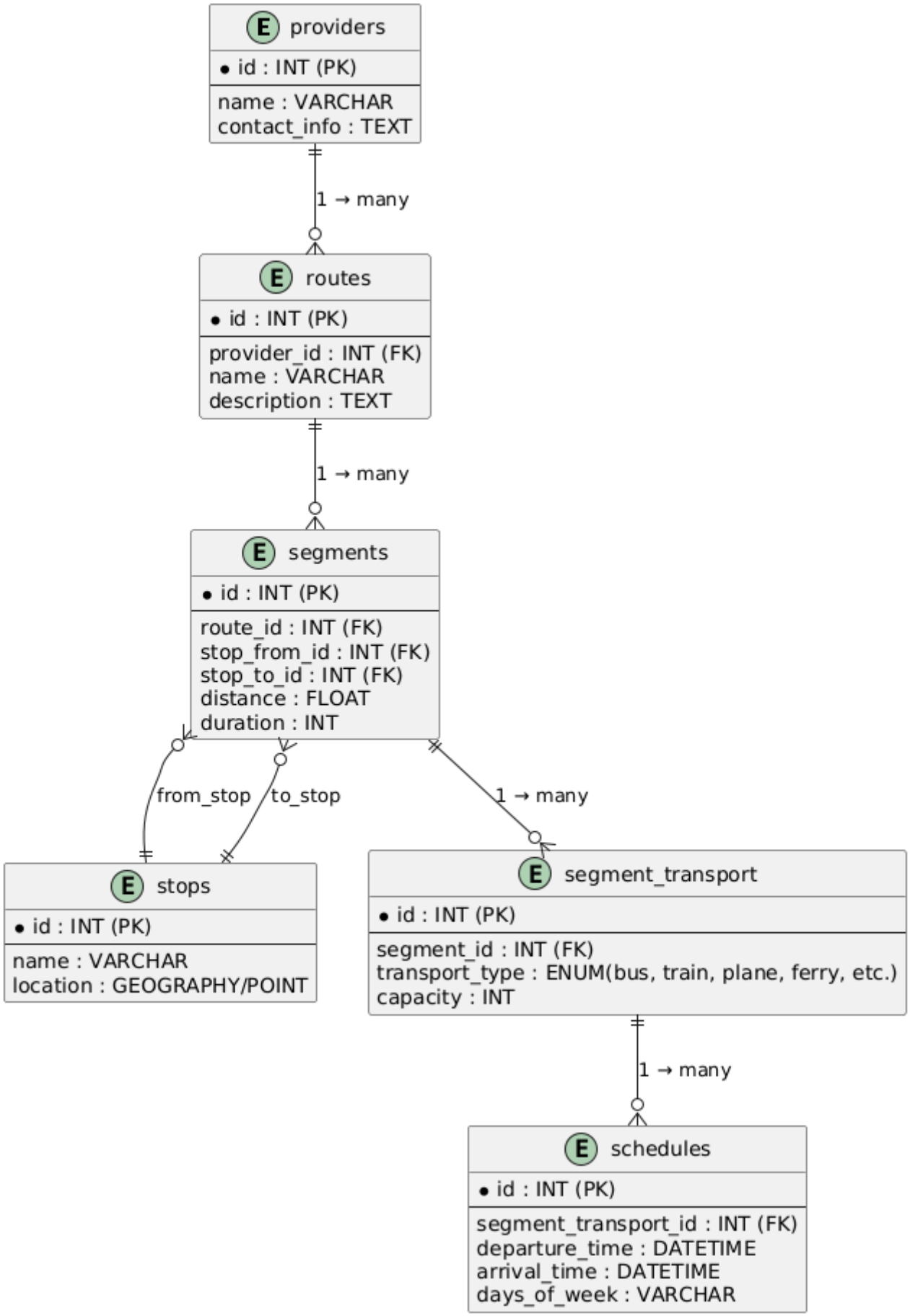
Общая логика работы

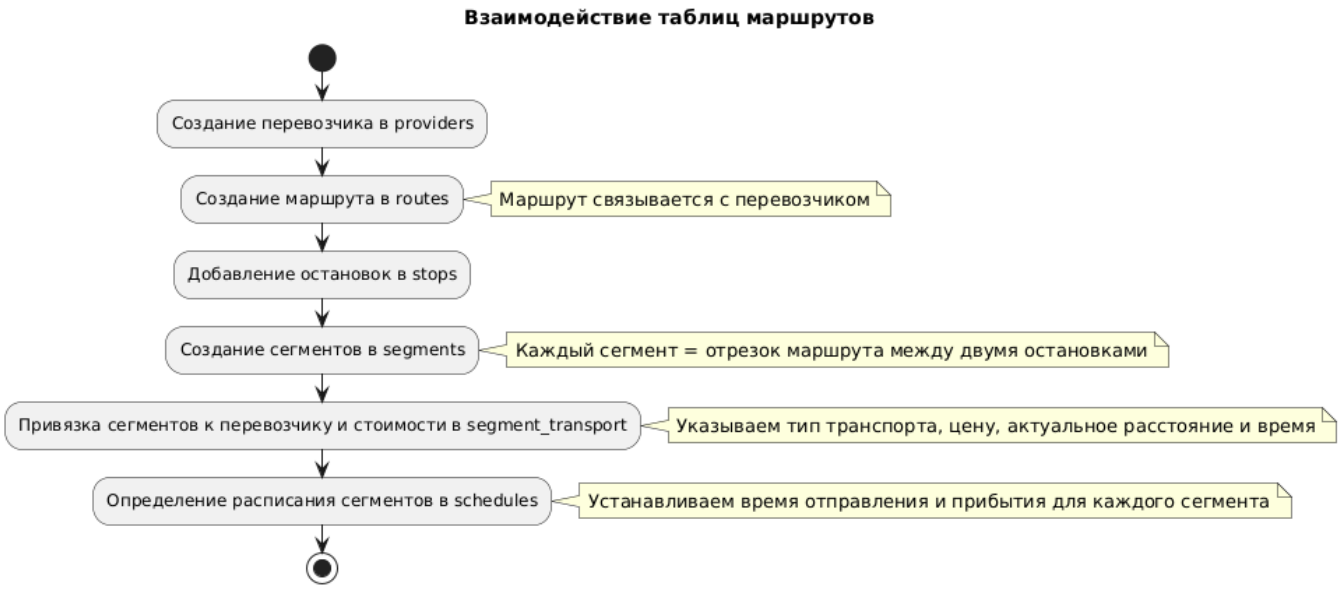
1. **Провайдер (providers)** регистрирует маршрут.
2. **Маршрут (routes)** создаётся и привязывается к провайдеру.
3. **Остановки (stops)** используются как точки начала и конца сегментов.
4. **Сегменты (segments)** описывают части маршрута между остановками.
5. Для каждого сегмента определяется **тип транспорта (segment_transport)**.
6. Для каждого сегмента назначается **расписание (schedules)**:
   - время отправления,
   - время прибытия,
   - периодичность (если есть).

Пример использования

- Перевозчик **"Городской автобус"** создаёт маршрут **"№15"**.
- Маршрут делится на сегменты:
  - Сегмент 1: `Остановка A → Остановка B`.
  - Сегмент 2: `Остановка B → Остановка C`.
- У сегмента 1 `segment_transport = bus`.
- У сегмента 2 `segment_transport = tram`.
- Для каждого сегмента вносятся записи в **schedules** (например, каждые 15 минут с 6:00 до 23:00).

Схема связей для построения маршрутов


providers
• id : INT (PK)
name : VARCHAR
contact_info : TEXT
1 → many
routes
• id : INT (PK)
provider_id : INT (FK)
name : VARCHAR
description : TEXT
1 → many
segments
• id : INT (PK)
route_id : INT (FK)
stop_from_id : INT (FK)
stop_to_id : INT (FK)
distance : FLOAT
duration : INT
from_stop
to_stop
1 → many
stops
• id : INT (PK)
name : VARCHAR
location : GEOGRAPHY/POINT
segment_transport
• id : INT (PK)
segment_id : INT (FK)
transport_type : ENUM(bus, train, plane, ferry, etc.)
capacity : INT
1 → many
schedules
• id : INT (PK)
segment_transport_id : INT (FK)
departure_time : DATETIME
arrival_time : DATETIME
days_of_week : VARCHAR



Бизнес-логика: Бронирование и Оплата

Таблицы и их роль

bookings

- Центральная сущность, которая фиксирует факт бронирования.
- Хранит базовую информацию:
  - `id` — уникальный идентификатор.
  - `user_id` — кто забронировал.
  - `created_at` — когда создано.
  - `status` — текущее состояние (draft, pending, confirmed, cancelled, completed).
- Является точкой входа для всех связанных данных.

---

booking_pricing

- Хранит расчётную стоимость бронирования.
- Может содержать:
  - `base_price` — базовый тариф.
  - `discount` — применённые скидки.
  - `loyalty_spent` — сколько бонусов использовано.
  - `loyalty_earned` — сколько начислено бонусов.
  - `final_price` — итоговая стоимость для оплаты.
- Связан с `bookings (1:1)`.

---

booking_details

- Детализирует каждое место / сегмент маршрута внутри бронирования.
- Примеры полей:

    - `booking_id` — ссылка на бронирование.
    - `segment_id` — сегмент маршрута (из `segments`).
    - `seat_number` — место (если фиксированное).
    - `passenger_name`, `passenger_doc` — данные пассажира.
- Связь: `bookings (1:M)`.

---

tickets

- Финальная сущность после подтверждения оплаты.
- Содержит уникальный билет для каждого сегмента/места.
- Поля:
    - `ticket_number` — уникальный код.
    - `qr_code` / `barcode`.
    - `status` (active, refunded, expired).
- Связан с `booking_details`.

---

payments

- Хранит информацию об оплате бронирования.
- Поля:
    - `booking_id` — для какого бронирования.
    - `amount` — сумма.
    - `payment_method` (card, wallet, cash).
    - `status` (pending, success, failed, refunded).
- Логика:
    - После успешной оплаты → статус `bookings` меняется на *confirmed*.
    - При возврате → создаётся запись с типом `refund`.

---

rewards

- Базовая таблица для всех вознаграждений.
- Поля:
    - `id`, `type` (promo | loyalty), `title`, `description`.
- Является «шлюзом» к специализированным таблицам.

---

promo_rewards

- Купоны, акции.
- Поля:
    - `code` — промокод.
    - `valid_from` / `valid_to`.
    - `usage_limit`.
- При применении:
    - Проверяется валидность.
    - Сумма скидки отражается в `booking_pricing.discount`.

---

loyalty_rewards

- Правила начисления и списания бонусов.
- Поля:
  - `earn_rate` — сколько бонусов за 1₽.
  - `spend_rate` — коэффициент списания бонусов.
- Используется при формировании `booking_pricing`.

---

bookings_history

- Бизнес-лог: фиксирует изменения по бронированию.
- Примеры:
  - смена статуса,
  - отмена,
  - применение скидки,
  - возврат средств.
- Позволяет восстановить весь путь бронирования.

---

user_files

- Хранение загруженных пользователем документов.
- Используется для:
  - верификации личности,
  - прикрепления билетов,
  - возвратов и жалоб.
- Связан с пользователями и конкретными бронированиями.

Общая логика процесса

1. **Создание бронирования**
   - Пользователь выбирает маршрут и сегменты.
   - Создаётся запись в `bookings` со статусом *draft*.
   - Добавляются `booking_details`.
2. **Расчёт цены**
   - Считается базовый тариф.
   - Применяются `promo_rewards` (если введён промокод).
   - Применяются `loyalty_rewards` (списание бонусов).
   - Итог пишется в `booking_pricing`.
3. **Оплата**
   - Создаётся запись в `payments (pending)`.
   - При успешной оплате:

- `payments.status = success`.
- `bookings.status = confirmed`.
- Генерация `tickets`.

4. **Лояльность**

   - Если в правилах начисления предусмотрено:
     - После завершения поездки → начисление бонусов (`loyalty_rewards`).
   - Баланс хранится в `user_loyalty_balance`.

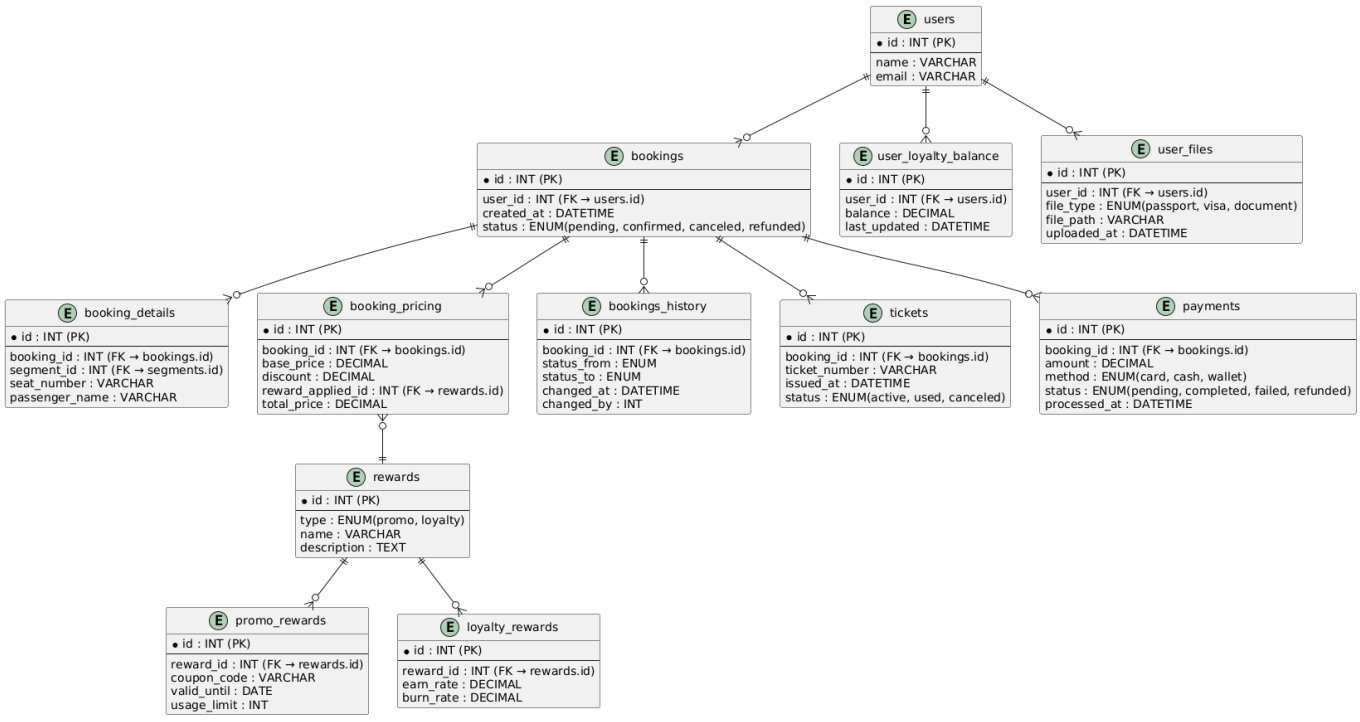
5. **История**

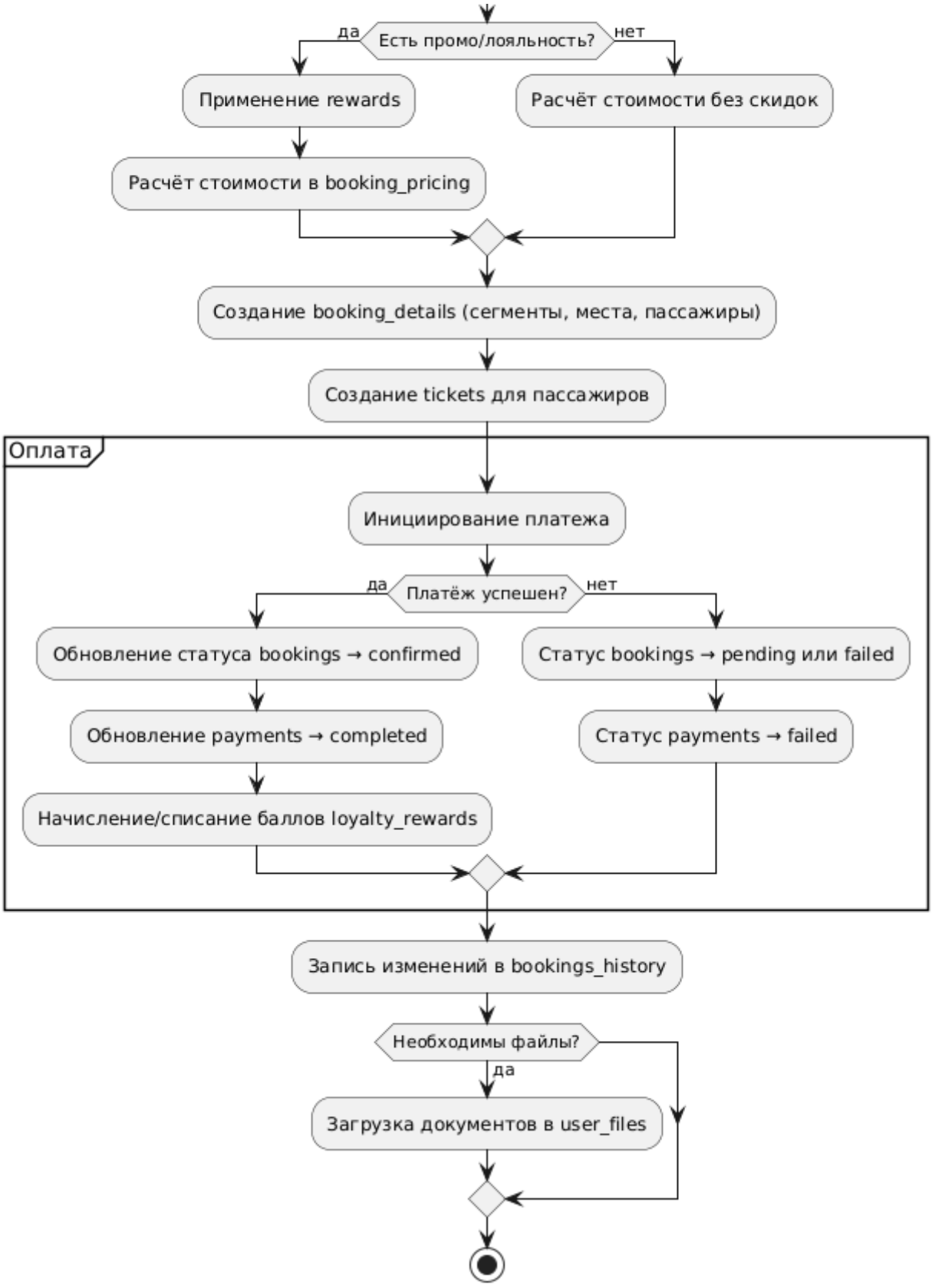
   - Любое изменение (отмена, оплата, возврат, применение купона) → запись в `bookings_history`.

6. **Возврат**

   - При отмене бронирования → создаётся `payments` с типом refund.
   - `tickets` меняют статус на *refunded*.
   - В `bookings_history` фиксируется отмена.

Схема связей для бронирования и оплаты





Бизнес-логика работы с пользователями, аккаунтами и историей

В этом блоке описаны таблицы, связанные с пользователями: их аккаунты, входы, уровень, бонусы и история маршрутов.

users

Главная таблица всех пользователей.
Хранит базовую информацию:

- email, пароль (hash),
- активность аккаунта (`is_active`),
- дата создания и обновления,
- последние успешные и неудачные входы (`last_login`, `failed_attempts`),
- верификация email (`email_verified`).

---

profiles

Содержит дополнительные данные о пользователях:

- Полное имя (`full_name`),
- Дата рождения (`birthday`),
- Часовой пояс (`timezone`),
- Настройки аккаунта в формате JSON (`preferences`),
- Уровень аккаунта (`account_level`) → FK к `account_levels`.

Связь: 1 к 1 с `users`.

---

account_levels

Таблица уровней аккаунтов:

- Gold, Silver, Bronze и т.д.,
- описание преимуществ и бонусный процент (`bonus_percent`).

Используется в `profiles` для присвоения уровня пользователю.

---

user_loyalty_balance

Баланс бонусных баллов пользователя:

- сколько баллов есть на счёте (`points_balance`),
- когда обновлялся последний раз (`last_updated`).

Связь: FK к `users`.
Баллы начисляются или списываются через `loyalty_rewards` при бронированиях или акциях.

---

user_routes

История поездок и выбранных маршрутов пользователем:

- Тип маршрута (`from_to` или `to_only`),
- Начальная и конечная остановка (`start_stop`, `end_stop`),

- Дата выбора маршрута (`created_at`),
- Время и расстояние сегмента.

Связь: FK к `users` и остановкам (`stops`).

---

login_history

История всех входов пользователя:

- Время входа (`login_at`),
- IP-адрес, устройство, браузер,
- Успешность входа (`success`).

Связь: FK к `users`.
Позволяет отслеживать активность и безопасность.

---

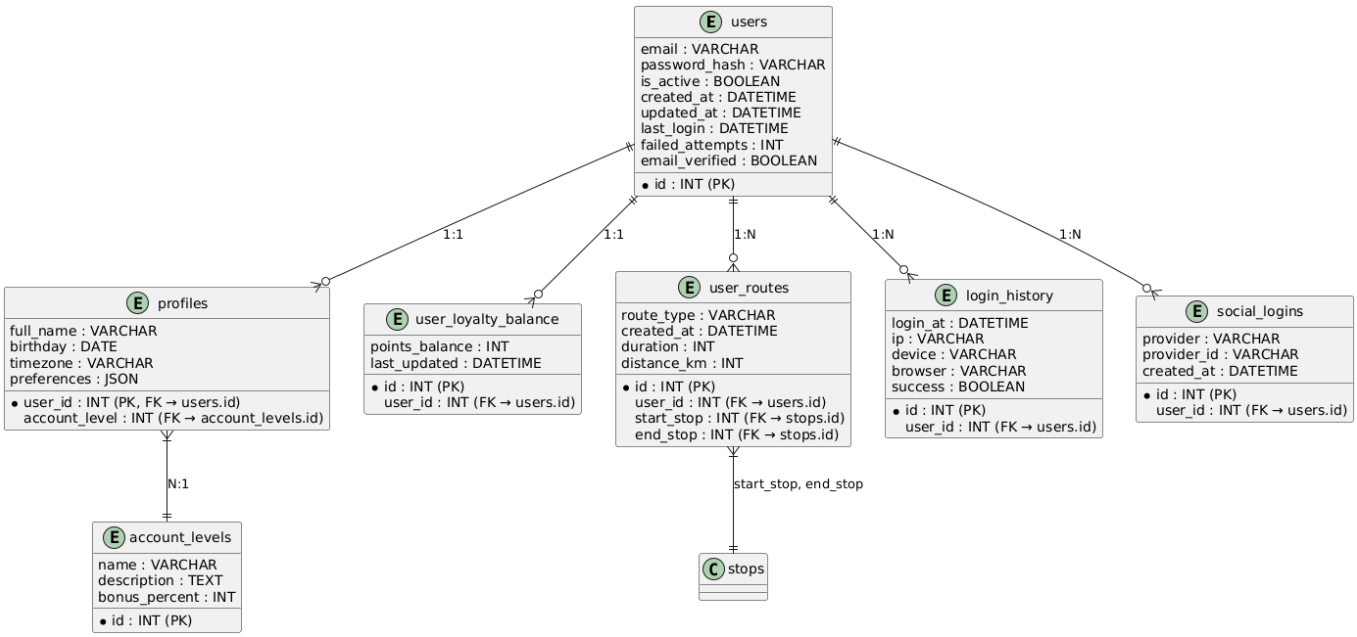
social_logins

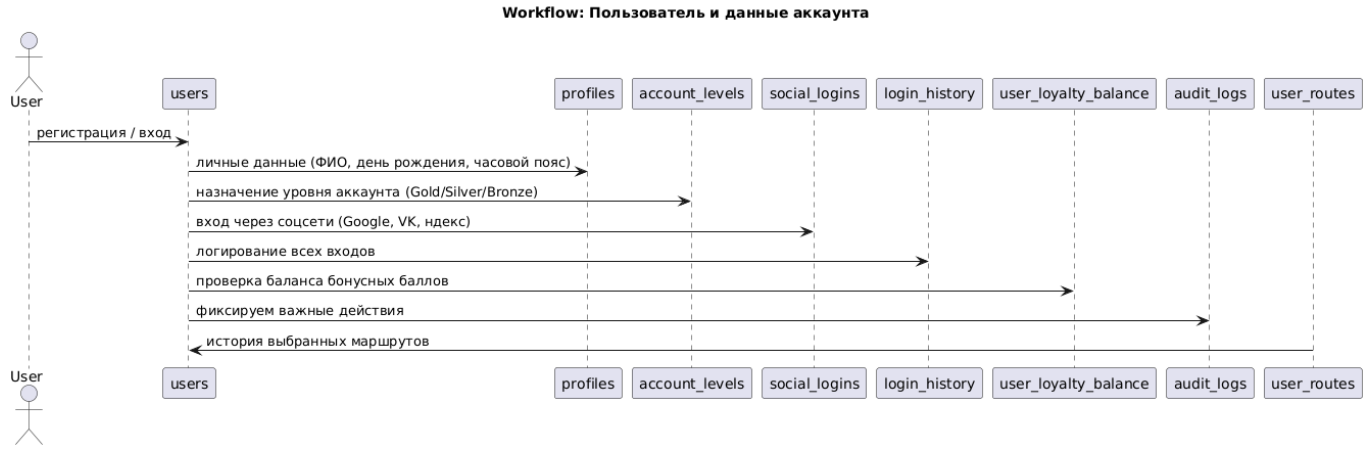
Вход через внешние сервисы:

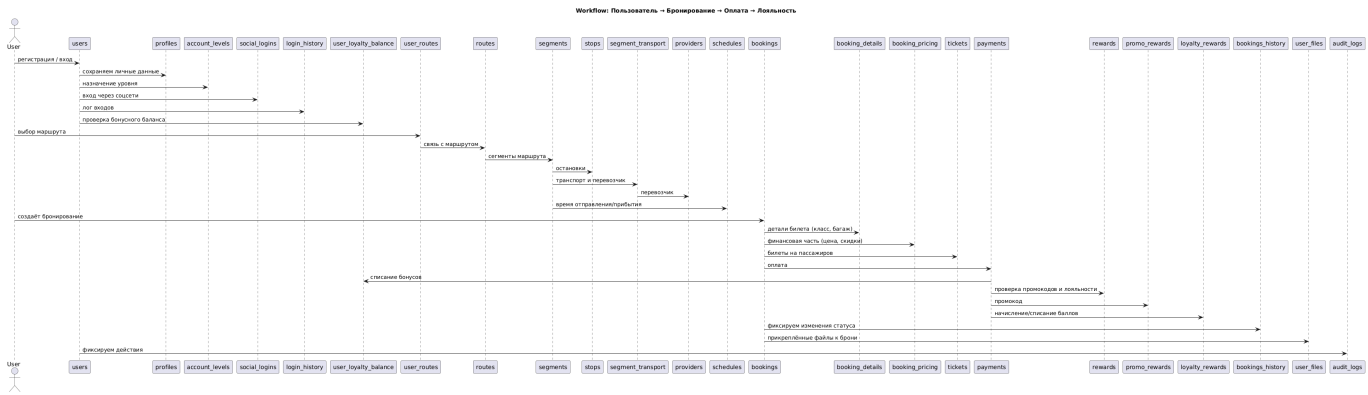
- Название сервиса (`provider`),
- ID пользователя в сервисе (`provider_id`),
- Дата подключения (`created_at`).

Связь: FK к `users`.
Используется для авторизации через Google, VK, Яндекс и др.

Схема связей для описания пользователя




Workflow: Пользователь и данные аккаунта
User
users
profiles
account_levels
social_logins
login_history
user_loyalty_balance
audit_logs
user_routes
регистрация / вход
личные данные (ФИО, день рождения, часовой пояс)
назначение уровня аккаунта (Gold/Silver/Bronze)
вход через соцсети (Google, VK, ндекс)
логирование всех входов
проверка баланса бонусных баллов
фиксируем важные действия
история выбранных маршрутов


Workflow: Пользователь → Бронирование → Оплата → Лояльность
регистрация / вход
выбор маршрута
создаёт бронирование
списание бонусов
фиксируем действия