

Задания для выполнения контрольной работы №1
по дисциплине
«Программирование аналитических приложений на
базе хранилищ данных»

Для обучающихся заочной формы обучения
Кафедра «Математика и информатика»
Донской государственный технический университет

2026

Содержание

Общие положения	3
Варианты контрольных работ	4
Варианты заданий	5
Вариант 1. Интернет-магазин (электронная коммерция)	5
Вариант 2. Сеть супермаркетов (розничная торговля)	10
Вариант 3. Онлайн-образование (платформа курсов)	15
Вариант 4. Банк (кредитование)	20
Вариант 5. Транспортная компания (грузоперевозки)	25
Вариант 6. Производственное предприятие (учёт деталей)	29
Вариант 7. Телеком-оператор (учёт звонков)	33
Вариант 8. Больница (учёт пациентов и посещений)	37
Вариант 9. Гостиница (бронирование номеров)	40
Вариант 10. Кинотеатр (продажа билетов)	43
Вариант 11. Авиакомпания (рейсы, бронирования)	47
Вариант 12. HR-система (сотрудники, вакансии)	51
Вариант 13. Склад (учёт товаров)	54
Вариант 14. Логистический центр (заказы, доставка)	58
Вариант 15. Энергоснабжающая компания (показания счетчиков)	62
Вариант 16. Недвижимость (продажа/аренда)	66
Вариант 17. Агентство недвижимости (с агентами)	70
Вариант 18. Рекламное агентство (кампании)	73
Вариант 19. Социальная сеть (пользователи, посты)	76
Вариант 20. Спортивный клуб (членство, тренировки)	80
Вариант 21. Ресторан (заказы блюд)	84
Вариант 22. Библиотека (книги, читатели)	88
Вариант 23. Автомастерская (ремонт)	92
Вариант 24. Страховая компания (полисы)	96
Вариант 25. Агрохолдинг (поля, урожай)	100

Вариант 26. Муниципалитет (население, платежи)	103
Вариант 27. Управление проектами (задачи)	107
Вариант 28. Фриланс-платформа (заказы, исполнители)	111
Вариант 29. Стоматологическая клиника (пациенты, услуги)	115
Вариант 30. Университет (студенты, курсы, оценки)	119

Общие положения

Контрольная работа является одним из основных видов самостоятельной работы обучающихся и важным этапом их профессиональной подготовки. Основные цели выполнения работы: расширение и углубление знаний, выработка навыков анализа теоретического и практического материала, обучение логичному и последовательному изложению мыслей в письменной форме. В ходе выполнения контрольной работы студент должен продемонстрировать умение работать с литературой, анализировать источники, аргументировать выводы и раскрывать выбранную тему.

По дисциплине «Программирование аналитических приложений на базе хранилищ данных» контрольная работа выполняется в форме проектного кейса. Студент получает вариант, соответствующий начальной букве его фамилии (см. таблицу вариантов). Каждый вариант содержит описание бизнес-кейса из определённой предметной области и четыре практических задания:

1. Разработать ER-диаграмму (логическую модель) из 3–5 таблиц и написать SQL-скрипт на языке PostgreSQL для создания базы данных.
2. Разработать SQL-скрипты для добавления, извлечения, изменения и удаления данных (CRUD-операции) для созданной структуры.
3. Разработать ETL-процесс: описать этапы извлечения данных из источников, трансформации и загрузки в хранилище или витрину данных.
4. Разработать дашборд или отчёт: предложить набор визуализаций, ключевые показатели (KPI), фильтры и срезы для поддержки принятия решений.

При выполнении работы необходимо соблюдать следующие требования:

- Титульный лист с указанием варианта.
- Текст должен быть написан в редакторе Word (или LaTeX), шрифт Times New Roman, кегль 14, полуторный интервал, выравнивание по ширине, поля по 2 см.
- Все таблицы должны иметь подстрочную ссылку на источник (при необходимости), быть пронумерованы и иметь названия.
- Части работы должны быть озаглавлены, страницы пронумерованы.
- Работа завершается списком использованных источников и приложений (SQL-скрипты, рисунки ER-диаграмм и макетов дашбордов).

Контрольная работа защищается устно. По результатам защиты выставляется оценка «зачтено» или «не зачтено» в соответствии с критериями, приведёнными в методических указаниях.

Варианты контрольных работ

Таблица 1: Соответствие начальной буквы фамилии и номера варианта

Начальная буква фамилии	Номер варианта	Начальная буква фамилии	Номер вар
А	1	П	16
Б	2	Р	17
В	3	С	18
Г	4	Т	19
Д	5	У	20
Е	6	Ф	21
Ё	7	Х	22
Ж	8	Ц	23
З	9	Ч	24
И	10	Ш	25
К	11	Щ	26
Л	12	Э	27
М	13	Ю	28
Н	14	Я	29
О	15	-	-

Варианты заданий

Вариант 1. Интернет-магазин (электронная коммерция)

Описание кейса

Интернет-магазин продаёт товары различных категорий (электроника, одежда, книги, бытовая техника и т.п.). Клиенты регистрируются на сайте, просматривают товары, оформляют заказы, оставляют отзывы и оценки. Необходимо вести учёт товаров, заказов, клиентов и отзывов для последующего анализа. Бизнес-цели:

- Анализ продаж по категориям и подкатегориям товаров.
- Оценка популярности товаров (по количеству заказов, выручке, отзывам).
- Сегментация клиентов по частоте покупок, сумме покупок, среднему чеку.
- Мониторинг динамики продаж (по дням, неделям, месяцам, сезонам).
- Оценка эффективности маркетинговых кампаний (если есть данные об источнике привлечения).

Для поддержки принятия решений необходимо создать хранилище данных, разработать ETL-процессы и построить дашборд для менеджера интернет-магазина.

В рамках выполнения задания необходимо:

1. Спроектировать структуру базы данных (ER-диаграмму).
2. Реализовать SQL-скрипты для создания таблиц и выполнения CRUD-операций.
3. Разработать ETL-процесс для построения витрины данных.
4. Предложить макет дашборда с ключевыми показателями и визуализациями.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Разработайте логическую модель базы данных (ER-диаграмму) для интернет-магазина, включающую как минимум следующие таблицы:

- **customers** — клиенты (идентификатор, имя, фамилия, email, телефон, дата регистрации, статус);
- **products** — товары (идентификатор, название, категория, цена, остаток на складе, дата добавления);
- **orders** — заказы (идентификатор, идентификатор клиента, дата заказа, общая сумма, статус);
- **order_items** — позиции заказа (идентификатор, идентификатор заказа, идентификатор товара, количество, цена на момент покупки);
- **reviews** — отзывы (идентификатор, идентификатор товара, идентификатор клиента, оценка от 1 до 5, текст, дата).

Для каждой таблицы определите:

- первичные ключи (PRIMARY KEY);

- внешние ключи (FOREIGN KEY) с указанием связей (один-ко-многим);
- ограничения целостности (NOT NULL, UNIQUE, CHECK, DEFAULT).

Напишите SQL-скрипт на языке PostgreSQL для создания всех таблиц в правильном порядке (с учётом зависимостей по внешним ключам). Добавьте комментарии к таблицам и столбцам с использованием команды `COMMENT ON`. Предусмотрите ограничения, например:

- цена товара должна быть положительной (`CHECK (price > 0)`);
- оценка в отзыве должна быть в диапазоне от 1 до 5 (`CHECK (rating BETWEEN 1 AND 5)`);
- статус заказа должен быть из допустимого списка (`CHECK (status IN ('new', 'processing', 's`
- email клиента должен быть уникальным (`UNIQUE`).

Приложите ER-диаграмму (можно в виде текстового описания связей или рисунка). Например, опишите связи:

- `orders` связан с `customers` по полю `customer_id` (один клиент — много заказов);
- `order_items` связан с `orders` по `order_id` (один заказ — много позиций);
- `order_items` связан с `products` по `product_id` (один товар — много позиций);
- `reviews` связан с `products` по `product_id` (один товар — много отзывов);
- `reviews` связан с `customers` по `customer_id` (один клиент — много отзывов).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций с данными:

1. **Добавление нового клиента.** Напишите запрос, который вставляет нового клиента с указанием имени, email, телефона и даты регистрации. Предусмотрите, что email должен быть уникальным.
2. **Обновление цены товара.** Напишите запрос, который изменяет цену товара по его идентификатору. Проверьте, что новая цена положительная.
3. **Удаление заказа (каскадно).** Напишите запрос, который удаляет заказ по его идентификатору вместе со всеми связанными позициями (`order_items`). Используйте каскадное удаление или явное удаление позиций перед удалением заказа.
4. **Извлечение списка заказов клиента за последний месяц с общей суммой.** Напишите запрос, который возвращает для заданного клиента все его заказы, сделанные за последние 30 дней, с указанием даты заказа, статуса и общей суммы. Отсортируйте по дате убывания.
5. **Расчёт общей выручки по категориям за период.** Напишите запрос для расчёта общей выручки по каждой категории товаров за заданный период (дата начала и дата окончания). Выведите категорию и сумму продаж. (Подсказка: используйте соединение `order_items` с `products` и `orders`.)

6. **Поиск клиентов с наибольшим количеством заказов.** Напишите запрос, который возвращает топ-10 клиентов по количеству заказов, с указанием их имени и email.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции (BEGIN/COMMIT).

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для загрузки данных в витрину данных «Продажи по дням и категориям», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому дню и категории товаров:

- `sale_date` — дата продажи (из `orders.order_date`);
- `category` — категория товара (из `products.category`);
- `total_revenue` — общая выручка за день по данной категории (сумма `order_items.quantity * order_items.price`);
- `total_orders` — количество заказов, в которых присутствовали товары данной категории (уникальные `order_id`);
- `total_items_sold` — общее количество проданных единиц товара в данной категории (сумма `order_items.quantity`);
- `avg_price` — средняя цена продажи товаров в данной категории (средняя `order_items.price`).

Опишите ETL-процесс поэтапно:

1. **Извлечение (Extract).** Укажите, из каких таблиц и какие данные извлекаются. Как обрабатываются инкрементальные изменения (только новые заказы) или выполняется полная перезагрузка? Предложите использовать `order_date` для фильтрации только новых записей.
2. **Трансформация (Transform).** Опишите все необходимые преобразования:
 - соединение таблиц (`orders`, `order_items`, `products`);
 - фильтрация: исключить отменённые заказы (`orders.status != 'cancelled'`);
 - денормализация: выборка нужных полей;
 - расчёт агрегатов: группировка по дате и категории;
 - обработка NULL-значений и приведение типов.
3. **Загрузка (Load).** Определите стратегию загрузки данных в витрину:
 - полная перезагрузка (`TRUNCATE + INSERT`) или инкрементальная (`INSERT ON CONFLICT / MERGE`);
 - как обрабатываются обновления старых заказов (если они меняются);
 - как обеспечивается качество данных (проверка на отрицательные суммы, контроль количества строк).
4. **Оркестрация.** Как часто будет запускаться процесс (ежедневно, по расписанию)? Какие инструменты вы бы использовали для его реализации (например, `psql` скрипты с `cron`, Apache Airflow, Python + SQLAlchemy)?

Приведите псевдокод или SQL-фрагменты основных операций (например, запрос для инкрементальной загрузки новых данных).

Задание 4. Дашборд

Разработайте макет дашборда для менеджера интернет-магазина, который позволит отслеживать ключевые показатели эффективности (KPI) в реальном времени и принимать оперативные решения. Дашборд должен быть интерактивным, удобным для анализа и визуально понятным.

Состав дашборда:

1. KPI-панель (верхняя часть):

- Общая выручка за выбранный период.
- Количество заказов.
- Средний чек (выручка / количество заказов).
- Конверсия (отношение числа заказов к числу уникальных посетителей — если есть данные о посещениях, либо можно использовать отношение к числу клиентов).
- Процент отменённых заказов.

2. Динамика продаж (график). Линейный график, показывающий изменение общей выручки и количества заказов по дням (или неделям) за выбранный период. С возможностью переключения между разными временными агрегациями (день/неделя/месяц).

3. Распределение продаж по категориям. Круговая диаграмма или столбчатая диаграмма, показывающая долю выручки каждой категории товаров в общей выручке за период. При наведении отображается точный процент и сумма.

4. Топ-5 товаров по выручке. Таблица или гистограмма, отображающая пять товаров, принёсших наибольшую выручку за период, с указанием названия товара, категории, количества проданных единиц и общей суммы.

5. Распределение заказов по статусам. Круговая диаграмма, отображающая долю заказов в различных статусах (новые, в обработке, отправлены, доставлены, отменены).

6. Список последних заказов. Таблица с 5–10 последними заказами, содержащая дату, клиента, сумму и статус для быстрого мониторинга.

7. Фильтры и элементы управления:

- Фильтр по датам (выбор периода: день, неделя, месяц, произвольный интервал).
- Фильтр по категориям товаров (мультивыбор).
- Фильтр по статусу заказа.

Все визуализации должны быть связаны между собой: при выборе категории на круговой диаграмме все остальные элементы (график динамики, топ-товары) должны фильтроваться соответственно.

Визуализации:

- **Линейный график** — для отображения динамики продаж во времени.
- **Столбчатая диаграмма** — для сравнения показателей по категориям или товарам.

- **Круговая диаграмма** — для показа структуры продаж по категориям и статусам.
- **Таблица** — для детального отображения списков (топ-товары, последние заказы).
- **Карточки KPI** — для ключевых показателей.

Опишите, из каких источников данных будут строиться визуализации (из созданной витрины `sales_daily` или напрямую из операционной базы). Предложите инструмент для реализации дашборда (например, Power BI, Tableau, Yandex DataLens или Grafana) и обоснуйте выбор. Приложите эскиз дашборда (можно в виде текстового описания компоновки или рисунка).

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 2. Сеть супермаркетов (розничная торговля)

Описание кейса

Сеть супермаркетов объединяет несколько розничных магазинов, расположенных в разных районах города или региона. Каждый магазин ежедневно осуществляет продажи товаров, а также получает новые поставки от различных поставщиков. Товары классифицируются по категориям, имеют закупочную и розничную цены. Руководство сети хочет анализировать эффективность каждого магазина, выявлять наиболее популярные товары и категории, отслеживать сезонные колебания спроса, оценивать маржинальность продаж и управлять ассортиментом. Бизнес-цели:

- Мониторинг выручки и количества продаж по каждому магазину в динамике.
- Сравнение эффективности магазинов (по выручке, маржинальности, оборачиваемости товаров).
- Выявление товаров-лидеров и товаров с низкой оборачиваемостью.
- Анализ сезонности продаж по категориям и магазинам.
- Оценка работы поставщиков (надёжность, цены, ассортимент).

Для поддержки принятия решений необходимо создать хранилище данных, разработать ETL-процессы и построить дашборд для руководителя сети.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для сети супермаркетов, включающую как минимум следующие таблицы:

- **stores** — магазины (идентификатор, название, адрес, контактный телефон);
- **products** — товары (идентификатор, название, категория, цена закупки, цена продажи, единица измерения);
- **suppliers** — поставщики (идентификатор, название, контактное лицо, телефон, email);
- **deliveries** — поставки (идентификатор, идентификатор магазина, идентификатор товара, идентификатор поставщика, количество, цена закупки на момент поставки, дата поставки);
- **sales** — продажи (идентификатор, идентификатор магазина, идентификатор товара, количество, цена продажи на момент продажи, дата продажи, сумма (вычисляемая или вручную)).

Для каждой таблицы определите:

- первичные ключи (PRIMARY KEY);
- внешние ключи (FOREIGN KEY) с указанием связей (один-ко-многим);
- ограничения целостности (NOT NULL, UNIQUE, CHECK, DEFAULT).

Напишите SQL-скрипт на языке PostgreSQL для создания всех таблиц в правильном порядке (с учётом зависимостей по внешним ключам). Добавьте комментарии к таблицам и столбцам с использованием команды `COMMENT ON`. Предусмотрите ограничения, например:

- цена закупки и цена продажи должны быть положительными (`CHECK (purchase_price > 0 AND sale_price > 0)`);
- количество в поставках и продажах должно быть больше нуля (`CHECK (quantity > 0)`);
- даты поставок и продаж не могут быть в будущем (можно использовать `DEFAULT CURRENT_DATE` или `CHECK` на уровне приложения);
- сумма продажи может вычисляться как `quantity * sale_price` (можно использовать генерацию).

Опишите связи между таблицами:

- `deliveries` связана с `stores` (один магазин — много поставок),
- `deliveries` связана с `products` (один товар — много поставок),
- `deliveries` связана с `suppliers` (один поставщик — много поставок),
- `sales` связана с `stores` (один магазин — много продаж),
- `sales` связана с `products` (один товар — много продаж).

Приложите ER-диаграмму (можно в виде текстового описания или рисунка).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций с данными:

1. **Добавление нового магазина.** Напишите запрос, который вставляет новый магазин с указанием названия, адреса и телефона.
2. **Изменение цены продажи товара.** Напишите запрос, который обновляет цену продажи конкретного товара по его идентификатору. Проверьте, что новая цена положительная.
3. **Удаление поставщика при отсутствии поставок.** Напишите запрос, который удаляет поставщика только в том случае, если на него нет ссылок в таблице `deliveries`. Используйте подзапрос с `NOT EXISTS`.
4. **Выбор всех продаж за конкретный день с указанием магазина и товара.** Напишите запрос, который возвращает для заданной даты список продаж с полями: название магазина, название товара, количество, цена, сумма. Отсортируйте по магазину и товару.
5. **Расчёт общей выручки и маржи по магазинам за период.** Напишите запрос для вычисления для каждого магазина за заданный интервал дат общей выручки, общей себестоимости (по закупочным ценам) и маржи (выручка - себестоимость). (Подсказка: себестоимость можно взять из таблицы `products`, но для исторической точности лучше хранить цену закупки в `sales` или использовать поставки.)
6. **Топ-5 товаров по количеству продаж за месяц.** Напишите запрос, который возвращает пять товаров с наибольшим суммарным количеством проданных единиц за последний месяц.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения ежемесячной витрины данных «Продажи по магазинам и категориям», которая обновляется по окончании каждого месяца. Целевая витрина должна содержать следующие показатели по каждому магазину и категории товаров за отчётный месяц:

- `store_id`, `store_name`;
- `category` — категория товара;
- `report_month` — месяц отчёта (например, '2025-01');
- `total_revenue` — общая выручка от продаж товаров данной категории в этом магазине за месяц (сумма `sales.amount`);
- `total_quantity` — общее количество проданных единиц;
- `total_cost` — общая себестоимость проданных товаров (сумма `quantity * purchase_price` из поставок, либо из справочника `products`, если цены не менялись);
- `gross_margin` — валовая прибыль (`total_revenue - total_cost`);
- `margin_percent` — процент маржи (`gross_margin / total_revenue * 100`);
- `avg_sale_price` — средняя цена продажи.

Опишите ETL-процесс поэтапно:

1. **Извлечение (Extract).** Укажите, из каких таблиц извлекаются данные (`sales`, `stores`, `products`, `deliveries` для себестоимости). Как учитываются изменения цен закупки (если они менялись в течение месяца)? Предложите использовать исторические данные из `deliveries` или хранить цену закупки в отдельной таблице исторических цен.
2. **Трансформация (Transform).** Опишите все необходимые преобразования:
 - соединение таблиц для получения полной информации о продажах (магазин, товар, категория, цена продажи);
 - расчёт себестоимости: для каждой продажи нужно определить соответствующую цену закупки на момент продажи (можно использовать среднюю цену закупки за месяц из поставок, либо точную по дате — более сложный вариант);
 - группировка по магазину, категории и месяцу;
 - вычисление агрегатов (суммы, средние, проценты).
3. **Загрузка (Load).** Определите стратегию загрузки: полная перезагрузка витрины за каждый месяц, либо инкрементальное обновление (если данные за месяц уже загружены, их можно пересчитать). Предложите использовать `MERGE` или `INSERT ON CONFLICT` для обновления существующих записей.
4. **Оркестрация.** Как часто будет запускаться процесс (ежемесячно)? Какие инструменты вы бы использовали (например, хранимая процедура в PostgreSQL, запускаемая по расписанию через `pg_cron` или внешний планировщик)?

Приведите псевдокод или SQL-фрагменты основных операций, например, запрос для расчёта себестоимости продаж с использованием оконных функций или подзапросов.

Задание 4. Дашборд

Разработайте макет дашборда для руководителя сети супермаркетов, который позволит отслеживать эффективность магазинов и управлять ассортиментом. Дашборд должен быть интерактивным и содержать следующие элементы:

1. KPI-панель:

- Общая выручка сети за выбранный период.
- Средняя маржинальность по сети.
- Количество проданных товаров (в единицах).
- Количество магазинов.
- Динамика изменения выручки по сравнению с предыдущим периодом (в процентах).

2. **Динамика выручки по магазинам.** Линейный график, показывающий изменение выручки каждого магазина по месяцам за последний год. С возможностью выбора одного или нескольких магазинов для сравнения.

3. **Сравнение магазинов по маржинальности.** Столбчатая диаграмма, где каждый столбец — магазин, высота — маржинальность (в процентах). Цветом можно выделить магазины с маржинальностью выше или ниже среднего.

4. **Распределение продаж по категориям.** Круговая диаграмма (для всей сети) или столбчатая диаграмма с группировкой по магазинам, показывающая долю выручки каждой категории товаров.

5. **Топ-10 товаров-лидеров по выручке.** Таблица или гистограмма с названиями товаров, категориями, суммарной выручкой и количеством проданных единиц за период.

6. **Товары с низкой оборачиваемостью.** Таблица, показывающая товары, которые продаются медленно (например, с количеством продаж меньше среднего по категории), для принятия решений о снятии с продажи или проведении акций.

7. Фильтры:

- по периоду (месяц, квартал, год, произвольный интервал);
- по магазину (мультивыбор);
- по категории товара.

Все визуализации должны быть связаны: выбор магазина пересчитывает все показатели для него.

Визуализации:

- **Линейный график** — для динамики выручки.
- **Столбчатая диаграмма** — для сравнения магазинов по маржинальности, для топ-товаров.
- **Круговая диаграмма** — для структуры продаж по категориям.

- **Таблица** — для списков товаров-лидеров и товаров с низкой оборачиваемостью.
- **Карточки KPI** — для ключевых показателей.

Опишите, из каких источников данных будут строиться визуализации (из созданной витрины или напрямую из базы). Предложите инструмент для реализации дашборда (например, Power BI, Tableau, Yandex DataLens). Приложите эскиз дашборда (можно в виде текстового описания компоновки или рисунка).

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 3. Онлайн-образование (платформа курсов)

Описание кейса

Образовательная платформа предлагает пользователям онлайн-курсы по различным тематикам: программирование, дизайн, маркетинг, иностранные языки и т.д. Пользователи регистрируются на платформе, просматривают каталог курсов, записываются на интересные курсы и проходят уроки в рамках этих курсов. Каждый курс состоит из набора уроков, расположенных в определённой последовательности. Прогресс пользователя отслеживается: фиксируется, какие уроки завершены, а какие ещё нет. Администрация платформы хочет анализировать активность пользователей, популярность курсов, успеваемость и выявлять факторы, влияющие на завершение курсов. Бизнес-цели:

- Мониторинг количества активных пользователей, новых регистраций и динамики записей на курсы.
- Оценка популярности курсов (по количеству записей, по проценту завершения).
- Анализ успеваемости пользователей: средний процент завершения курсов, количество пользователей, успешно завершивших курс.
- Выявление «слабых» мест в курсах — уроков, на которых пользователи чаще всего останавливаются.
- Прогнозирование оттока пользователей на основе их активности.

Для поддержки принятия решений необходимо создать хранилище данных, разработать ETL-процессы и построить дашборд для администратора платформы.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для платформы онлайн-образования, включающую как минимум следующие таблицы:

- **users** — пользователи (идентификатор, имя, email, дата регистрации, статус, страна, дата рождения — опционально);
- **courses** — курсы (идентификатор, название, описание, цена, автор, категория, дата публикации, статус);
- **lessons** — уроки (идентификатор, идентификатор курса, название, порядковый номер в курсе, длительность в минутах, тип (видео, текст, тест), ссылка на контент);
- **enrollments** — записи на курсы (идентификатор, идентификатор пользователя, идентификатор курса, дата записи, статус записи (активна, завершена, отменена), дата завершения);
- **progress** — прогресс по урокам (идентификатор, идентификатор записи, идентификатор урока, флаг завершения, дата завершения, оценка за тест — если есть).

Для каждой таблицы определите:

- первичные ключи (PRIMARY KEY);
- внешние ключи (FOREIGN KEY) с указанием связей (один-ко-многим);

- ограничения целостности (NOT NULL, UNIQUE, CHECK, DEFAULT).

Напишите SQL-скрипт на языке PostgreSQL для создания всех таблиц в правильном порядке (с учётом зависимостей по внешним ключам). Добавьте комментарии к таблицам и столбцам с использованием команды `COMMENT ON`. Предусмотрите ограничения, например:

- email пользователя должен быть уникальным (`UNIQUE`);
- цена курса должна быть неотрицательной (`CHECK (price >= 0)`);
- длительность урока должна быть больше нуля (`CHECK (duration_minutes > 0)`);
- статус записи на курс должен быть из допустимого списка (`CHECK (status IN ('active', 'completed', 'cancelled'))`);
- порядковый номер урока должен быть уникальным в пределах одного курса (можно наложить уникальное ограничение на (`course_id, order_number`)).

Опишите связи между таблицами:

- `courses` и `users` связаны через `enrollments` (многие-ко-многим);
- `courses` и `lessons` — один курс содержит много уроков;
- `enrollments` и `progress` — одна запись содержит много записей прогресса (по каждому уроку);
- `progress` связывает `enrollments` и `lessons`.

Приложите ER-диаграмму (можно в виде текстового описания или рисунка).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций с данными:

1. **Регистрация нового пользователя.** Напишите запрос, который вставляет нового пользователя с указанием имени, email и даты регистрации. Проверьте, что email уникален.
2. **Обновление длительности урока.** Напишите запрос, который изменяет длительность урока по его идентификатору. Убедитесь, что новая длительность больше нуля.
3. **Отмена записи на курс (удаление enrollment).** Напишите запрос, который удаляет запись пользователя на курс. Предусмотрите, что при удалении записи должны каскадно удаляться связанные записи прогресса (или можно просто изменить статус на 'cancelled' без удаления — выберите вариант).
4. **Выборка пользователей, завершивших более 80% курса.** Напишите запрос, который для заданного курса возвращает список пользователей, у которых доля завершённых уроков превышает 80%. Выведите имя пользователя, email, общее количество уроков в курсе, количество завершённых уроков и процент завершения.
5. **Расчёт средней длительности прохождения курса.** Напишите запрос, который вычисляет среднее время (в днях) между датой записи и датой завершения для всех завершённых записей по каждому курсу.

6. **Поиск наиболее популярных курсов.** Напишите запрос, который возвращает топ-5 курсов по количеству активных записей (статус 'active') за последний месяц.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения ежедневной витрины данных «Активность пользователей и популярность курсов», которая обновляется каждый день. Целевая витрина должна содержать следующие показатели за каждый день (или за отчётный день):

- **date** — дата отчёта;
- **new_users** — количество новых зарегистрированных пользователей за день;
- **active_users** — количество уникальных пользователей, которые выполнили хотя бы одно действие за день (завершили урок, записались на курс);
- **total_enrollments** — общее количество записей на курсы за день;
- **completed_lessons** — общее количество завершённых уроков за день;
- **completed_courses** — количество курсов, полностью завершённых пользователями за день (т.е. записей со статусом 'completed').

Также можно добавить агрегацию по курсам: количество записей, количество завершённых, процент завершения.

Опишите ETL-процесс поэтапно:

1. **Извлечение (Extract).** Укажите, из каких таблиц извлекаются данные (**users**, **enrollments**, **progress**, **lessons**, **courses**). Как обрабатываются инкрементальные изменения (только новые записи за день)? Предложите использовать поля дат (регистрация, дата завершения) для фильтрации.
2. **Трансформация (Transform).** Опишите необходимые преобразования:
 - подсчёт количества новых пользователей по дате регистрации;
 - подсчёт активных пользователей (по дате завершения урока или дате записи на курс) — нужно определить, что считается активностью;
 - подсчёт завершённых уроков по дате завершения;
 - расчёт количества завершённых курсов (по статусу **enrollment**);
 - группировка по дате.
3. **Загрузка (Load).** Определите стратегию загрузки: полная перезагрузка за весь период (если витрина небольшая) или инкрементальная (добавление только новых дней). Предложите использовать **INSERT ON CONFLICT** для обновления существующих записей, если они пересчитываются.
4. **Оркестрация.** Как часто будет запускаться процесс (ежедневно)? Какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, запускаемые через **pg_cron**, или внешние скрипты на Python).

Приведите псевдокод или SQL-фрагменты основных операций, например, запрос для расчёта активных пользователей за день.

Задание 4. Дашборд

Разработайте макет дашборда для администратора платформы онлайн-образования, который позволит отслеживать ключевые метрики и управлять контентом. Дашборд должен содержать следующие элементы:

1. KPI-панель:

- Общее количество пользователей.
- Количество активных пользователей за последние 30 дней.
- Количество курсов.
- Средний процент завершения курсов по всем пользователям.
- Количество новых регистраций за сегодня/неделю.

2. **Динамика регистраций и активности.** Линейный график с двумя линиями: количество новых регистраций и количество активных пользователей по дням за последние 30 дней.

3. **Популярность курсов.** Столбчатая диаграмма (топ-10 курсов по количеству записей) с возможностью переключения на топ по проценту завершения.

4. **Проходимость курсов.** Гистограмма распределения пользователей по проценту завершения курса (например, группы: 0–20%, 20–40%, 40–60%, 60–80%, 80–100%) для выбранного курса или в среднем по всем.

5. **Успеваемость пользователей.** Таблица с пользователями, имеющими наибольшее количество завершённых курсов, и пользователями, имеющими самый низкий прогресс (для выявления отстающих).

6. **Список уроков с низкой завершаемостью.** Таблица, показывающая уроки, которые пользователи чаще всего пропускают (низкий процент завершения относительно других уроков в курсе).

7. Фильтры:

- по дате (период);
- по курсу (выбор конкретного курса для детального анализа);
- по категории курса.

Все визуализации должны быть взаимосвязаны: выбор курса обновляет данные о проходимости и успеваемости.

Визуализации:

- **Линейный график** — для динамики регистраций и активности.
- **Столбчатая диаграмма** — для популярности курсов и распределения по проценту завершения.
- **Круговая диаграмма** — для структуры пользователей по статусам (например, активные/неактивные).
- **Таблица** — для списков пользователей и уроков.

- Карточки KPI — для ключевых показателей.

Опишите, из каких источников данных будут строиться визуализации (из созданной витрины активности или напрямую из базы). Предложите инструмент для реализации дашборда (например, Power BI, Tableau, Yandex DataLens). Приложите эскиз дашборда (можно в виде текстового описания компоновки или рисунка).

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 4. Банк (кредитование)

Описание кейса

Банк предоставляет кредиты физическим лицам на различные цели: потребительские кредиты, ипотеку, автокредиты, кредитные карты. Ведётся учёт клиентов, кредитных продуктов (программ кредитования), кредитных договоров и платежей по ним. Каждый кредитный договор содержит сумму, процентную ставку, срок, дату выдачи, график платежей. Клиенты вносят платежи (основной долг, проценты, комиссии, штрафы) согласно графику или досрочно. Банк хочет анализировать:

- Общий кредитный портфель: сумма выданных кредитов, количество активных договоров, средняя ставка.
- Качество кредитного портфеля: доля просроченных кредитов, объём просроченной задолженности, динамика просрочек.
- Эффективность кредитных продуктов: популярность, доходность (процентный доход), уровень дефолтности.
- Кредитную дисциплину клиентов: количество просрочек, средняя длительность просрочки, доля проблемных заёмщиков.
- Прогнозирование дефолтов и формирование резервов.

Для поддержки принятия решений необходимо создать хранилище данных, разработать ETL-процессы и построить дашборд для кредитного отдела и риск-менеджмента.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта кредитов, включающую как минимум следующие таблицы:

- **clients** — клиенты (идентификатор, ФИО, паспортные данные, дата рождения, контактный телефон, email, адрес, дата регистрации в системе);
- **credit_products** — кредитные продукты (идентификатор, название, описание, базовая процентная ставка, минимальная и максимальная сумма, минимальный и максимальный срок, тип кредита);
- **loan_contracts** — кредитные договоры (идентификатор, идентификатор клиента, идентификатор продукта, сумма кредита, процентная ставка (фактическая, может отличаться от базовой), срок в месяцах, дата выдачи, дата окончания, остаток основного долга, статус договора);
- **payments** — платежи (идентификатор, идентификатор договора, дата платежа, сумма платежа, тип платежа (основной долг, проценты, штраф, комиссия), статус платежа);
- **payment_schedule** — график платежей (идентификатор, идентификатор договора, плановая дата платежа, плановая сумма, тип платежа) — опционально, но полезно для расчёта просрочек.

Для каждой таблицы определите:

- первичные ключи (PRIMARY KEY);

- внешние ключи (FOREIGN KEY) с указанием связей;
- ограничения целостности (NOT NULL, UNIQUE, CHECK, DEFAULT).

Напишите SQL-скрипт на языке PostgreSQL для создания всех таблиц в правильном порядке. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например:

- сумма кредита должна быть положительной (CHECK (amount > 0));
- процентная ставка должна быть в разумных пределах (CHECK (interest_rate BETWEEN 0 AND 100));
- срок в месяцах должен быть больше нуля;
- статус договора должен быть из списка ('active', 'closed', 'overdue', 'defaulted').

Опишите связи между таблицами:

- один клиент может иметь много договоров,
- один продукт может быть использован в многих договорах,
- один договор имеет много платежей и записей в графике.

Приложите ER-диаграмму (можно в виде текстового описания или рисунка).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций с данными:

1. **Добавление нового клиента.** Напишите запрос, который вставляет нового клиента с указанием всех необходимых полей. Проверьте уникальность паспортных данных.
2. **Изменение процентной ставки по кредитному продукту.** Напишите запрос, который обновляет базовую процентную ставку для указанного продукта. Проверьте, что новая ставка находится в допустимом диапазоне.
3. **Удаление договора при закрытии.** Напишите запрос, который либо удаляет договор (если нет платежей), либо изменяет его статус на 'closed'. Предусмотрите, что удаление допускается только для договоров без платежей.
4. **Выборка договоров с просрочкой более 30 дней.** Напишите запрос, который возвращает все активные договоры, по которым есть просрочка более 30 дней (т.е. есть неоплаченные платежи по графику с датой, превышающей текущую дату на 30 дней). Выведите ФИО клиента, сумму договора, дату выдачи, сумму просрочки.
5. **Расчёт общей суммы задолженности по клиенту.** Напишите запрос, который для заданного клиента вычисляет общую сумму остатка по всем его активным договорам.
6. **Анализ платежей за период.** Напишите запрос, который группирует платежи по месяцам и типам (основной долг, проценты, штрафы) и возвращает суммы за указанный период.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения ежедневной витрины данных «Просроченная задолженность и качество портфеля», которая обновляется каждый день. Целевая витрина должна содержать следующие показатели по каждому кредитному договору на текущую дату:

- `contract_id`, `client_id`, `client_name`;
- `product_name`, `contract_amount`, `interest_rate`;
- `issue_date`, `planned_end_date`;
- `total_paid_principal` — сумма выплаченного основного долга;
- `total_paid_interest` — сумма выплаченных процентов;
- `remaining_debt` — остаток основного долга;
- `next_payment_date` — дата следующего планового платежа;
- `days_overdue` — количество дней просрочки (если есть);
- `overdue_amount` — сумма просроченных платежей;
- `payment_status` — статус платежа (on time, overdue, defaulted).

Также добавьте агрегацию по портфелю в целом: общая сумма выданных кредитов, общая сумма просрочки, доля просроченных договоров.

Опишите ETL-процесс поэтапно:

1. **Извлечение (Extract).** Укажите, из каких таблиц извлекаются данные (`loan_contracts`, `payments`, `clients`, `credit_products`, `payment_schedule`). Как обрабатываются инкрементальные изменения (новые договоры, платежи, изменение статусов)? Предложите использовать поля дат для фильтрации изменений.
2. **Трансформация (Transform).** Опишите необходимые преобразования:
 - расчёт остатка основного долга на основе суммы договора и всех платежей по основному долгу;
 - определение даты следующего платежа и суммы просрочки (путём сравнения графика платежей с текущей датой);
 - вычисление количества дней просрочки;
 - классификация статуса договора (active, overdue, defaulted, closed);
 - объединение с данными клиентов и продуктов.
3. **Загрузка (Load).** Определите стратегию загрузки: полная перезагрузка всей витрины каждый день или инкрементальное обновление (пересчёт только изменённых договоров). Предложите использовать `MERGE` или `INSERT ON CONFLICT` для обновления.
4. **Оркестрация.** Как часто будет запускаться процесс (ежедневно)? Какие инструменты вы бы использовали?

Приведите псевдокод или SQL-фрагменты основных операций, например, запрос для расчёта просрочки с использованием оконных функций или подзапросов.

Задание 4. Дашборд

Разработайте макет дашборда для кредитного отдела и риск-менеджеров, который позволит отслеживать состояние кредитного портфеля и управлять просрочками. Дашборд должен содержать следующие элементы:

1. KPI-панель:

- Общая сумма выданных кредитов (активные договоры).
- Количество активных договоров.
- Доля просроченных договоров (в % от общего числа активных).
- Общая сумма просроченной задолженности.
- Средневзвешенная процентная ставка по портфелю.
- Динамика просрочки (изменение за день/неделю).

2. **Динамика просрочек.** Линейный график, показывающий изменение общей суммы просрочки и доли просроченных договоров по дням/месяцам за последние 6–12 месяцев.

3. **Распределение портфеля по кредитным продуктам.** Столбчатая диаграмма, показывающая общую сумму выданных кредитов и долю просрочки по каждому продукту.

4. **Возрастная структура просрочки.** Столбчатая диаграмма, показывающая распределение просроченной задолженности по длительности просрочки (например, 1–30 дней, 31–60 дней, 61–90 дней, >90 дней).

5. **Список должников (топ-10).** Таблица с клиентами, имеющими наибольшую сумму просроченной задолженности, с указанием ФИО, суммы договора, суммы просрочки и количества дней просрочки.

6. **Карта/географическое распределение** (если есть регионы). Тепловая карта, показывающая концентрацию просрочек по регионам.

7. Фильтры:

- по дате (период);
- по кредитному продукту;
- по типу клиента (физическое лицо, ИП, юрлицо — если есть);
- по статусу договора.

Все визуализации должны быть взаимосвязаны.

Визуализации:

- **Линейный график** — для динамики просрочек.
- **Столбчатая диаграмма** — для распределения по продуктам и по длительности просрочки.
- **Круговая диаграмма** — для структуры портфеля по статусам.
- **Таблица** — для списка должников.

- Тепловая карта — для географического распределения.
- Карточки KPI — для ключевых показателей.

Опишите, из каких источников данных будут строиться визуализации (из созданной витрины просрочки). Предложите инструмент для реализации дашборда. Приложите эскиз дашборда.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 5. Транспортная компания (грузоперевозки)

Описание кейса

Транспортная компания осуществляет грузовые перевозки по территории России. Компания имеет парк транспортных средств (грузовики, фуры), штат водителей и базу клиентов, заказывающих перевозки. Каждый заказ содержит информацию о маршруте (город отправления и город назначения), весе груза, дате подачи и сроке доставки. Компания стремится оптимизировать загрузку автомобилей, сократить время доставки и повысить эффективность работы водителей. В качестве бизнес-целей выделяются:

- Анализ загрузки парка транспортных средств (процент использования, простой).
- Оценка эффективности водителей (среднее время в рейсе, соблюдение сроков, количество выполненных заказов).
- Мониторинг выполнения заказов (статусы, задержки).
- Выявление наиболее загруженных маршрутов и регионов.

Для решения этих задач необходимо создать аналитическое хранилище данных, построить ETL-процессы и разработать дашборд для руководства компании.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта перевозок. Модель должна включать как минимум следующие сущности:

- **customers** — клиенты (заказчики перевозок);
- **vehicles** — транспортные средства;
- **drivers** — водители;
- **orders** — заказы на перевозку;
- **shipments** — фактически выполненные рейсы (отправки).

Для каждой сущности определите необходимый набор атрибутов, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами с указанием кратности.

Опишите назначение каждой таблицы и предполагаемую бизнес-логику (например, как формируется shipment, как связан с заказом и т.п.). На основе ER-диаграммы напишите SQL-скрипт на языке PostgreSQL для создания всех таблиц в правильном порядке (с учётом внешних ключей). В скрипте должны присутствовать:

- команды **CREATE TABLE** с полным перечнем столбцов;
- ограничения **PRIMARY KEY**, **FOREIGN KEY**, **CHECK** (например, грузоподъёмность > 0 , статус в списке допустимых);
- комментарии к таблицам и столбцам (команда **COMMENT** в PostgreSQL).

Примерная структура таблиц (вы можете расширить или уточнить атрибуты по своему усмотрению):

1. **customers:** `customer_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `contact_person`, `phone`, `email`, `registration_date` (DATE DEFAULT CURRENT_DATE).
2. **vehicles:** `vehicle_id` (SERIAL PRIMARY KEY), `plate_number` (VARCHAR(20) UNIQUE NOT NULL), `capacity_kg` (NUMERIC(10,2) CHECK > 0), `model`, `year`, `status` (VARCHAR(20) CHECK status IN ('active', 'maintenance', 'retired')).
3. **drivers:** `driver_id` (SERIAL PRIMARY KEY), `full_name` (NOT NULL), `license_number` (UNIQUE), `hire_date`, `phone`, `status` ('active', 'inactive').
4. **orders:** `order_id` (SERIAL PRIMARY KEY), `customer_id` (INTEGER REFERENCES customers), `origin_city` (VARCHAR(100) NOT NULL), `dest_city` (VARCHAR(100) NOT NULL), `weight_kg` (NUMERIC(10,2) CHECK > 0), `order_date` (DATE DEFAULT CURRENT_DATE), `required_delivery_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('new', 'assigned', 'in_transit', 'delivered', 'cancelled')).
5. **shipments:** `shipment_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders), `vehicle_id` (INTEGER REFERENCES vehicles), `driver_id` (INTEGER REFERENCES drivers), `departure_date` (TIMESTAMP), `arrival_date` (TIMESTAMP), `actual_weight_kg` (NUMERIC(10,2)), `delivery_status` (VARCHAR(20) CHECK status IN ('planned', 'departed', 'arrived', 'delayed')).

Помимо этих таблиц, возможно, потребуется справочник городов или регионов, но по условию задачи достаточно указанных.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций с данными:

1. **Добавление нового заказа.** Вставьте запись в таблицу `orders` с указанием клиента, маршрута, веса и даты. Предусмотрите проверку существования клиента.
2. **Обновление статуса заказа.** Напишите запрос, который изменяет статус заказа на 'delivered' при условии, что соответствующий рейс завершён (задайте условие через подзапрос к `shipments`).
3. **Удаление транспортного средства.** Удалите запись из таблицы `vehicles`, если у данного ТС нет связанных рейсов (проверка через NOT EXISTS).
4. **Извлечение отчёта за период.** Выведите все заказы за определённый месяц (с заданными датами) с указанием клиента, маршрута, статуса и ответственного водителя (если назначен).
5. **Расчёт средней загрузки.** Напишите запрос, который для каждого автомобиля за последний месяц вычисляет средний вес перевезённого груза (на основе `actual_weight_kg` в `shipments`) и количество выполненных рейсов.
6. (дополнительно) **Поиск задержек.** Выявите заказы, у которых дата фактической доставки превышает требуемую дату доставки более чем на 1 день.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции (BEGIN/COMMIT) для операций изменения.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Эффективность водителей», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому водителю за день (или за неделю):

- `driver_id, driver_name`;
- `date` (отчётная дата);
- `total_orders` — количество выполненных заказов за период;
- `total_distance_km` — суммарный пробег (можно вычислить как расстояние между городами, для упрощения используйте справочник расстояний, либо примите, что расстояние указано в заказе — добавьте поле);
- `avg_load_kg` — средний вес перевезённого груза;
- `avg_trip_time_hours` — среднее время рейса (разница между `arrival` и `departure`);
- `delivery_on_time_percent` — процент доставок без задержек.

Опишите ETL-процесс поэтапно:

1. **Extract (Извлечение)**. Укажите, из каких таблиц-источников берутся данные (`orders`, `shipments`, `drivers`, возможно, справочник расстояний). Какие данные загружаются (полная загрузка или инкрементальная — за прошедший день).
2. **Transform (Трансформация)**. Опишите все необходимые преобразования:
 - очистка данных (проверка на `NULL`, удаление дублей);
 - расчёт пробега (если нет прямого поля, можно использовать внешний справочник или приближённую формулу);
 - вычисление времени в пути и проверка на задержку (сравнение `arrival_date` с `required_delivery_date`);
 - агрегация по водителям и датам (группировка);
 - объединение с таблицей водителей для получения имён.
3. **Load (Загрузка)**. Определите, как данные будут загружаться в витрину:
 - полная перезагрузка (`TRUNCATE + INSERT`) или инкрементальная (`UPSERT / MERGE`) по ключу (`driver_id, date`). Предложите стратегию обновления.
 - обеспечение качества данных (валидация, логирование ошибок).

Также опишите, какие инструменты вы бы использовали для реализации ETL (например, с использованием PostgreSQL хранимых процедур и триггеров, или внешнего инструмента, например, Apache Airflow, Talend, Python + pandas). Приведите псевдокод или SQL-фрагменты основных операций.

Задание 4. Дашборд

Разработайте макет дашборда для руководства транспортной компании, который позволит отслеживать ключевые показатели деятельности и принимать оперативные решения. Дашборд должен включать следующие элементы:

1. **Ключевые индикаторы (KPI):** общее количество заказов, выполненных за сегодня/неделю/месяц; среднее время доставки; процент задержек; загрузка автопарка (количество активных ТС / общее количество).
2. **Динамика выполнения заказов** — линейный график, показывающий количество заказов по дням, с разбивкой по статусам (new, assigned, in transit, delivered).
3. **Карта маршрутов** — географическая карта с нанесёнными маршрутами (откуда-куда) и интенсивностью перевозок (толщина линии или цвет в зависимости от объёма).
4. **Рейтинг водителей** — таблица или гистограмма, показывающая топ-10 водителей по количеству выполненных заказов, по проценту доставок вовремя, по среднему времени рейса.
5. **Загрузка транспортных средств** — столбчатая диаграмма, показывающая использование каждого ТС (количество рейсов, загрузка в тоннах).
6. **Фильтры и срезы:** по датам (период), по регионам (города отправления/назначения), по типу груза (если есть), по водителям.

Для каждого элемента опишите, какой тип визуализации используется и из какого источника данных (из витрины или напрямую из базы) он будет строиться. Предложите также возможные взаимодействия (например, клик по водителю открывает детальный отчёт по его рейсам).

Оформите описание дашборда в виде текста с перечислением всех панелей и их назначения, а также приложите эскиз (можно в виде блок-схемы или описания расположения элементов на экране).

По результатам выполнения всех четырёх заданий студент должен предоставить:

- отчёт в электронном виде (Word или LaTeX) с ER-диаграммой (рисунок), SQL-скриптами, описанием ETL и макетом дашборда;
- файлы SQL-скриптов для создания базы данных и выполнения CRUD-операций.

Работа оценивается по критериям: полнота и корректность проектирования, работоспособность SQL-кода, обоснованность решений в ETL, продуманность визуализаций, качество оформления и защиты.

Вариант 6. Производственное предприятие (учёт деталей)

Описание кейса

Производственное предприятие выпускает готовые изделия (продукцию), которые собираются из различных деталей (комплектующих). Ведётся учёт деталей, их складских остатков, поставок от поставщиков, а также заказов на производство (сборку) готовых изделий. Бизнес-цели:

- Оптимизация складских запасов (минимизация дефицита и перепроизводства).
- Планирование производственных заказов на основе спроса и остатков.
- Анализ себестоимости продукции (стоимость деталей, затраты на сборку).
- Оценка эффективности поставщиков (сроки, цена, качество).

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для производственного отдела и руководства.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта деталей, складских операций и производственных заказов. Модель должна включать следующие сущности:

- **parts** — справочник деталей (комплектующих);
- **warehouses** — склады (если несколько);
- **stock** — остатки деталей на складах;
- **suppliers** — поставщики деталей;
- **procurement** — поставки (закупки деталей от поставщиков);
- **products** — готовые изделия (конечная продукция);
- **bill_of_materials** — спецификация (перечень деталей, необходимых для сборки одного изделия);
- **production_orders** — заказы на производство (сборку) готовых изделий.

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами (один-ко-многим, многие-ко-многим) и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **parts**: `part_id` (SERIAL PRIMARY KEY), `part_number` (VARCHAR(50) UNIQUE NOT NULL), `description`, `unit` (шт, кг, м), `price` (NUMERIC(10,2)), `supplier_id` (INTEGER REFERENCES suppliers) — можно указать основного поставщика.
2. **suppliers**: `supplier_id` SERIAL PRIMARY KEY, `name` NOT NULL, `contact_person`, `phone`, `email`, `address`.
3. **warehouses**: `warehouse_id` SERIAL PRIMARY KEY, `name`, `location`.

4. `stock`: `stock_id` SERIAL PRIMARY KEY, `part_id` REFERENCES `parts`, `warehouse_id` REFERENCES `warehouses`, `quantity` NUMERIC(10,2) DEFAULT 0 CHECK (`quantity` >= 0), `last_update` TIMESTAMP DEFAULT CURRENT_TIMESTAMP. Уникальность по паре (`part_id`, `warehouse_id`).
5. `procurement`: `procurement_id` SERIAL PRIMARY KEY, `part_id` REFERENCES `parts`, `supplier_id` REFERENCES `suppliers`, `quantity` CHECK >0, `price` NUMERIC(10,2), `delivery_date` DATE, `status` (VARCHAR(20) CHECK статус в ('ordered', 'received', 'cancelled')).
6. `products`: `product_id` SERIAL PRIMARY KEY, `name` NOT NULL, `description`, `selling_price` NUMERIC(10,2).
7. `bill_of_materials`: `bom_id` SERIAL PRIMARY KEY, `product_id` REFERENCES `products`, `part_id` REFERENCES `parts`, `quantity_per_unit` NUMERIC(10,2) CHECK >0. Уникальность по паре (`product_id`, `part_id`).
8. `production_orders`: `order_id` SERIAL PRIMARY KEY, `product_id` REFERENCES `products`, `planned_quantity` INTEGER CHECK >0, `start_date` DATE, `completion_date` DATE, `status` (VARCHAR(20) CHECK статус в ('planned', 'in_progress', 'completed', 'cancelled')), `warehouse_id` REFERENCES `warehouses` (куда оприходовать готовую продукцию).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц с учётом порядка (сначала справочники, потом таблицы с внешними ключами). Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что количество деталей не может быть отрицательным.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения операций с данными в соответствии с бизнес-логикой:

1. **Добавление новой детали** с указанием поставщика и цены.
2. **Изменение цены детали** (обновление поля `price`).
3. **Удаление поставщика** при условии, что на него нет ссылок в таблицах `procurement` и `parts` (проверка через подзапрос).
4. **Выборка деталей с остатком ниже порогового значения** (например, меньше 10) с указанием склада.
5. **Расчёт себестоимости готового изделия** на основе спецификации (сумма `quantity_per_unit` * `price` по всем деталям). Напишите запрос, который для заданного `product_id` возвращает общую стоимость материалов.
6. **Создание производственного заказа** (вставка в `production_orders`) с проверкой наличия достаточного количества деталей на складе (если необходимо, можно без проверки, только для примера).
7. **Обновление статуса производственного заказа** на 'completed' и списание деталей со склада (списание можно сделать в рамках одной транзакции с обновлением `stock`). Для упрощения опишите запросы, но можно ограничиться изменением статуса и отдельным запросом для списания.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Производственная эффективность», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели за день (или за месяц) по каждому производственному заказу и по дням:

- `order_id`, `product_name`, `planned_quantity`, `completed_quantity` (если заказ завершён), `start_date`, `completion_date`;
- `total_material_cost` — общая стоимость деталей, списанных на заказ (из спецификации);
- `production_duration_days` — длительность выполнения (разница между `completion_date` и `start_date`);
- `status`.

Также в витрину можно добавить агрегацию по дням: количество запущенных и завершённых заказов, общая стоимость выпущенной продукции, дефицит деталей (если заказ не может быть выполнен из-за нехватки).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`production_orders`, `products`, `bill_of_materials`, `parts`, `procurement`, `stock`). Укажите, какие данные загружаются (инкрементально — только изменения за последний день).
2. **Transform:** опишите преобразования:
 - объединение заказов с продуктами и спецификацией;
 - расчёт общей стоимости материалов (сумма `quantity_per_unit` * цена детали на момент заказа, либо текущая цена);
 - вычисление длительности;
 - агрегация по дням (если нужно) и группировка.
3. **Load:** как данные загружаются в витрину (полная перезагрузка или инкрементальная, например, через MERGE). Предложите механизм контроля качества (проверка расчётов, отсутствие отрицательных значений).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите инструменты, которые вы бы использовали (например, хранимые процедуры в PostgreSQL, Python с `pandas`, или Apache Airflow).

Задание 4. Дашборд

Разработайте макет дашборда для начальника производственного отдела, который позволит контролировать производственные процессы и управлять запасами. Дашборд должен включать:

1. **KPI:** количество активных заказов, общее количество выпущенных изделий за период, процент выполнения плана, общая стоимость запасов на складах, количество деталей с дефицитом (остаток ниже порога).

2. **Динамика производства** — линейный график, показывающий количество завершённых заказов по дням/неделям.
3. **Структура затрат** — круговая или столбчатая диаграмма, отражающая долю стоимости материалов по видам продукции.
4. **Остатки деталей** — таблица или тепловая карта, показывающая текущие остатки по складам, с выделением критических (красным).
5. **Список заказов в работе** — таблица с деталями: номер заказа, продукт, дата начала, планируемая дата завершения, статус, ответственный (если есть поле).
6. **Фильтры**: по складу, по продукту, по периоду, по статусу заказа.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из операционной БД). Предложите интерактивные взаимодействия, например, клик по продукту открывает детальную спецификацию.

Оформите описание в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент должен предоставить отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с кодом для проверки.

Вариант 7. Телеком-оператор (учёт звонков)

Описание кейса

Телекоммуникационная компания предоставляет услуги сотовой связи. В её базе данных хранится информация об абонентах, тарифных планах, совершённых звонках, а также платежах за услуги. Бизнес-цели:

- Анализ доходности тарифных планов.
- Мониторинг качества обслуживания (длительность звонков, частота обрывов).
- Прогнозирование оттока абонентов на основе их активности.
- Оценка эффективности маркетинговых кампаний (переходы на новые тарифы).

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства компании.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта абонентов, тарифов, звонков и платежей. Модель должна включать как минимум следующие сущности:

- **subscribers** — абоненты (физические лица, пользующиеся услугами);
- **tariffs** — тарифные планы (название, стоимость минуты, абонентская плата и т.п.);
- **calls** — записи о звонках (абонент, дата, длительность, стоимость, направление);
- **payments** — платежи абонентов (сумма, дата, способ оплаты);
- **subscriber_tariffs** — история подключений абонентов к тарифам (с датами начала и окончания действия, текущий активный тариф).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **subscribers**: `subscriber_id` (SERIAL PRIMARY KEY), `phone_number` (VARCHAR(20) UNIQUE NOT NULL), `full_name`, `registration_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('active', 'blocked', 'terminated')), `balance` (NUMERIC(10,2) DEFAULT 0).
2. **tariffs**: `tariff_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(50) NOT NULL), `description`, `monthly_fee` (NUMERIC(10,2) DEFAULT 0), `call_cost_per_minute` (NUMERIC(10,4) NOT NULL), `free_minutes` (INTEGER DEFAULT 0), `valid_from` (DATE), `valid_to` (DATE).
3. **subscriber_tariffs**: `id` (SERIAL PRIMARY KEY), `subscriber_id` REFERENCES `subscribers`, `tariff_id` REFERENCES `tariffs`, `start_date` (DATE NOT NULL), `end_date` (DATE), `is_active` (BOOLEAN DEFAULT TRUE). Уникальность по (`subscriber_id`, `start_date`) с учётом активного.

4. **calls:** `call_id` (SERIAL PRIMARY KEY), `subscriber_id` REFERENCES subscribers, `call_date` (TIMESTAMP NOT NULL), `duration_seconds` (INTEGER CHECK >=0), `call_type` (VARCHAR(20) CHECK call_type IN ('incoming', 'outgoing')), `destination_number` (VARCHAR(20)), `cost` (NUMERIC(10,4) DEFAULT 0).
5. **payments:** `payment_id` (SERIAL PRIMARY KEY), `subscriber_id` REFERENCES subscribers, `payment_date` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `amount` (NUMERIC(10,2) NOT NULL CHECK >0), `payment_method` (VARCHAR(30) CHECK payment_method IN ('cash', 'card', 'online')).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что длительность звонка не может быть отрицательной.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Регистрация нового абонента** с присвоением тарифа по умолчанию (например, базовый тариф).
2. **Изменение тарифа для абонента** — обновление записи в `subscriber_tariffs` (заккрытие старого тарифа и открытие нового с текущей даты).
3. **Удаление абонента** (если его баланс нулевой и нет активных звонков).
4. **Вставка нового звонка** с автоматическим расчётом стоимости на основе тарифа абонента на момент звонка (можно использовать триггер или подзапрос).
5. **Выборка топ-10 абонентов по количеству звонков за последний месяц**.
6. **Расчёт выручки по тарифам за период** — группировка с суммой стоимости звонков и абонентской платы.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Доходность по тарифам», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому тарифу за день (или за месяц):

- `tariff_id`, `tariff_name`;
- `date` (отчётная дата);
- `total_subscribers` — количество абонентов, использующих данный тариф;
- `total_calls` — общее количество звонков;
- `total_duration_minutes` — общая длительность в минутах;
- `revenue_from_calls` — выручка от звонков (сумма `cost`);

- `revenue_from_subscription` — выручка от абонентской платы (расчётная, например, `monthly_fee / кол-во дней месяца * кол-во дней действия`);
- `total_revenue` — общая выручка (сумма звонков + абонентская плата);
- `avg_call_duration` — средняя длительность звонка.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`calls`, `subscriber_tariffs`, `tariffs`, `subscribers`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение звонков с абонентами и тарифами (через `subscriber_tariffs` с учётом даты звонка);
 - расчёт абонентской платы (пропорционально дням активности тарифа в месяце);
 - группировка по тарифам и датам;
 - вычисление средних значений.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `tariff_id, date`). Предложите механизм контроля качества (проверка сумм, сравнение с операционными данными).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с библиотекой `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для финансового директора компании, который позволит отслеживать динамику доходов и активности абонентов. Дашборд должен включать следующие элементы:

1. **KPI:** общая выручка за период, количество активных абонентов, средняя выручка на абонента (ARPU), процент активных абонентов от общего числа.
2. **Динамика выручки** — линейный график с разбивкой по источникам (звонки, абонентская плата) за последние 12 месяцев.
3. **Распределение абонентов по тарифам** — круговая или столбчатая диаграмма, показывающая долю абонентов на каждом тарифе.
4. **Топ-10 тарифов по выручке** — гистограмма.
5. **Динамика длительности звонков** — график средней длительности по дням, с возможностью фильтрации по тарифам.
6. **Таблица активности абонентов** — список абонентов с наибольшим количеством звонков, с указанием тарифа и суммы платежей.
7. **Фильтры:** по периоду, по тарифам, по типу звонков (входящие/исходящие).

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по тарифу открывает детальный отчёт по его абонентам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 8. Больница (учёт пациентов и посещений)

Описание кейса

Больница ведёт электронную запись пациентов, врачей и приёмов (посещений). Каждое посещение фиксируется с указанием пациента, врача, даты и времени, длительности приёма, диагноза (при необходимости) и отделения, к которому прикреплён врач. Бизнес-цели:

- Анализ загруженности врачей и отделений (количество приёмов, среднее время приёма).
- Оценка ожиданий пациентов (среднее время до записи).
- Выявление наиболее востребованных специалистов.
- Мониторинг эффективности работы отделений.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для главного врача и руководства больницы.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта пациентов, врачей, приёмов и отделений. Модель должна включать как минимум следующие сущности:

- **patients** — пациенты (личные данные, контакты);
- **doctors** — врачи (ФИО, специализация, отделение);
- **departments** — отделения больницы (название, корпус);
- **appointments** — приёмы (пациент, врач, дата и время, длительность, диагноз, статус).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **departments**: `department_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `building` (VARCHAR(50)), `phone` (VARCHAR(20)).
2. **doctors**: `doctor_id` (SERIAL PRIMARY KEY), `full_name` (VARCHAR(100) NOT NULL), `specialization` (VARCHAR(100)), `department_id` (INTEGER REFERENCES departments), `phone`, `email` (UNIQUE), `hire_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('active', 'on_leave', 'retired')).
3. **patients**: `patient_id` (SERIAL PRIMARY KEY), `full_name` (VARCHAR(100) NOT NULL), `date_of_birth` (DATE), `gender` (CHAR(1) CHECK gender IN ('M', 'F')), `phone`, `email`, `registration_date` (DATE DEFAULT CURRENT_DATE), `address` (TEXT).
4. **appointments**: `appointment_id` (SERIAL PRIMARY KEY), `patient_id` (INTEGER REFERENCES patients), `doctor_id` (INTEGER REFERENCES doctors), `appointment_datetime` (TIMESTAMP NOT NULL), `duration_minutes` (INTEGER CHECK >0), `diagnosis` (TEXT), `status` (VARCHAR(20) CHECK status IN ('scheduled', 'completed', 'cancelled', 'no_show')), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что длительность приёма должна быть положительной, а статус — из списка допустимых.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового пациента** (вставка всех необходимых полей).
2. **Добавление нового врача** с указанием отделения.
3. **Запись пациента на приём** (вставка в appointments с проверкой существования пациента и врача, а также занятости времени — можно без проверки, только для примера).
4. **Обновление статуса приёма** на 'completed' и добавление диагноза.
5. **Удаление врача** при условии, что у него нет запланированных приёмов.
6. **Выборка всех приёмов за сегодня** с указанием пациента, врача, отделения и статуса.
7. **Расчёт средней загруженности врача** — количество приёмов в день за последнюю неделю.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Загруженность врачей и отделений», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому врачу и отделению за день (или за неделю):

- `doctor_id`, `doctor_name`, `department_id`, `department_name`;
- `date` (отчётная дата);
- `total_appointments` — общее количество приёмов (не считая cancelled и no_show, только completed и scheduled);
- `completed_appointments` — количество завершённых приёмов;
- `avg_duration_minutes` — средняя длительность приёма (только для завершённых);
- `total_patients_served` — уникальное количество пациентов, принятых врачом;
- `no_show_count` — количество неявок;
- `utilization_rate` — коэффициент загрузки (можно рассчитать как отношение общей длительности приёмов к рабочему времени, если задано; для упрощения — процент занятости от максимально возможного числа приёмов).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (appointments, doctors, departments, patients). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - фильтрация по статусу (исключение cancelled, возможно);
 - соединение с врачами и отделениями;
 - агрегация (группировка по врачу и дате);
 - вычисление средней длительности;
 - подсчёт уникальных пациентов через COUNT(DISTINCT patient_id).
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу doctor_id, date). Предложите механизм контроля качества (проверка на отрицательные значения, логирование ошибок).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для главного врача, который позволит отслеживать загруженность врачей и отделений, а также качество обслуживания. Дашборд должен включать следующие элементы:

1. **KPI:** общее количество приёмов за день/неделю, средняя длительность приёма, количество неявок (no-show), общее количество пациентов, прошедших приём.
2. **Загруженность врачей** — столбчатая диаграмма (количество приёмов по каждому врачу) с возможностью фильтрации по отделению.
3. **Динамика загруженности** — линейный график количества приёмов по дням за последний месяц, с разбивкой по отделениям.
4. **Распределение приёмов по статусам** — круговая диаграмма (scheduled, completed, cancelled, no_show).
5. **Топ-5 диагнозов** — гистограмма или таблица с частотой диагнозов.
6. **Таблица загруженности отделений** — список отделений с количеством приёмов, средним временем ожидания (можно рассчитать как разницу между созданием записи и временем приёма, если есть).
7. **Фильтры:** по дате, по отделению, по врачу.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по врачу открывает детальный отчёт по его приёмам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 9. Гостиница (бронирование номеров)

Описание кейса

Гостиница предоставляет услуги размещения. Ведётся учёт номеров, гостей, бронирований и оплат. Каждый номер имеет категорию (стандарт, люкс и т.п.), цену за ночь и статус (свободен, занят, на ремонте). Бронирования создаются гостями на конкретные даты, могут быть отменены или изменены. Бизнес-цели:

- Анализ заполняемости номеров (occupancy rate) в разрезе времени, категорий номеров.
- Оценка средней стоимости ночи и выручки.
- Прогнозирование спроса для управления ценами.
- Выявление наиболее популярных категорий номеров и сезонных трендов.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства гостиницы.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта гостей, номеров, бронирований и оплат. Модель должна включать как минимум следующие сущности:

- **guests** — гости (личные данные, контакты);
- **rooms** — номера (категория, этаж, цена, статус);
- **bookings** — бронирования (гость, номер, даты заезда и выезда, статус);
- **payments** — оплаты (сумма, дата, способ оплаты, связь с бронированием).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **rooms**: `room_id` (SERIAL PRIMARY KEY), `room_number` (VARCHAR(10) UNIQUE NOT NULL), `category` (VARCHAR(50) CHECK category IN ('standard', 'comfort', 'suite', 'lux'), `floor` (INTEGER), `price_per_night` (NUMERIC(10,2) NOT NULL CHECK >0), `status` (VARCHAR(20) CHECK status IN ('available', 'occupied', 'maintenance'))).
2. **guests**: `guest_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `passport_number` (VARCHAR(20) UNIQUE), `registration_date` (DATE DEFAULT CURRENT_DATE).
3. **bookings**: `booking_id` (SERIAL PRIMARY KEY), `guest_id` (INTEGER REFERENCES guests), `room_id` (INTEGER REFERENCES rooms), `check_in_date` (DATE NOT NULL), `check_out_date` (DATE NOT NULL, CHECK check_out_date > check_in_date), `status` (VARCHAR(20) CHECK status IN ('confirmed', 'checked_in', 'checked_out', 'cancelled')), `total_price` (NUMERIC(10,2) DEFAULT 0), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).

4. `payments: payment_id (SERIAL PRIMARY KEY), booking_id (INTEGER REFERENCES bookings), payment_date (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), amount (NUMERIC(10,2) NOT NULL CHECK >0), payment_method (VARCHAR(30) CHECK payment_method IN ('cash', 'card', 'online')), status (VARCHAR(20) CHECK status IN ('pending', 'completed', 'refunded'))`.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что дата выезда должна быть позже даты заезда, а цена не может быть отрицательной.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового номера** (вставка всех полей).
2. **Регистрация нового гостя**.
3. **Создание бронирования** (проверка доступности номера на указанные даты, если не проверять, то хотя бы вставка с проверкой существования гостя и номера).
4. **Обновление статуса бронирования** на `'checked_in'` (заезд) и `'checked_out'` (выезд) с обновлением статуса номера.
5. **Отмена бронирования** (установка статуса `'cancelled'`).
6. **Выборка всех активных бронирований на сегодня** с указанием гостя и номера.
7. **Расчёт выручки за месяц** — сумма всех `completed` платежей за период.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Заполняемость гостиницы» (occupancy), которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому дню (или по неделе) в разрезе категорий номеров:

- `date` (отчётная дата);
- `room_category` (категория номера);
- `total_rooms` — общее количество номеров данной категории;
- `occupied_rooms` — количество номеров, занятых в эту дату (когда `check_in <= date < check_out`);
- `occupancy_rate` — заполняемость в процентах (`occupied_rooms / total_rooms * 100`);
- `total_revenue` — выручка от проживания в эту дату (можно расчётную, как сумма стоимости ночей для всех занятых номеров);
- `average_price_per_night` — средняя цена за ночь по категории.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (bookings, rooms, payments). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - генерация всех дат в анализируемом периоде (можно использовать `generate_series`);
 - для каждой даты и каждой категории определить, сколько номеров занято (через пересечение интервалов `check_in` - `check_out` с датой, учитывая статус `'confirmed'` или `'checked_in'`);
 - расчёт выручки: либо по фактическим платежам, либо пропорционально длительности проживания;
 - группировка по дате и категории.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная — например, добавлять только новые дни). Предложите механизм контроля качества (проверка, что `occupancy_rate <= 100%`).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для менеджера гостиницы, который позволит отслеживать ключевые показатели деятельности. Дашборд должен включать следующие элементы:

1. **KPI:** текущая заполняемость (сегодня), средняя заполняемость за месяц, общая выручка за месяц, средняя цена ночи, количество свободных номеров.
2. **Динамика заполняемости** — линейный график за последние 30 дней с разбивкой по категориям номеров.
3. **Заполняемость по категориям** — столбчатая диаграмма (сравнение категорий за текущий месяц).
4. **Календарь занятости** — тепловая карта (дни x категории) с цветовой индикацией заполняемости.
5. **Выручка по источникам бронирования** (если есть поле `source`) — круговая диаграмма, но у нас нет, можно заменить на распределение по способу оплаты.
6. **Список предстоящих заездов** — таблица с датами, гостями, номерами, статусом.
7. **Фильтры:** по датам, по категории номера, по статусу бронирования.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по дате на графике показывает детали по бронированиям.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 10. Кинотеатр (продажа билетов)

Описание кейса

Сеть кинотеатров ведёт учёт фильмов, залов, сеансов и проданных билетов. Каждый фильм имеет название, жанр, возрастное ограничение, длительность. Залы характеризуются вместимостью и типом (стандартный, IMAX, VIP). Сеанс привязан к фильму, залу и времени начала. Продажа билетов фиксирует место, цену, способ оплаты и статус (продан, возвращён). Бизнес-цели:

- Анализ кассовых сборов по фильмам, жанрам и периодам.
- Оценка популярности фильмов (количество проданных билетов, заполняемость залов).
- Выявление наиболее прибыльных сеансов (время, зал, день недели).
- Прогнозирование спроса для планирования расписания.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства кинотеатра.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта фильмов, залов, сеансов и билетов. Модель должна включать как минимум следующие сущности:

- **films** — фильмы (название, жанр, длительность, возрастное ограничение);
- **halls** — залы (название, вместимость, тип);
- **sessions** — сеансы (фильм, зал, дата и время начала, цена билета);
- **tickets** — билеты (сеанс, ряд, место, статус, цена продажи, способ оплаты);
- **customers** (опционально) — покупатели, если нужна аналитика по клиентам, но можно обойтись без неё.

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **films**: `film_id` (SERIAL PRIMARY KEY), `title` (VARCHAR(200) NOT NULL), `genre` (VARCHAR(50)), `duration_minutes` (INTEGER CHECK >0), `age_restriction` (VARCHAR(10) CHECK age_restriction IN ('0+', '6+', '12+', '16+', '18+')), `release_date` (DATE).
2. **halls**: `hall_id` (SERIAL PRIMARY KEY), `hall_name` (VARCHAR(50) NOT NULL), `capacity` (INTEGER NOT NULL CHECK >0), `hall_type` (VARCHAR(30) CHECK hall_type IN ('standard', 'imax', 'vip')).
3. **sessions**: `session_id` (SERIAL PRIMARY KEY), `film_id` (INTEGER REFERENCES films), `hall_id` (INTEGER REFERENCES halls), `start_datetime` (TIMESTAMP NOT NULL), `base_price` (NUMERIC(10,2) NOT NULL CHECK >0), `status` (VARCHAR(20) CHECK status IN ('scheduled', 'ongoing', 'finished', 'cancelled')).

4. `tickets`: `ticket_id` (SERIAL PRIMARY KEY), `session_id` (INTEGER REFERENCES sessions), `row_number` (INTEGER NOT NULL), `seat_number` (INTEGER NOT NULL), `price` (NUMERIC(10,2) NOT NULL), `status` (VARCHAR(20) CHECK status IN ('available','sold'), `sale_datetime` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `payment_method` (VARCHAR(30) CHECK payment_method IN ('cash','card','online')), `customer_email` (VARCHAR(100) — опционально).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что цена билета не может быть отрицательной, а статус — из допустимого списка. Также можно добавить уникальное ограничение на (`session_id`, `row_number`, `seat_number`) для предотвращения двойной продажи.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового фильма.**
2. **Создание сеанса** (привязка к фильму и залу с проверкой пересечения по времени в одном зале — для упрощения можно опустить проверку).
3. **Продажа билета** (вставка записи в `tickets` со статусом 'sold' и обновление доступных мест — если нужно, но можно просто вставлять).
4. **Возврат билета** (обновление статуса на 'refunded').
5. **Выборка всех сеансов на сегодня** с указанием фильма, зала, времени и количества свободных мест (вычисляем через подзапрос).
6. **Отчёт по кассовым сборам за день** — сумма продаж по всем билетам со статусом 'sold' за текущую дату.
7. **Топ-5 фильмов по продажам за месяц** — группировка по фильму с суммой выручки.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Кассовые сборы и популярность фильмов», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому фильму за день (или за месяц):

- `film_id`, `film_title`, `genre`;
- `date` (отчётная дата);
- `total_tickets_sold` — количество проданных билетов;
- `total_revenue` — общая выручка (сумма цен проданных билетов);
- `avg_ticket_price` — средняя цена билета;

- **occupancy_rate** — заполняемость залов (отношение числа проданных билетов к общей вместимости сеансов данного фильма в этот день).

Также можно добавить показатели по залам или времени сеанса, но для базового варианта достаточно.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (tickets, sessions, films, halls). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение билетов с сеансами, фильмами и залами;
 - фильтрация только проданных билетов (status='sold');
 - агрегация (группировка по фильму и дате);
 - вычисление средней цены и заполняемости (для заполняемости нужно знать общую вместимость всех сеансов фильма в этот день, что можно получить, суммируя capacity по сеансам).
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу film_id, date). Предложите механизм контроля качества (проверка, что total_revenue >= 0 и occupancy_rate <= 100%).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя кинотеатра, который позволит отслеживать динамику продаж и популярность фильмов. Дашборд должен включать следующие элементы:

1. **KPI:** общая выручка за сегодня/неделю/месяц, общее количество проданных билетов, средняя цена билета, заполняемость залов (средняя за период).
2. **Динамика выручки** — линейный график по дням за последние 30 дней, с разбивкой по жанрам или фильмам (по выбору).
3. **Топ-5 фильмов по выручке** — столбчатая диаграмма за выбранный период.
4. **Распределение продаж по жанрам** — круговая диаграмма.
5. **Загрузка сеансов** — тепловая карта (дни недели × время сеанса) с цветовой индикацией заполняемости.
6. **Таблица сеансов с низкой заполняемостью** — список сеансов, где occupancy_rate < 30%, для принятия решений (отмена или перенос).
7. **Фильтры:** по дате, по фильму, по жанру, по залу.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по фильму открывает детальный отчёт по его сеансам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 11. Авиакомпания (рейсы, бронирования)

Описание кейса

Авиакомпания выполняет регулярные и чартерные рейсы. Ведётся учёт рейсов, самолётов, аэропортов, пассажиров и бронирований. Каждый рейс имеет маршрут (аэропорт вылета и прилёта), дату и время вылета/прилёта, статус (запланирован, в пути, прибыл, отменён, задержан). Пассажиры бронируют места на рейсы, оплачивают билеты, проходят регистрацию. Бизнес-цели:

- Анализ задержек рейсов: причины, частота, влияние на пассажиров.
- Оценка загрузки самолётов (количество пассажиров, занятость мест).
- Мониторинг эффективности маршрутов (прибыльность, заполняемость).
- Управление расписанием и планирование флота.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства авиакомпании.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта рейсов, пассажиров, бронирований и самолётов. Модель должна включать как минимум следующие сущности:

- **airports** — аэропорты (код, название, город, страна);
- **aircraft** — самолёты (бортовой номер, модель, вместимость, год выпуска);
- **flights** — рейсы (номер, аэропорт вылета, аэропорт прилёта, дата и время вылета/прилёта, статус, самолёт);
- **passengers** — пассажиры (ФИО, контакты, паспортные данные);
- **bookings** — бронирования (пассажир, рейс, дата бронирования, класс обслуживания, стоимость, статус);
- **check_ins** — регистрация на рейс (пассажир, рейс, время регистрации, номер места).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **airports**: `airport_code` (CHAR(3) PRIMARY KEY — код IATA), `name` (VARCHAR(100) NOT NULL), `city` (VARCHAR(100)), `country` (VARCHAR(100)), `timezone` (VARCHAR(50)).
2. **aircraft**: `aircraft_id` (SERIAL PRIMARY KEY), `tail_number` (VARCHAR(20) UNIQUE NOT NULL), `model` (VARCHAR(100) NOT NULL), `capacity` (INTEGER NOT NULL CHECK >0), `manufacture_year` (INTEGER CHECK >1900), `status` (VARCHAR(20) CHECK status IN ('active', 'maintenance', 'retired')).

3. **flights:** `flight_id` (SERIAL PRIMARY KEY), `flight_number` (VARCHAR(10) NOT NULL), `aircraft_id` (INTEGER REFERENCES aircraft), `origin_code` (CHAR(3) REFERENCES airports), `dest_code` (CHAR(3) REFERENCES airports), `departure_datetime` (TIMESTAMP NOT NULL), `arrival_datetime` (TIMESTAMP NOT NULL, CHECK `arrival_datetime > departure_datetime`), `status` (VARCHAR(20) CHECK `status IN ('scheduled', 'delayed', 'boarding', 'in_air', 'arrived', 'cancelled')`), `delay_minutes` (INTEGER DEFAULT 0).
4. **passengers:** `passenger_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `date_of_birth` (DATE), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `passport_number` (VARCHAR(20) UNIQUE).
5. **bookings:** `booking_id` (SERIAL PRIMARY KEY), `passenger_id` (INTEGER REFERENCES passengers), `flight_id` (INTEGER REFERENCES flights), `booking_date` (DATE DEFAULT CURRENT_DATE), `seat_class` (VARCHAR(20) CHECK `seat_class IN ('economy', 'business', 'first')`), `price` (NUMERIC(10,2) NOT NULL CHECK `>0`), `status` (VARCHAR(20) CHECK `status IN ('confirmed', 'cancelled', 'checked_in')`).
6. **check_ins:** `check_in_id` (SERIAL PRIMARY KEY), `booking_id` (INTEGER REFERENCES bookings UNIQUE — одно бронирование может быть зарегистрировано один раз), `check_in_time` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `seat_number` (VARCHAR(10) NOT NULL). Также можно добавить отдельно `boarding_pass` для деталей, но для упрощения достаточно.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что стоимость билета не может быть отрицательной, а время прилёта должно быть позже времени вылета, а также что статус рейса — из допустимого списка.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового рейса** с указанием всех обязательных полей.
2. **Бронирование билета** (вставка в bookings с проверкой наличия пассажира и рейса).
3. **Регистрация пассажира** (запись в check_ins с обновлением статуса бронирования на 'checked_in').
4. **Отмена бронирования** (обновление статуса на 'cancelled').
5. **Обновление статуса рейса** (например, на 'arrived' по прибытии).
6. **Выборка всех рейсов с задержками** за текущую неделю (задержка > 0).
7. **Расчёт средней загрузки самолёта** за месяц — отношение числа зарегистрированных пассажиров к вместимости самолёта для каждого рейса.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Эффективность рейсов и задержки», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому рейсу (или по дням) в разрезе маршрутов, самолётов:

- `flight_id`, `flight_number`, `origin`, `dest`, `aircraft_model`;
- `date` (дата вылета);
- `scheduled_departure`, `actual_departure`, `delay_minutes`;
- `total_passengers` — количество пассажиров с подтверждённым бронированием;
- `checked_in_passengers` — количество прошедших регистрацию;
- `load_factor` — коэффициент загрузки ($\text{checked_in} / \text{capacity} * 100$);
- `on_time_flag` — признак выполнения вовремя (задержка ≤ 15 минут).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`flights`, `bookings`, `check_ins`, `passengers`, `aircraft`, `airports`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение таблиц для получения полной информации о рейсе;
 - подсчёт количества пассажиров (по статусу `'confirmed'`), зарегистрированных;
 - расчёт загрузки (через вместимость самолёта);
 - вычисление задержки и признака `punctuality`;
 - агрегация (группировка по рейсам или по дням).
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `flight_id`). Предложите механизм контроля качества (проверка, что `load_factor` не превышает 100%).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для диспетчера авиакомпании, который позволит отслеживать задержки и эффективность рейсов. Дашборд должен включать следующие элементы:

1. **KPI:** процент рейсов, выполненных вовремя (за последние сутки/неделю), средняя задержка, общее количество рейсов, количество задержанных.
2. **Динамика задержек** — линейный график (средняя задержка по дням за последние 30 дней).
3. **Причины задержек** — столбчатая диаграмма (если есть поле причины, но у нас его нет, можно просто показать распределение по маршрутам).

4. **Загрузка самолётов** — столбчатая диаграмма по каждому самолёту (средняя загрузка за период).
5. **Карта маршрутов** — географическая карта с линиями маршрутов, где толщина линии зависит от интенсивности рейсов.
6. **Список задержанных рейсов на сегодня** — таблица с рейсами, временем задержки, статусом.
7. **Фильтры:** по дате, по самолёту, по маршруту, по статусу рейса.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по рейсу открывает детальный отчёт о пассажирах.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 12. HR-система (сотрудники, вакансии)

Описание кейса

HR-отдел компании ведёт учёт сотрудников, открытых вакансий и заявок от соискателей. Каждая вакансия имеет название, отдел, требования, уровень зарплаты, статус (открыта, закрыта, приостановлена). Кандидаты подают заявки на вакансии, проходят собеседования, и в результате некоторые заявки переходят в статус «нанят» (кандидат становится сотрудником). Бизнес-цели:

- Анализ эффективности процесса найма (время закрытия вакансий, стоимость найма, качество кандидатов).
- Мониторинг открытых вакансий и планирование найма.
- Оценка отделов по скорости и качеству подбора персонала.
- Выявление наиболее востребованных навыков и проблемных позиций.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для HR-директора и менеджеров по персоналу.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта сотрудников, вакансий, кандидатов и заявок. Модель должна включать как минимум следующие сущности:

- **departments** — отделы (название, руководитель);
- **employees** — сотрудники (ФИО, отдел, должность, дата приёма, контакты);
- **vacancies** — вакансии (название, отдел, требования, зарплата, статус);
- **candidates** — кандидаты (ФИО, контакты, резюме, навыки);
- **applications** — заявки (кандидат, вакансия, дата подачи, статус, результаты собеседований).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **departments**: `department_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `manager_id` (INTEGER — ссылка на сотрудника, позже).
2. **employees**: `employee_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `department_id` (INTEGER REFERENCES departments), `position` (VARCHAR(100)), `hire_date` (DATE NOT NULL), `phone`, `email` (UNIQUE), `status` (VARCHAR(20) CHECK status IN ('active', 'inactive', 'on_leave')).
3. **vacancies**: `vacancy_id` (SERIAL PRIMARY KEY), `title` (VARCHAR(100) NOT NULL), `department_id` (INTEGER REFERENCES departments), `requirements` (TEXT), `salary_range_min` (NUMERIC(10,2)), `salary_range_max` (NUMERIC(10,2)), `posted_date` (DATE DEFAULT CURRENT_DATE), `closing_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('open', 'closed', 'on_hold')).

4. **candidates:** `candidate_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone`, `email` (UNIQUE), `resume_text` (TEXT), `skills` (TEXT), `registered_date` (DATE DEFAULT CURRENT_DATE).
5. **applications:** `application_id` (SERIAL PRIMARY KEY), `candidate_id` (INTEGER REFERENCES candidates), `vacancy_id` (INTEGER REFERENCES vacancies), `application_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(30) CHECK status IN ('submitted', 'screening', 'interview', 'offer', 'hired', 'rejected')), `interview_feedback` (TEXT), `hire_date` (DATE) — дата, когда кандидат стал сотрудником (если статус hired).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что зарплатные диапазоны положительные, статусы из допустимых списков, а также уникальность email для сотрудников и кандидатов.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового сотрудника.**
2. **Создание новой вакансии** с указанием отдела и требований.
3. **Регистрация нового кандидата.**
4. **Подача заявки на вакансию** (вставка в applications).
5. **Обновление статуса заявки** (например, с 'submitted' на 'interview', затем на 'hired').
6. **Закрытие вакансии** (установка статуса 'closed' и даты закрытия).
7. **Выборка всех открытых вакансий** с указанием отдела и количества поданных заявок (подзапрос).
8. **Расчёт среднего времени закрытия вакансий** — разница между `closing_date` и `posted_date` для закрытых вакансий.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Эффективность найма», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому отделу и по вакансиям за месяц (или за неделю):

- `vacancy_id`, `vacancy_title`, `department`;
- `posted_date`, `closing_date`, `days_to_fill` — время закрытия в днях;
- `total_applications` — общее количество заявок;
- `applications_to_interview` — количество заявок, дошедших до собеседования;
- `applications_to_hire` — количество заявок, завершившихся наймом;

- `hire_rate` — процент заявок, перешедших в найм;
- `avg_salary_offered` — средняя зарплата по наймам (если доступна);
- `cost_per_hire` — стоимость найма (если есть данные о затратах, можно добавить).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`vacancies`, `applications`, `candidates`, `departments`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение заявок с вакансиями и отделами;
 - расчёт времени закрытия (разница между `closing_date` и `posted_date`);
 - подсчёт количества заявок по статусам (например, группировка с `CASE`);
 - вычисление показателей `hire_rate` и т.д.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `vacancy_id`). Предложите механизм контроля качества (проверка, что `hire_rate` в пределах 0-100%).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для HR-директора, который позволит отслеживать процесс найма и эффективность отделов. Дашборд должен включать следующие элементы:

1. **KPI:** количество открытых вакансий, среднее время закрытия вакансий, общее количество нанятых за период, процент успешных заявок (`hire rate`).
2. **Динамика закрытия вакансий** — линейный график с количеством закрытых вакансий по дням/неделям за последние 3 месяца.
3. **Среднее время закрытия по отделам** — столбчатая диаграмма.
4. **Воронка найма** — визуализация этапов (`submitted` → `screening` → `interview` → `offer` → `hired`) в виде воронки.
5. **Топ-5 вакансий с наибольшим количеством заявок** — гистограмма.
6. **Список вакансий, открытых более 30 дней** — таблица для принятия мер.
7. **Фильтры:** по отделу, по дате, по статусу вакансии.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по отделу открывает детальный отчёт по его вакансиям.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 13. Склад (учёт товаров)

Описание кейса

Складское хозяйство компании ведёт учёт товаров, их хранения и перемещений. Каждый товар имеет наименование, артикул, категорию, цену, единицу измерения. Склады имеют название, местоположение, вместимость. Движения товаров фиксируют приход (поступление от поставщиков), расход (отгрузка клиентам или списание), перемещение между складами, инвентаризацию (корректировка остатков). Бизнес-цели:

- Мониторинг текущих остатков товаров в разрезе складов и категорий.
- Анализ оборачиваемости товаров (скорость продаж, период хранения).
- Выявление товаров с низкой оборачиваемостью (залежавшихся) и дефицитных позиций.
- Планирование закупок на основе прогноза спроса.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководителя складского отдела и отдела закупок.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта товаров, складов и движений. Модель должна включать как минимум следующие сущности:

- **warehouses** — склады (название, адрес, вместимость);
- **products** — товары (наименование, артикул, категория, цена, единица измерения);
- **suppliers** — поставщики (название, контакты) — опционально, для учёта поставок;
- **stock_movements** — движения товаров (тип: приход, расход, перемещение, инвентаризация; количество, дата, склад-отправитель/получатель, связь с поставщиком/клиентом);
- **current_stock** — текущие остатки на складах (можно вычислять на основе движений, но для производительности можно хранить отдельную таблицу).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **warehouses**: `warehouse_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `location` (VARCHAR(200)), `capacity_volume` (NUMERIC(10,2)) — опционально.
2. **products**: `product_id` (SERIAL PRIMARY KEY), `sku` (VARCHAR(50) UNIQUE NOT NULL), `name` (VARCHAR(200) NOT NULL), `category` (VARCHAR(100)), `unit` (VARCHAR(20) CHECK unit IN ('шт', 'кг', 'л', 'м')), `price` (NUMERIC(10,2) NOT NULL CHECK >0).
3. **suppliers**: `supplier_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `contact_person`, `phone`, `email`.
4. **customers** — можно добавить для учёта отгрузок клиентам, но по условию не обязательно.

5. **stock_movements**: **movement_id** (SERIAL PRIMARY KEY), **product_id** (INTEGER REFERENCES products), **from_warehouse_id** (INTEGER REFERENCES warehouses), **to_warehouse_id** (INTEGER REFERENCES warehouses), **movement_type** (VARCHAR(20) CHECK movement_type IN ('incoming', 'outgoing', 'transfer', 'adjustment')), **quantity** (NUMERIC(10,2) NOT NULL CHECK >0), **movement_date** (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), **reference_id** (INTEGER) — ссылка на заказ, поставку и т.п., **remarks** (TEXT).
6. **current_stock**: **stock_id** (SERIAL PRIMARY KEY), **product_id** (INTEGER REFERENCES products), **warehouse_id** (INTEGER REFERENCES warehouses), **quantity** (NUMERIC(10,2) DEFAULT 0 CHECK >=0). Уникальность по папе (product_id, warehouse_id).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что количество в движении должно быть положительным, тип движения из допустимого списка, а также что для перемещения оба склада должны быть указаны (или один из них — для прихода/расхода). При создании current_stock предусмотрите триггер для автоматического обновления при вставке движений (для целостности, но это не обязательно в рамках задания).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового товара.**
2. **Регистрация нового склада.**
3. **Внесение движения (приход):** добавление записи в stock_movements с типом 'incoming', обновление current_stock (увеличение остатка на складе-получателе). Для простоты можно выполнять UPDATE отдельным запросом.
4. **Внесение движения (расход):** тип 'outgoing', уменьшение остатка на складе-отправителе.
5. **Внесение перемещения между складами:** тип 'transfer', уменьшение на складе-отправителе и увеличение на складе-получателе.
6. **Корректировка остатков (инвентаризация):** тип 'adjustment', установка нового остатка (можно через установку абсолютного значения).
7. **Выборка текущих остатков** с фильтром по складу и товару.
8. **Отчёт по оборачиваемости:** выборка всех движений за период с группировкой по товарам, вычисление общего прихода и расхода.

Все запросы должны быть корректными и учитывать ссылочную целостность. Для операций изменения остатков рекомендуется использовать транзакции (BEGIN/COMMIT).

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Оборачиваемость товаров», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому товару за месяц (или за неделю):

- `product_id`, `product_name`, `category`;
- `period_start`, `period_end` (даты);
- `opening_stock` — остаток на начало периода (можно взять из `current_stock` на дату начала);
- `total_incoming` — суммарный приход за период;
- `total_outgoing` — суммарный расход за период;
- `closing_stock` — остаток на конец периода ($\text{opening} + \text{incoming} - \text{outgoing}$);
- `turnover_rate` — оборачиваемость ($\text{total_outgoing} / \text{average_stock}$, где $\text{average_stock} = (\text{opening} + \text{closing}) / 2$);
- `days_of_stock` — период хранения в днях ($365 / \text{turnover_rate}$, если $\text{turnover_rate} > 0$).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`stock_movements`, `products`, `warehouses`, `current_stock`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - определение начального остатка (может потребоваться вычисление на основе истории движений до начала периода, либо использование снимка `current_stock`);
 - агрегация прихода и расхода по товарам (суммирование `quantity` по типам);
 - расчёт оборачиваемости по формуле;
 - фильтрация товаров без движений (можно исключить).
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка за каждый месяц, либо инкрементальное обновление за последние дни). Предложите механизм контроля качества (проверка, что `closing_stock` ≥ 0 , `turnover_rate` неотрицательная).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя склада, который позволит отслеживать остатки и оборачиваемость товаров. Дашборд должен включать следующие элементы:

1. **КРІ:** общее количество товаров на складах, суммарная стоимость запасов ($\text{quantity} * \text{price}$), оборачиваемость в целом за период, количество товаров с нулевым остатком (дефицит).
2. **Остатки по категориям** — столбчатая диаграмма (общее количество или стоимость по категориям).
3. **Динамика остатков** — линейный график по дням для выбранного товара или категории.
4. **Товары с низкой оборачиваемостью** — таблица с товарами, у которых `days_of_stock` $>$ порогового значения (например, > 90 дней).
5. **Товары с высоким спросом** — топ-10 товаров по оборотам (`total_outgoing`) за период.
6. **Тепловая карта складов** — визуализация загрузки складов (по количеству товаров или по объёму).
7. **Фильтры:** по складу, по категории товара, по периоду.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по товару открывает детальную историю движений.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 14. Логистический центр (заказы, доставка)

Описание кейса

Логистический центр занимается обработкой и доставкой заказов клиентов. Заказы поступают от клиентов, упаковываются на складе, затем передаются водителям для доставки по адресам. Каждый заказ имеет статус (новый, упакован, передан в доставку, доставлен, отменён), информацию о клиенте, список товаров, адрес доставки, дату и время создания, плановую дату доставки. Водители выполняют маршруты, получая задания на доставку нескольких заказов. Бизнес-цели:

- Отслеживание выполнения заказов в реальном времени (от создания до доставки).
- Оценка эффективности доставки (среднее время доставки, процент выполненных вовремя).
- Мониторинг загрузки водителей и маршрутов.
- Выявление задержек и проблемных зон (по адресам, по водителям).
- Анализ популярности товаров и сегментов клиентов.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководителя логистического центра.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта заказов, клиентов, товаров, водителей и статусов доставки. Модель должна включать как минимум следующие сущности:

- **customers** — клиенты (ФИО, контакты, адрес по умолчанию);
- **products** — товары (название, категория, цена, вес);
- **orders** — заказы (клиент, дата, адрес доставки, статус, плановая дата доставки);
- **order_items** — позиции заказа (заказ, товар, количество, цена);
- **drivers** — водители (ФИО, контакты, статус);
- **deliveries** — доставки (заказ, водитель, дата отправки, дата доставки, статус, комментарий);
- **delivery_status_history** — история статусов заказа для отслеживания изменений.

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **customers**: **customer_id** (SERIAL PRIMARY KEY), **first_name** (VARCHAR(50) NOT NULL), **last_name** (VARCHAR(50) NOT NULL), **phone** (VARCHAR(20) NOT NULL), **email** (VARCHAR(100) UNIQUE), **default_address** (TEXT), **registration_date** (DATE DEFAULT CURRENT_DATE).

2. **products:** `product_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `category` (VARCHAR(50)), `price` (NUMERIC(10,2) NOT NULL CHECK >0), `weight_kg` (NUMERIC(10,2) DEFAULT 0), `stock_quantity` (INTEGER DEFAULT 0).
3. **orders:** `order_id` (SERIAL PRIMARY KEY), `customer_id` (INTEGER REFERENCES customers), `order_date` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `delivery_address` (TEXT NOT NULL), `planned_delivery_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('new','packed','dispatched','delivered','cancelled')), `total_amount` (NUMERIC(10,2)).
4. **order_items:** `item_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders ON DELETE CASCADE), `product_id` (INTEGER REFERENCES products), `quantity` (INTEGER NOT NULL CHECK >0), `price_at_time` (NUMERIC(10,2) NOT NULL).
5. **drivers:** `driver_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20) NOT NULL), `vehicle_type` (VARCHAR(30)), `status` (VARCHAR(20) CHECK status IN ('available','on_route','off_duty')).
6. **deliveries:** `delivery_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders UNIQUE), `driver_id` (INTEGER REFERENCES drivers), `dispatch_datetime` (TIMESTAMP), `delivery_datetime` (TIMESTAMP), `status` (VARCHAR(20) CHECK status IN ('assigned','in_transit','delivered','failed')), `delivery_notes` (TEXT).
7. **delivery_status_history:** `history_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders), `status` (VARCHAR(20)), `change_datetime` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `changed_by` (VARCHAR(50)).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что цена не может быть отрицательной, статус из допустимых списков, а также уникальность delivery по заказу (один заказ — одна доставка).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового клиента.**
2. **Создание нового заказа** с добавлением позиций и расчётом общей суммы (можно сделать через триггер, но в рамках задания — вычислять вручную).
3. **Обновление статуса заказа** (например, с 'new' на 'packed', затем на 'dispatched').
4. **Назначение водителя на доставку** (вставка в deliveries с указанием driver_id и статуса 'assigned').
5. **Завершение доставки:** обновление статуса deliveries на 'delivered', установка delivery_datetime, обновление статуса заказа на 'delivered'.
6. **Отмена заказа:** установка статуса 'cancelled' и удаление связанной доставки (если есть).

7. **Выборка всех заказов со статусом 'dispatched'** с указанием водителя и плановой даты доставки.
8. **Расчёт среднего времени доставки** за последнюю неделю — разница между `delivery_datetime` и `dispatch_datetime` для завершённых доставок.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Эффективность доставки», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому дню, водителю или маршруту:

- `date` (отчётная дата);
- `driver_id, driver_name`;
- `total_orders_assigned` — количество заказов, назначенных водителю;
- `total_orders_delivered` — количество доставленных заказов;
- `on_time_deliveries` — количество доставок, выполненных не позднее плановой даты (`planned_delivery_date`);
- `avg_delivery_time_hours` — среднее время доставки (от `dispatch` до `delivery`);
- `delivery_success_rate` — процент успешных доставок (`delivered / assigned`);
- `total_revenue` — выручка от доставленных заказов за день.

Также можно добавить агрегацию по регионам или по категориям товаров.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`orders`, `deliveries`, `drivers`, `customers`, `order_items`, `products`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение заказов, доставок и водителей;
 - вычисление времени доставки и признака вовремя;
 - агрегация по водителям и датам;
 - расчёт средней длительности и процентов.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `driver_id, date`). Предложите механизм контроля качества (проверка, что `delivery_success_rate` $\leq 100\%$ и `avg_delivery_time` ≥ 0).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя логистического центра, который позволит отслеживать статусы заказов и эффективность доставки. Дашборд должен включать следующие элементы:

1. **KPI:** общее количество заказов (за сегодня/неделю), процент доставленных вовремя, среднее время доставки, количество заказов в обработке, количество активных водителей.
2. **Статусы заказов** — круговая диаграмма (new, packed, dispatched, delivered, cancelled).
3. **Динамика заказов** — линейный график (количество заказов по дням за последние 30 дней) с наложением статусов.
4. **Эффективность водителей** — столбчатая диаграмма (количество доставок, процент успешных, среднее время) по каждому водителю.
5. **Карта маршрутов** — географическая карта с отметками точек доставки (можно условно).
6. **Список заказов с просроченной доставкой** — таблица с заказами, где `planned_delivery_date` < текущей даты, а статус не 'delivered'.
7. **Фильтры:** по дате, по водителю, по статусу заказа, по региону (если есть).

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по водителю открывает детальный отчёт по его доставкам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 15. Энергоснабжающая компания (показания счетчиков)

Описание кейса

Энергоснабжающая компания предоставляет услуги по поставке электроэнергии, газа или воды. Она обслуживает множество абонентов (физических и юридических лиц), у каждого из которых установлены приборы учёта (счётчики). Периодически (обычно раз в месяц) снимаются показания счётчиков, на основе которых вычисляется объём потреблённого ресурса, выставляются счета и производится оплата. Кроме того, компания имеет информацию о тарифах, которые могут различаться по регионам, типам потребителей (население, бюджетные организации, промышленность) и времени суток (дифференцированные тарифы). Бизнес-цели:

- Мониторинг общего потребления ресурсов в региональном разрезе, по категориям потребителей.
- Анализ динамики потребления (сезонность, рост/падение), выявление аномалий.
- Контроль задолженности по оплате, прогнозирование поступлений.
- Оценка эффективности работы счётчиков и точности показаний.
- Планирование закупок ресурсов и инвестиций в инфраструктуру.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства компании.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта абонентов, счётчиков, показаний и оплат. Модель должна включать как минимум следующие сущности:

- **regions** — регионы (области, города, районы) для географической аналитики;
- **customers** — абоненты (ФИО, адрес, категория (население, юрлицо, бюджет), контакты);
- **meters** — счётчики (номер, тип ресурса, установленная мощность, дата установки, статус);
- **tariffs** — тарифы (тип ресурса, категория потребителя, тарифная зона, цена за единицу, период действия);
- **readings** — показания счётчиков (дата снятия, показание, тип (текущее, поверочное), способ передачи);
- **consumption** — расчётное потребление за период (объём, сумма к оплате, статус оплаты);
- **payments** — оплаты (сумма, дата, способ, статус).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **regions:** `region_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `parent_region_id` (INTEGER REFERENCES regions) — для иерархии (страна, область, район).
2. **customers:** `customer_id` (SERIAL PRIMARY KEY), `full_name` (VARCHAR(200) NOT NULL), `address` (TEXT NOT NULL), `region_id` (INTEGER REFERENCES regions), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `customer_type` (VARCHAR(20) CHECK `customer_type` IN ('residential', 'commercial', 'budget', 'industrial')), `registration_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK `status` IN ('active', 'suspended', 'terminated')).
3. **meters:** `meter_id` (SERIAL PRIMARY KEY), `meter_number` (VARCHAR(50) UNIQUE NOT NULL), `customer_id` (INTEGER REFERENCES customers), `resource_type` (VARCHAR(20) CHECK `resource_type` IN ('electricity', 'gas', 'water', 'heat')), `installed_date` (DATE), `last_reading_date` (DATE), `initial_reading` (NUMERIC(12,3) DEFAULT 0), `status` (VARCHAR(20) CHECK `status` IN ('active', 'broken', 'decommissioned')).
4. **tariffs:** `tariff_id` (SERIAL PRIMARY KEY), `resource_type` (VARCHAR(20)), `customer_type` (VARCHAR(20)), `region_id` (INTEGER REFERENCES regions), `valid_from` (DATE), `valid_to` (DATE), `price_per_unit` (NUMERIC(10,4) NOT NULL CHECK > 0).
5. **readings:** `reading_id` (SERIAL PRIMARY KEY), `meter_id` (INTEGER REFERENCES meters), `reading_date` (DATE NOT NULL), `reading_value` (NUMERIC(12,3) NOT NULL), `reading_type` (VARCHAR(20) CHECK `reading_type` IN ('regular', 'check', 'adjustment')), `submission_method` (VARCHAR(20) CHECK `submission_method` IN ('manual', 'online', 'estimated')), `upload_timestamp` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).
6. **consumption_periods:** `period_id` (SERIAL PRIMARY KEY), `start_date` (DATE), `end_date` (DATE), `status` (VARCHAR(20) CHECK `status` IN ('draft', 'calculated', 'billed', 'paid')), `customer_id` (INTEGER REFERENCES customers), `total_consumption` (NUMERIC(12,3)), `total_amount` (NUMERIC(12,2)).
7. **payments:** `payment_id` (SERIAL PRIMARY KEY), `period_id` (INTEGER REFERENCES consumption_periods), `payment_date` (DATE), `amount` (NUMERIC(12,2) NOT NULL CHECK > 0), `payment_method` (VARCHAR(30) CHECK `payment_method` IN ('cash', 'card', 'bank_transfer', 'online')), `status` (VARCHAR(20) CHECK `status` IN ('pending', 'completed', 'failed')).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что показания не могут быть отрицательными, дата показания не может быть в будущем, тариф должен действовать в диапазоне дат.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Регистрация нового абонента** (вставка в customers).
2. **Добавление нового счётчика** для существующего абонента.
3. **Внесение показаний счётчика** (вставка в readings) с проверкой, что новое показание больше предыдущего (можно через триггер или в приложении).

4. **Расчёт потребления за период** (например, за месяц) — вычисление разницы между последним и предыдущим показаниями для каждого счётчика и сохранение в `consumption_periods`.
5. **Обновление статуса оплаты** (например, с `'pending'` на `'completed'`).
6. **Выборка всех абонентов с просроченной задолженностью** (периоды со статусом `'billed'`, но без оплаты или с частичной оплатой).
7. **Отчёт по потреблению за месяц по регионам** — группировка с суммированием `total_consumption` по регионам.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Потребление и задолженность по регионам», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому региону (и по категориям потребителей) за месяц:

- `region_id`, `region_name`;
- `report_month` (дата первого дня месяца);
- `customer_type` (опционально, если нужна детализация);
- `total_customers` — количество активных абонентов;
- `total_consumption` — общий объём потреблённого ресурса;
- `total_revenue` — общая сумма начислений по тарифам;
- `total_paid` — сумма оплаченных счетов за этот месяц;
- `debt_amount` — сумма задолженности (начисления минус оплаты);
- `debt_ratio` — отношение долга к начислениям.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`customers`, `meters`, `readings`, `consumption_periods`, `payments`, `tariffs`, `regions`). Укажите, загружаются ли данные за весь период или инкрементально (например, за последний месяц).
2. **Transform:** опишите преобразования:
 - расчёт потребления для каждого абонента за месяц (если нет готовых `consumption_periods`, то по разности показаний);
 - расчёт суммы к оплате по тарифам, действовавшим в этом периоде (необходимо определить тариф для каждого абонента на месяц);
 - объединение с платежами для вычисления задолженности;
 - агрегация по регионам и категориям.

3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `region_id`, `report_month`). Предложите механизм контроля качества (проверка, что `debt_amount` ≥ 0 и `debt_ratio` ≤ 1).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя энергоснабжающей компании, который позволит отслеживать потребление и задолженность. Дашборд должен включать следующие элементы:

1. **KPI:** общее потребление за месяц, общая выручка, сумма задолженности, процент оплаты (`paid / revenue`), количество абонентов с долгом.
2. **Динамика потребления** — линейный график за последние 12 месяцев по регионам (с возможностью выбора).
3. **Структура потребления по категориям** — столбчатая диаграмма (население, промышленность, бюджет).
4. **Карта регионов** — тепловая карта с оттенками, показывающими уровень потребления или задолженности.
5. **Топ-10 должников** — таблица с абонентами, имеющими наибольшую задолженность.
6. **Распределение оплат по способам** — круговая диаграмма.
7. **Фильтры:** по региону, по типу потребителя, по периоду.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по региону на карте открывает детальный отчёт по этому региону.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 16. Недвижимость (продажа/аренда)

Описание кейса

Агентство недвижимости специализируется на продаже и аренде жилой и коммерческой недвижимости. Ведётся учёт объектов недвижимости (квартиры, дома, офисы, склады), клиентов (покупатели, арендаторы, собственники), а также сделок (продажа, аренда). Каждый объект имеет характеристики: тип недвижимости, площадь, количество комнат, этаж, адрес, цена продажи или аренды, статус (активно, продано, сдано, в аренде). Клиенты подразделяются на продавцов/арендодателей и покупателей/арендаторов. Сделки фиксируют дату, тип (продажа, аренда), стороны, сумму, комиссию агентства, статус (завершена, в процессе). Бизнес-цели:

- Анализ рынка недвижимости: динамика цен по типам объектов, районам, периодам.
- Оценка эффективности работы агентства: количество сделок, выручка от комиссий, конверсия.
- Мониторинг активности клиентов и объектов (сроки продажи, аренды).
- Прогнозирование спроса и сезонных колебаний.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства агентства.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта объектов, клиентов и сделок. Модель должна включать как минимум следующие сущности:

- **property_types** — справочник типов недвижимости (квартира, дом, офис и т.п.);
- **properties** — объекты недвижимости (характеристики, адрес, цена, статус);
- **clients** — клиенты (ФИО, контакты, тип (покупатель, продавец, арендатор, арендодатель));
- **transactions** — сделки (тип, дата, стороны, сумма, комиссия, статус);
- **property_media** — медиафайлы объектов (фото, документы) — опционально.

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **property_types**: `type_id` (SERIAL PRIMARY KEY), `type_name` (VARCHAR(50) NOT NULL UNIQUE), `description` (TEXT).
2. **clients**: `client_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20) NOT NULL), `email` (VARCHAR(100) UNIQUE), `client_type` (VARCHAR(30) CHECK client_type IN ('buyer', 'seller', 'renter', 'landlord')), `registration_date` (DATE DEFAULT CURRENT_DATE)

3. **properties:** `property_id` (SERIAL PRIMARY KEY), `address` (TEXT NOT NULL), `city` (VARCHAR(100)), `district` (VARCHAR(100)), `type_id` (INTEGER REFERENCES `property_types`), `area_sqm` (NUMERIC(10,2) CHECK >0), `rooms_count` (INTEGER DEFAULT 0), `floor` (INTEGER), `total_floors` (INTEGER), `price` (NUMERIC(12,2) CHECK >0), `status` (VARCHAR(20) CHECK status IN ('active', 'sold', 'rented', 'reserved')), `listed_date` (DATE DEFAULT CURRENT_DATE), `description` (TEXT), `owner_client_id` (INTEGER REFERENCES `clients`).
4. **transactions:** `transaction_id` (SERIAL PRIMARY KEY), `property_id` (INTEGER REFERENCES `properties`), `seller_id` (INTEGER REFERENCES `clients`), `buyer_id` (INTEGER REFERENCES `clients`), `transaction_type` (VARCHAR(20) CHECK `transaction_type` IN ('sale', 'rent')), `transaction_date` (DATE NOT NULL), `amount` (NUMERIC(12,2) NOT NULL CHECK >0), `commission_percent` (NUMERIC(5,2) DEFAULT 5.0), `commission_amount` (NUMERIC(12,2) GENERATED ALWAYS AS (`amount` * `commission_percent` / 100) STORED), `status` (VARCHAR(20) CHECK status IN ('pending', 'completed', 'cancelled')), `notes` (TEXT).
5. **property_media:** `media_id` (SERIAL PRIMARY KEY), `property_id` (INTEGER REFERENCES `properties` ON DELETE CASCADE), `media_type` (VARCHAR(30) CHECK `media_type` IN ('image', 'video', 'document')), `media_url` (TEXT), `upload_date` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что площадь положительна, цена положительна, статус из допустимого списка, а также что для продажи указаны и продавец, и покупатель.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового объекта недвижимости** (вставка в `properties`).
2. **Регистрация нового клиента.**
3. **Создание сделки** (вставка в `transactions` с проверкой существования объектов и клиентов).
4. **Обновление статуса сделки** (например, с 'pending' на 'completed').
5. **Обновление статуса объекта** (например, с 'active' на 'sold' при завершении продажи).
6. **Изменение цены объекта** (обновление поля `price`).
7. **Выборка всех активных объектов** с фильтрацией по типу, району, цене.
8. **Расчёт средней цены за квадратный метр** по районам и типам недвижимости.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Рыночная аналитика», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому району (или городу) и типу недвижимости за месяц:

- `region` (город/район);
- `property_type` (тип недвижимости);
- `report_month` (месяц отчёта);
- `average_price` — средняя цена объекта (всех активных и проданных);
- `average_price_per_sqm` — средняя цена за квадратный метр;
- `total_active_listings` — количество активных объектов;
- `total_transactions` — количество завершённых сделок;
- `total_revenue` — общая сумма сделок;
- `avg_days_on_market` — среднее время от публикации до сделки (для проданных объектов).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`properties`, `transactions`, `property_types`, `clients`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший месяц).
2. **Transform:** опишите преобразования:
 - объединение таблиц для получения полной информации об объектах и сделках;
 - расчёт времени на рынке (для завершённых сделок);
 - вычисление средней цены и цены за квадратный метр;
 - группировка по району, типу и месяцу.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `region`, `property_type`, `report_month`). Предложите механизм контроля качества (проверка, что `avg_price` ≥ 0 , `total_revenue` ≥ 0).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя агентства недвижимости, который позволит отслеживать рыночные тенденции и эффективность работы. Дашборд должен включать следующие элементы:

1. **KPI:** общее количество активных объектов, количество сделок за период, выручка от комиссий, средняя цена за квадратный метр, среднее время продажи.
2. **Динамика цен** — линейный график (средняя цена за месяц по основным типам недвижимости).
3. **Карта распределения цен** — географическая карта с ценовыми зонами (по районам).
4. **Распределение объектов по типам** — круговая или столбчатая диаграмма.
5. **Топ-5 районов по активности сделок** — гистограмма.
6. **Список объектов с наибольшим временем на рынке** — таблица для принятия решений (корректировка цены).
7. **Фильтры:** по типу недвижимости, по району, по периоду, по типу сделки (продажа/аренда).

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по району на карте открывает детальный отчёт по объектам в этом районе.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 17. Агентство недвижимости (с агентами)

Описание кейса

Агентство недвижимости (расширение варианта 16) работает с риэлторами (агентами), которые закреплены за объектами, клиентами и сделками. Каждый агент имеет свои показатели эффективности: количество заключённых сделок, выручка от комиссий, среднее время продажи. Руководство агентства хочет не только анализировать рыночные тенденции (цены, спрос), но и оценивать производительность каждого агента, распределять объекты и клиентов, управлять бонусами и мотивацией. Бизнес-цели:

- Оценка эффективности работы агентов (количество сделок, сумма комиссий, конверсия, время продажи).
- Анализ рынка недвижимости: динамика цен, популярность районов, типы объектов.
- Управление объектами: сроки публикации, снижение цены, рекомендации по корректировке.
- Мониторинг клиентской базы: активные покупатели/продавцы, повторные обращения.
- Прогнозирование спроса и оптимизация загрузки агентов.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных для учёта объектов, клиентов, сделок и агентов (риэлторов). Модель должна включать как минимум следующие сущности:

- **agents** — агенты (ФИО, контакты, дата найма, комиссионный процент, статус);
- **clients** — клиенты (ФИО, контакты, тип (покупатель, продавец, арендатор, арендодатель), предпочтения);
- **property_types** — справочник типов недвижимости (квартира, дом, офис и т.п.);
- **properties** — объекты недвижимости (адрес, характеристики, цена, статус, дата публикации, агент, ответственный за объект);
- **transactions** — сделки (объект, клиенты, агент, сумма, комиссия, дата, статус);
- **client_agent_assignment** — закрепление клиентов за агентами (для учёта клиентской базы).

Определите атрибуты, типы данных, ограничения. Предложите связи.

Примерная структура:

1. **agents**: `agent_id` (SERIAL PRIMARY KEY), `first_name`, `last_name`, `phone`, `email` (UNIQUE), `hire_date` (DATE), `commission_rate` (NUMERIC(5,2) DEFAULT 5.0), `status` (VARCHAR(20) CHECK status IN ('active','inactive','on_leave')).
2. **clients**: как в варианте 16, но добавить поле `preferred_agent_id` (INTEGER REFERENCES agents) и `registered_by_agent_id` (INTEGER REFERENCES agents).
3. **property_types**: как в варианте 16.

4. **properties**: добавить поле **responsible_agent_id** (INTEGER REFERENCES agents) — агент, отвечающий за объект; также **listed_date** DATE, **status** VARCHAR.
5. **transactions**: добавить поле **agent_id** (INTEGER REFERENCES agents) — агент, заключивший сделку; также **commission_amount** (NUMERIC) может вычисляться.
6. **client_agent_assignment**: для гибкости закрепления (один клиент может быть закреплён за несколькими агентами? Обычно за одним, но можно сделать таблицу связей).

Напишите SQL-скрипт создания таблиц с комментариями и ограничениями.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для:

1. Добавление нового агента.
2. Назначение агента на объект (обновление **responsible_agent_id**).
3. Регистрация нового клиента с указанием ответственного агента.
4. Создание сделки с указанием агента, автоматический расчёт комиссии.
5. Обновление статуса объекта и сделки.
6. Выборка объектов, закреплённых за конкретным агентом.
7. Отчёт по агентам: количество сделок, выручка за период.

Задание 3. ETL-процесс

Разработайте ETL для витрины «Эффективность агентов», обновляемой ежедневно. Витрина должна содержать по каждому агенту за месяц:

- **agent_id**, **agent_name**;
- **report_month**;
- **total_transactions** — количество завершённых сделок;
- **total_commission** — сумма комиссий;
- **avg_transaction_amount** — средняя сумма сделки;
- **total_properties_listed** — количество объектов, выставленных агентом;
- **avg_days_to_sell** — среднее время продажи объектов агентом;
- **conversion_rate** — отношение сделок к числу клиентов (или к объектам).

Опишите этапы ETL, используемые таблицы, преобразования и загрузку.

Задание 4. Дашборд

Разработайте макет дашборда для руководителя агентства, который включает:

- KPI: общая выручка от комиссий, количество агентов, количество активных объектов, средняя комиссия.
- Рейтинг агентов (по сумме комиссий, количеству сделок) — столбчатая диаграмма.
- Динамика сделок по агентам (линейный график).
- Распределение объектов по типам и агентам.
- Карта с объектами (по районам) с возможностью фильтра по агенту.
- Список агентов с низкой эффективностью (например, мало сделок).
- Фильтры: по агенту, по дате, по типу недвижимости.

Опишите визуализации и источники данных.

По результатам выполнения всех заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда.

Вариант 18. Рекламное агентство (кампании)

Описание кейса

Рекламное агентство проводит цифровые рекламные кампании для своих клиентов. Каждая кампания включает несколько рекламных объявлений (креативов), которые размещаются на различных площадках (веб-сайты, социальные сети, поисковые системы). Фиксируются показы (impressions), клики (clicks) и конверсии (целевые действия, например, покупки, регистрации). Агентство хочет анализировать эффективность кампаний, рассчитывать ROI (Return on Investment), оценивать стоимость клика (CPC), стоимость показа (CPM) и рентабельность. Бизнес-цели:

- Оценка эффективности рекламных кампаний (ROI, CPA, конверсия).
- Выявление наиболее успешных площадок, креативов и аудиторий.
- Оптимизация бюджета: перераспределение средств между кампаниями и каналами.
- Прогнозирование результатов на основе исторических данных.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных для учёта рекламных кампаний, объявлений, показов, кликов и конверсий. Модель должна включать как минимум:

- **campaigns** — рекламные кампании (название, бюджет, даты начала и окончания, целевая аудитория, статус);
- **ads** — объявления (креативы), принадлежащие кампании (текст, изображение, ссылка, тип);
- **publishers** — площадки (веб-сайты или платформы), где размещаются объявления;
- **impressions** — показы объявлений (дата и время, площадка, стоимость показа, ID объявления);
- **clicks** — клики (дата, время, стоимость клика, ID объявления, ID пользователя, если есть);
- **conversions** — конверсии (целевые действия, связанные с кликом, например, покупка, регистрация).

Определите атрибуты, типы, ограничения, связи.

Примерная структура:

1. **campaigns**: `campaign_id` SERIAL PRIMARY KEY, `name` VARCHAR(100) NOT NULL, `budget` NUMERIC(10,2) NOT NULL, `start_date` DATE, `end_date` DATE, `target_audience` TEXT, `status` VARCHAR(20) CHECK status IN ('draft', 'active', 'paused', 'completed').
2. **publishers**: `publisher_id` SERIAL PRIMARY KEY, `name` VARCHAR(100) NOT NULL, `url` VARCHAR(200), `type` VARCHAR(30) CHECK type IN ('website', 'social_media', 'search_engine').
3. **ads**: `ad_id` SERIAL PRIMARY KEY, `campaign_id` INTEGER REFERENCES campaigns, `title` VARCHAR(100), `content` TEXT, `image_url` VARCHAR(200), `click_url` VARCHAR(200), `ad_type` VARCHAR(30) CHECK ad_type IN ('banner', 'video', 'native', 'text').

4. `impressions: impression_id SERIAL PRIMARY KEY, ad_id INTEGER REFERENCES ads, publisher_id INTEGER REFERENCES publishers, impression_datetime TIMESTAMP DEFAULT CURRENT_TIMESTAMP, cost NUMERIC(10,4) DEFAULT 0.`
5. `clicks: click_id SERIAL PRIMARY KEY, ad_id INTEGER REFERENCES ads, publisher_id INTEGER REFERENCES publishers, click_datetime TIMESTAMP DEFAULT CURRENT_TIMESTAMP, user_id VARCHAR(50) (может быть анонимный ID), cost NUMERIC(10,4) DEFAULT 0.`
6. `conversions: conversion_id SERIAL PRIMARY KEY, click_id INTEGER REFERENCES clicks, conversion_datetime TIMESTAMP DEFAULT CURRENT_TIMESTAMP, conversion_type VARCHAR(30) CHECK conversion_type IN ('purchase', 'lead', 'registration', 'download'), revenue NUMERIC(10,2) DEFAULT 0 (доход от конверсии).`

Напишите SQL-скрипт для создания таблиц с комментариями и ограничениями (например, CHECK для положительных сумм, NOT NULL для обязательных полей, каскадные удаления).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для:

1. Создание новой кампании.
2. Добавление объявления в кампанию.
3. Регистрация показа и клика (вставка записей).
4. Фиксация конверсии (связать с кликом).
5. Обновление статуса кампании (например, на 'completed' по окончании).
6. Изменение бюджета кампании.
7. Выборка итоговой статистики по кампании: суммарные затраты, количество показов, кликов, конверсий, доход, ROI.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины «Эффективность рекламных кампаний» (ROI, CPA, CTR), которая обновляется ежедневно. Витрина должна содержать по каждой кампании и по дням:

- `campaign_id`, `campaign_name`;
- `date`;
- `impressions` — общее количество показов;
- `clicks` — количество кликов;
- `conversions` — количество конверсий;
- `total_cost` — суммарные затраты (показы + клики);
- `total_revenue` — сумма дохода от конверсий;

- `ctr` — click-through rate ($\text{clicks} / \text{impressions} * 100$);
- `conversion_rate` — conversions / clicks * 100;
- `cpc` — cost per click ($\text{total_cost} / \text{clicks}$, если $\text{clicks} > 0$);
- `cpa` — cost per acquisition ($\text{total_cost} / \text{conversions}$, если $\text{conversions} > 0$);
- `roi` — $(\text{total_revenue} - \text{total_cost}) / \text{total_cost} * 100$.

Опишите этапы ETL, используемые таблицы, преобразования и загрузку (инкрементально или полная перезагрузка). Приведите фрагменты SQL.

Задание 4. Дашборд

Разработайте макет дашборда для руководителя рекламного агентства, который позволит отслеживать эффективность кампаний. Дашборд должен включать:

- KPI: общие затраты, общий доход, средний ROI, общее количество конверсий, CTR.
- Динамика CTR и конверсий по дням (линейный график) с фильтром по кампании.
- Сравнение кампаний: столбчатая диаграмма (ROI, CPA, CPC) по каждой кампании.
- Распределение затрат по площадкам (круговая диаграмма).
- Топ-5 объявлений по CTR или по конверсиям (таблица).
- Календарь с отметкой активных кампаний.
- Фильтры: по дате, по кампании, по площадке, по типу конверсии.

Опишите визуализации и источники данных. Предложите интерактивные взаимодействия (например, клик по кампании открывает детальный отчёт).

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда.

Вариант 19. Социальная сеть (пользователи, посты)

Описание кейса

Социальная сеть предоставляет пользователям возможность создавать посты, ставить лайки, добавлять друг друга в друзья и комментировать записи. Платформа накапливает большие объёмы данных о поведении пользователей, которые могут быть использованы для анализа вовлечённости, выявления популярных тем, построения рекомендательных систем и обнаружения сообществ. Бизнес-цели:

- Мониторинг активности пользователей: количество постов, лайков, комментариев, новых друзей.
- Выявление наиболее активных пользователей и контента с высокой вовлечённостью.
- Анализ структуры социального графа (дружеских связей) для определения сообществ, влиятельных пользователей (инфлюенсеров).
- Оценка эффективности рекомендательных алгоритмов (если есть) и сегментация аудитории.
- Прогнозирование оттока пользователей на основе снижения активности.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для аналитиков и менеджеров продукта.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных для учёта пользователей, постов, лайков и дружеских связей. Модель должна включать как минимум следующие сущности:

- **users** — пользователи (идентификатор, имя, email, дата регистрации, статус, геолокация, дата рождения и т.п.);
- **posts** — посты (идентификатор, автор, текст, дата публикации, тип (текст, фото, видео), количество просмотров);
- **likes** — лайки (идентификатор, пользователь, пост, дата);
- **friendships** — дружеские связи (два пользователя, дата установления связи, статус (подтверждена, ожидание));
- **comments** — комментарии (опционально, но можно добавить для полноты; если ограничиться четырьмя таблицами, то без комментариев).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **users**: `user_id` (SERIAL PRIMARY KEY), `username` (VARCHAR(50) UNIQUE NOT NULL), `email` (VARCHAR(100) UNIQUE NOT NULL), `full_name` (VARCHAR(100)), `registration_date` (DATE DEFAULT CURRENT_DATE), `last_active_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('active', 'inactive', 'blocked')), `location` (VARCHAR(100)), `birth_date` (DATE).

2. **posts**: `post_id` (SERIAL PRIMARY KEY), `user_id` (INTEGER REFERENCES users ON DELETE CASCADE), `content` (TEXT NOT NULL), `post_type` (VARCHAR(20) CHECK `post_type` IN ('text', 'photo', 'video', 'link')), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `views_count` (INTEGER DEFAULT 0).
3. **likes**: `like_id` (SERIAL PRIMARY KEY), `user_id` (INTEGER REFERENCES users ON DELETE CASCADE), `post_id` (INTEGER REFERENCES posts ON DELETE CASCADE), `liked_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP). Уникальность по паре (`user_id`, `post_id`) для предотвращения повторных лайков.
4. **friendships**: `friendship_id` (SERIAL PRIMARY KEY), `user_id_1` (INTEGER REFERENCES users ON DELETE CASCADE), `user_id_2` (INTEGER REFERENCES users ON DELETE CASCADE), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `status` (VARCHAR(20) CHECK `status` IN ('pending', 'accepted', 'blocked')), `updated_at` (TIMESTAMP). Уникальность по паре (`user_id_1`, `user_id_2`) с проверкой, что `user_id_1` < `user_id_2` (или любое ограничение) для избежания дублирования.
5. **comments** (опционально): `comment_id` (SERIAL PRIMARY KEY), `user_id` (INTEGER REFERENCES users), `post_id` (INTEGER REFERENCES posts), `content` (TEXT NOT NULL), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что текст поста не может быть пустым, статусы из допустимых списков, а также уникальные ограничения для предотвращения дублирования лайков и дружеских связей.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Регистрация нового пользователя** (вставка в `users`).
2. **Создание нового поста** пользователем.
3. **Постановка лайка** (вставка в `likes` с проверкой, что пользователь не лайкал этот пост ранее).
4. **Удаление лайка** (удаление записи из `likes`).
5. **Добавление друга** (вставка в `friendships` со статусом 'pending' или 'accepted').
6. **Подтверждение дружбы** (обновление статуса `friendships` на 'accepted').
7. **Удаление пользователя** (каскадное удаление его постов, лайков, дружеских связей — через ON DELETE CASCADE).
8. **Выборка топ-10 постов по количеству лайков за последнюю неделю**.
9. **Выборка друзей пользователя** (включая тех, кто подтвердил дружбу) — через запрос к `friendships`.
10. **Расчёт среднего количества постов на пользователя за день** (группировка по дате).

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Активность пользователей и вовлечённость», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому пользователю за день (или за месяц):

- `user_id, username`;
- `date` (отчётная дата);
- `posts_created` — количество созданных постов;
- `likes_received` — количество лайков, полученных постами пользователя (через посты);
- `likes_given` — количество лайков, поставленных пользователем;
- `comments_given` — количество комментариев, оставленных пользователем (если есть);
- `new_friends` — количество новых дружеских связей (подтверждённых) за день;
- `total_friends` — общее количество друзей (накопленное).

Также можно добавить агрегацию по времени суток, по типу постов и другие.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`users`, `posts`, `likes`, `friendships`, `comments`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - подсчёт количества постов и лайков по пользователям и датам;
 - расчёт лайков, полученных каждым пользователем (через `join` постов и лайков);
 - подсчёт количества новых друзей (по дате создания связи);
 - объединение всех агрегатов по пользователю и дате.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `user_id, date`). Предложите механизм контроля качества (например, проверка, что количество лайков не превышает количество постов, и т.п.).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для аналитика социальной сети, который позволит отслеживать активность и вовлечённость пользователей. Дашборд должен включать следующие элементы:

1. **KPI:** общее количество активных пользователей за день, количество новых постов, количество лайков, количество новых дружеских связей, среднее количество постов на пользователя.

2. **Динамика активности** — линейный график (количество постов, лайков, комментариев по дням за последние 30 дней) с возможностью выбора метрики.
3. **Топ-5 пользователей по полученным лайкам** и **топ-5 по созданным постам** — столбчатые диаграммы.
4. **Граф друзей** — визуализация социального графа для выбранного пользователя (с использованием библиотек, например, D3). Можно показать просто таблицу с количеством друзей у пользователей.
5. **Распределение активности по времени суток** — тепловая карта (часы x дни недели) для выявления пиковых нагрузок.
6. **Список пользователей с наибольшим снижением активности** (чтобы выявить риск оттока) — таблица.
7. **Фильтры:** по дате, по географическому региону (если есть), по типу постов.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по пользователю открывает его профиль с детальной статистикой.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 20. Спортивный клуб (членство, тренировки)

Описание кейса

Спортивный клуб предлагает различные групповые и индивидуальные тренировки. Клиенты клуба оформляют членство, записываются на тренировки, посещают их. Тренеры проводят занятия, каждый тренер специализируется на определённых типах тренировок (йога, пилатес, кроссфит, силовые, кардио и т.п.). Клуб хочет анализировать посещаемость, загруженность тренеров, популярность занятий, удержание клиентов и выявлять сезонные тенденции. Бизнес-цели:

- Мониторинг загруженности тренеров (количество проведённых занятий, среднее количество участников, заполняемость).
- Анализ популярности различных типов тренировок.
- Оценка удержания клиентов (частотность посещений, отток).
- Планирование расписания на основе спроса и загруженности залов/тренеров.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства клуба.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта членов клуба, тренеров, типов тренировок, расписания и посещений. Модель должна включать как минимум следующие сущности:

- **members** — члены клуба (ФИО, контакты, дата регистрации, тип членства, статус);
- **trainers** — тренеры (ФИО, специализация, стаж, контакты, статус);
- **workout_types** — типы тренировок (название, описание, длительность, максимальное количество участников);
- **sessions** — тренировочные сессии (тип тренировки, тренер, дата и время, длительность, статус);
- **attendance** — посещения (член клуба, сессия, дата посещения, отметка (пришёл/не пришёл), оценка занятия (опционально)).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **members**: `member_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `registration_date` (DATE DEFAULT CURRENT_DATE), `membership_type` (VARCHAR(30) CHECK membership_type IN ('monthly', 'quarterly', 'yearly', 'single')), `status` (VARCHAR(20) CHECK status IN ('active', 'suspended', 'expired')), `date_of_birth` (DATE).

2. **trainers:** `trainer_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `specialization` (VARCHAR(100)), `phone`, `email` (UNIQUE), `hire_date` (DATE), `experience_years` (INTEGER CHECK ≥ 0), `status` (VARCHAR(20) CHECK status IN ('active', 'on_leave', 'terminated')).
3. **workout_types:** `type_id` (SERIAL PRIMARY KEY), `type_name` (VARCHAR(100) NOT NULL), `description` (TEXT), `default_duration_minutes` (INTEGER CHECK > 0), `max_participants` (INTEGER CHECK > 0).
4. **sessions:** `session_id` (SERIAL PRIMARY KEY), `type_id` (INTEGER REFERENCES workout_types), `trainer_id` (INTEGER REFERENCES trainers), `start_datetime` (TIMESTAMP NOT NULL), `end_datetime` (TIMESTAMP GENERATED ALWAYS AS (`start_datetime` + (`default_duration_minutes` * INTERVAL '1 minute')) STORED — или просто вычисляемое, можно не хранить), `status` (VARCHAR(20) CHECK status IN ('scheduled', 'in_progress', 'cancelled')), `room` (VARCHAR(30)).
5. **attendance:** `attendance_id` (SERIAL PRIMARY KEY), `member_id` (INTEGER REFERENCES members ON DELETE CASCADE), `session_id` (INTEGER REFERENCES sessions ON DELETE CASCADE), `check_in_time` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `status` (VARCHAR(20) CHECK status IN ('present', 'absent', 'late')), `feedback` (INTEGER CHECK feedback BETWEEN 1 AND 5) — оценка занятия.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что длительность и максимальное количество участников положительные, статусы из допустимых списков, а также уникальность email для членов и тренеров.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового члена клуба.**
2. **Добавление нового тренера.**
3. **Создание новой тренировки** (вставка в sessions с проверкой существующих типа и тренера).
4. **Запись члена клуба на тренировку** (вставка в attendance со статусом 'absent' по умолчанию или 'present' после отметки).
5. **Отметка посещения** (обновление статуса attendance на 'present' или 'late').
6. **Отмена тренировки** (обновление статуса sessions на 'cancelled').
7. **Удаление члена клуба** (при условии отсутствия будущих записей).
8. **Выборка всех активных членов** с количеством посещений за последний месяц.
9. **Выборка тренера с наибольшим количеством проведённых занятий за месяц.**

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Загруженность тренеров и популярность занятий», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому тренеру и типу тренировки за день (или за неделю):

- `trainer_id`, `trainer_name`;
- `workout_type_id`, `workout_type_name`;
- `date` (отчётная дата);
- `total_sessions` — количество проведённых сессий;
- `total_participants_booked` — общее количество записавшихся (сумма по `attendance` для всех сессий);
- `total_participants_present` — количество фактически присутствовавших;
- `avg_participants_present` — среднее количество присутствующих на сессию;
- `occupancy_rate` — процент заполняемости ($\text{present} / \text{max_participants} * 100$);
- `avg_feedback` — средняя оценка занятий (из `feedback`).

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`sessions`, `attendance`, `trainers`, `workout_types`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение сессий с тренерами, типами и посещениями;
 - фильтрация по статусу сессий (только `'completed'`);
 - агрегация (группировка по тренеру, типу и дате);
 - вычисление среднего количества присутствующих и заполняемости (для заполняемости нужно знать `max_participants` из `workout_types`);
 - расчёт средней оценки.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `trainer_id`, `workout_type_id`, `date`). Предложите механизм контроля качества (проверка, что `occupancy_rate` между 0 и 100, `avg_feedback` в диапазоне 1-5).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для управляющего спортивным клубом, который позволит отслеживать ключевые показатели работы. Дашборд должен включать следующие элементы:

1. **KPI:** общее количество активных членов, количество проведённых занятий за сегодня/неделю, средняя заполняемость, средняя оценка занятий, общее количество тренеров.
2. **Загруженность тренеров** — столбчатая диаграмма (количество занятий и среднее количество участников по каждому тренеру) с возможностью фильтрации по типу занятий.
3. **Динамика посещаемости** — линейный график (количество присутствующих по дням за последние 30 дней) с разбивкой по типам тренировок.
4. **Популярность типов тренировок** — круговая диаграмма (распределение занятий по типам).
5. **Тренерский рейтинг** — таблица тренеров, отсортированных по средней оценке или по заполняемости.
6. **Члены с наибольшей посещаемостью** — таблица активных членов для поощрения.
7. **Фильтры:** по дате, по тренеру, по типу тренировки, по статусу членства.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по тренеру открывает детальный отчёт по его занятиям.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 21. Ресторан (заказы блюд)

Описание кейса

Ресторан ведёт учёт заказов посетителей, блюд и продаж. Каждый заказ связан с клиентом (или гостем), содержит несколько позиций (блюда) с указанием количества и цены на момент заказа. Блюда имеют название, категорию, стоимость приготовления, цену продажи. Ресторан хочет анализировать продажи, популярность блюд, выручку по дням, оценивать маржинальность и эффективность работы официантов (если есть). Бизнес-цели:

- Анализ выручки по дням, неделям, месяцам.
- Выявление самых популярных блюд (по количеству заказов, по сумме продаж).
- Оценка маржинальности блюд (цена продажи - себестоимость).
- Сегментация клиентов по частоте и сумме заказов.
- Прогнозирование спроса для планирования закупок.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для управляющего рестораном.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта клиентов, блюд, заказов и позиций заказа. Модель должна включать как минимум следующие сущности:

- **dishes** — блюда (название, категория, цена продажи, себестоимость, статус);
- **categories** — категории блюд (например, супы, салаты, горячее, напитки);
- **customers** — клиенты (ФИО, телефон, email, дата регистрации) — можно объединить с гостями (без регистрации), но для аналитики удобно;
- **orders** — заказы (клиент, дата и время, статус, общая сумма, официант (опционально));
- **order_items** — позиции заказа (заказ, блюдо, количество, цена на момент заказа).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **categories**: `category_id` (SERIAL PRIMARY KEY), `category_name` (VARCHAR(100) NOT NULL UNIQUE), `description` (TEXT).
2. **dishes**: `dish_id` (SERIAL PRIMARY KEY), `name` (VARCHAR(100) NOT NULL), `category_id` (INTEGER REFERENCES categories), `selling_price` (NUMERIC(10,2) NOT NULL CHECK >0), `cost_price` (NUMERIC(10,2) NOT NULL CHECK >0), `status` (VARCHAR(20) CHECK status IN ('available', 'unavailable', 'discontinued')), `description` (TEXT).

3. `customers: customer_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20) UNIQUE), `email` (VARCHAR(100) UNIQUE), `registration_date` (DATE DEFAULT CURRENT_DATE), `total_spent` (NUMERIC(12,2) DEFAULT 0) — можно вычислять, но для удобства хранить.
4. `orders: order_id` (SERIAL PRIMARY KEY), `customer_id` (INTEGER REFERENCES customers), `order_datetime` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `status` (VARCHAR(20) CHECK status IN ('new', 'preparing', 'served', 'paid', 'cancelled')), `total_amount` (NUMERIC(10,2) NOT NULL DEFAULT 0), `waiter_name` (VARCHAR(50)) — опционально.
5. `order_items: item_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders ON DELETE CASCADE), `dishe_id` (INTEGER REFERENCES dishes), `quantity` (INTEGER NOT NULL CHECK >0), `unit_price` (NUMERIC(10,2) NOT NULL CHECK >0) — цена на момент заказа, `total_price` (NUMERIC(10,2) GENERATED ALWAYS AS (quantity * unit_price) STORED) — можно вычислять.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что цены положительные, статусы из допустимых списков, а также уникальность названия категории и блюда (если требуется).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового блюда** с указанием категории.
2. **Регистрация нового клиента.**
3. **Создание нового заказа** (вставка в orders) с автоматическим расчётом общей суммы (можно через триггер или вручную при добавлении позиций).
4. **Добавление позиции в заказ** (вставка в order_items) с указанием блюда, количества и цены.
5. **Обновление статуса заказа** (например, на 'paid' или 'served').
6. **Отмена заказа** (установка статуса 'cancelled').
7. **Выборка всех заказов за сегодня** с указанием клиента, общей суммы и статуса.
8. **Расчёт выручки за день** (сумма total_amount для заказов со статусом 'paid' за текущую дату).
9. **Топ-5 самых продаваемых блюд за неделю** (по сумме количеств).

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Продажи и популярность блюд», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому блюду (или категории) за день (или за месяц):

- `dishe_id`, `dishe_name`, `category`;
- `date` (отчётная дата);
- `total_orders` — количество заказов, включающих это блюдо;
- `total_quantity` — общее количество проданных порций;
- `total_revenue` — общая выручка от продажи этого блюда (сумма `total_price` по `order_items`);
- `avg_price` — средняя цена продажи за порцию ($\text{revenue} / \text{quantity}$, если $\text{quantity} > 0$);
- `total_cost` — общая себестоимость ($\text{quantity} * \text{cost_price}$, где `cost_price` из таблицы `dishes` на дату отчёта, либо может быть историческая);
- `margin` — маржинальность ($\text{revenue} - \text{cost}$).

Также можно добавить агрегацию по клиентам (сколько уникальных клиентов заказывали блюдо) и т.п.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`orders`, `order_items`, `dishes`, `categories`, `customers`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение `order_items` с заказами (фильтр по статусу `'paid'`), блюдами и категориями;
 - агрегация по блюдам и датам (суммирование количества, выручки);
 - вычисление себестоимости (можно использовать `cost_price` из `dishes` на текущий момент, но лучше историческую — если есть изменения цены, нужно хранить историю себестоимости; для упрощения можно использовать текущую или хранить в `order_items` отдельно).
 - расчёт маржи.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `dishe_id`, `date`). Предложите механизм контроля качества (проверка, что `revenue` не отрицательная, `margin` ≥ 0 и т.д.).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для управляющего рестораном, который позволит отслеживать продажи и популярность блюд. Дашборд должен включать следующие элементы:

1. **KPI:** общая выручка за сегодня/неделю/месяц, количество заказов, средний чек, количество уникальных клиентов, доля наличных/безналичных (если есть данные).
2. **Динамика выручки** — линейный график по дням за последние 30 дней.
3. **Распределение продаж по категориям** — круговая диаграмма.
4. **Топ-10 блюд по выручке** — столбчатая диаграмма.
5. **Маржинальность блюд** — точечная диаграмма (цена vs себестоимость) или таблица с самыми маржинальными и низкомаржинальными блюдами.
6. **Часовая динамика заказов** — тепловая карта (часы дня × дни недели) для выявления пиковых нагрузок.
7. **Список заказов с высокой суммой** — таблица для детального анализа.
8. **Фильтры:** по дате, по категории блюда, по статусу заказа.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по блюду открывает детальную историю его продаж.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 22. Библиотека (книги, читатели)

Описание кейса

Библиотека ведёт учёт книг, читателей и выдачи книг. Каждая книга имеет название, автора, ISBN, жанр, год издания, количество экземпляров в библиотеке. Читатели регистрируются, получают читательский билет. При выдаче книги фиксируется дата выдачи и плановая дата возврата. При возврате фиксируется фактическая дата, вычисляется просрочка (если есть). Библиотека хочет анализировать популярность книг, задолженность читателей, эффективность работы библиотеки. Бизнес-цели:

- Мониторинг книговыдачи: количество выдач в день/неделю/месяц.
- Выявление самых популярных книг и авторов.
- Контроль просроченных книг и задолжников.
- Планирование закупок новых книг на основе спроса.
- Анализ читательской активности (частотность, предпочтения).

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для заведующего библиотекой.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта книг, читателей и выдач. Модель должна включать как минимум следующие сущности:

- **books** — книги (название, автор, ISBN, жанр, год, общее количество экземпляров, доступные экземпляры);
- **authors** — авторы (имя, фамилия, страна) — опционально, можно хранить как строку в книгах, но лучше нормализовать;
- **genres** — жанры (название, описание) — опционально;
- **readers** — читатели (ФИО, телефон, email, дата регистрации, адрес);
- **loans** — выдачи (книга, читатель, дата выдачи, плановая дата возврата, фактическая дата возврата, статус, штраф).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **authors**: `author_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `country` (VARCHAR(100)) — опционально.
2. **genres**: `genre_id` (SERIAL PRIMARY KEY), `genre_name` (VARCHAR(100) NOT NULL UNIQUE), `description` (TEXT).

3. **books:** `book_id` (SERIAL PRIMARY KEY), `title` (VARCHAR(200) NOT NULL), `author_id` (INTEGER REFERENCES authors), `genre_id` (INTEGER REFERENCES genres), `isbn` (VARCHAR(20) UNIQUE), `publication_year` (INTEGER CHECK ≥ 1800), `total_copies` (INTEGER NOT NULL CHECK ≥ 0), `available_copies` (INTEGER NOT NULL CHECK ≥ 0) — можно вычислять, но для удобства хранить.
4. **readers:** `reader_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `registration_date` (DATE DEFAULT CURRENT_DATE), `address` (TEXT), `status` (VARCHAR(20) CHECK status IN ('active', 'suspended', 'inactive')).
5. **loans:** `loan_id` (SERIAL PRIMARY KEY), `book_id` (INTEGER REFERENCES books), `reader_id` (INTEGER REFERENCES readers), `loan_date` (DATE NOT NULL), `due_date` (DATE NOT NULL, CHECK `due_date` $\geq$ `loan_date`), `return_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('issued', 'returned', 'overdue') GENERATED ALWAYS AS (CASE WHEN `return_date` IS NULL AND `due_date` < CURRENT_DATE THEN 'overdue' WHEN `return_date` IS NOT NULL THEN 'returned' ELSE 'issued' END) STORED — в PostgreSQL можно вычислить, или задавать вручную), `fine` (NUMERIC(10,2) DEFAULT 0) — штраф за просрочку.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что ISBN уникален, `total_copies` и `available_copies` неотрицательные, статусы из допустимых списков. Для обновления `available_copies` можно использовать триггеры, но в рамках задания достаточно описать логику.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление новой книги** с указанием автора и жанра (предварительно добавив их, если их нет).
2. **Регистрация нового читателя.**
3. **Выдача книги читателю** (вставка в `loans`, уменьшение `available_copies` на 1).
4. **Возврат книги** (обновление `return_date`, увеличение `available_copies` на 1, расчёт штрафа за просрочку, если фактическая дата возврата > `due_date`).
5. **Обновление статуса читателя** (например, на 'suspended' при нарушении).
6. **Удаление книги** (если нет активных выдач).
7. **Выборка всех книг, взятых читателем** с указанием даты выдачи и статуса.
8. **Поиск книг с наибольшим количеством выдач за последний месяц.**
9. **Список читателей, имеющих просроченные книги** (`loans` со статусом 'overdue').

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Популярность книг и задолженность читателей», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждой книге (или по каждому читателю) за месяц (или за день):

- `book_id`, `title`, `author`, `genre`;
- `report_month` (месяц отчёта);
- `total_loans` — общее количество выдач за период;
- `total_readers` — количество уникальных читателей, бравших книгу;
- `avg_loan_duration` — средняя длительность выдачи (в днях) для возвращённых книг;
- `on_loan_count` — количество экземпляров, находящихся на руках на конец периода;
- `overdue_count` — количество просроченных экземпляров на конец периода;
- `total_fines` — сумма штрафов, начисленных за период.

Также можно добавить агрегацию по читателям: количество взятых книг, просрочек, сумму штрафов.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`loans`, `books`, `authors`, `genres`, `readers`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение `loans` с книгами, авторами, жанрами;
 - вычисление длительности выдачи (для возвращённых);
 - фильтрация по дате выдачи (за период);
 - агрегация по книгам (или по читателям) и по месяцам;
 - подсчёт просроченных на конец периода (можно использовать статус или сравнение `due_date` с датой отчёта).
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `book_id`, `report_month`). Предложите механизм контроля качества (проверка, что `avg_loan_duration` ≥ 0 , `total_fines` ≥ 0).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для заведующего библиотекой, который позволит отслеживать популярность книг и задолженность. Дашборд должен включать следующие элементы:

1. **KPI:** общее количество выдач за день/неделю, количество просроченных книг, общая сумма штрафов, количество активных читателей.
2. **Топ-10 книг по выдаче** — столбчатая диаграмма.
3. **Динамика выдач** — линейный график по дням за последние 30 дней.
4. **Распределение выдач по жанрам** — круговая диаграмма.
5. **Список задолжников** — таблица с читателями, имеющими просроченные книги, с указанием книги, срока просрочки и штрафа.
6. **Статистика по авторам** — столбчатая диаграмма (количество выдач по авторам).
7. **Фильтры:** по дате, по жанру, по автору, по статусу читателя.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по книге открывает детальную историю её выдач.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 23. Автомастерская (ремонт)

Описание кейса

Автомастерская предоставляет услуги по ремонту и техническому обслуживанию автомобилей. Ведётся учёт клиентов, их автомобилей, механиков (сотрудников), видов ремонтных работ (услуг), а также заказов на ремонт. Каждый заказ включает один или несколько видов работ, выполняется определённым механиком (или командой), имеет дату начала и окончания, общую стоимость и статус. Клиенты могут иметь несколько автомобилей. Механики имеют специализацию и уровень квалификации. Бизнес-цели:

- Мониторинг доходов от ремонта (по дням, неделям, месяцам, по типам услуг).
- Оценка загруженности механиков (количество заказов, время выполнения, средняя выручка на механика).
- Выявление самых востребованных услуг и типов ремонта.
- Анализ клиентов (частота обращений, средний чек, предпочтения).
- Планирование загрузки мастерской, управление очередью заказов.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для управляющего мастерской.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта клиентов, автомобилей, механиков, услуг и заказов. Модель должна включать как минимум следующие сущности:

- **customers** — клиенты (владельцы авто) — ФИО, телефон, email, адрес;
- **cars** — автомобили (госномер, марка, модель, год выпуска, VIN, клиент-владелец);
- **mechanics** — механики (ФИО, специализация, стаж, статус, контакты);
- **services** — виды ремонтных работ (название, описание, стандартная стоимость, длительность в часах);
- **orders** — заказы на ремонт (автомобиль, дата поступления, плановая дата завершения, фактическая дата завершения, статус, общая стоимость);
- **order_services** — услуги, включённые в заказ (заказ, услуга, количество, цена, механик (кто выполнял), дата выполнения).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **customers**: **customer_id** (SERIAL PRIMARY KEY), **first_name** (VARCHAR(50) NOT NULL), **last_name** (VARCHAR(50) NOT NULL), **phone** (VARCHAR(20) NOT NULL), **email** (VARCHAR(100) UNIQUE), **address** (TEXT), **registration_date** (DATE DEFAULT CURRENT_DATE).

2. **cars:** `car_id` (SERIAL PRIMARY KEY), `license_plate` (VARCHAR(15) UNIQUE NOT NULL), `brand` (VARCHAR(50) NOT NULL), `model` (VARCHAR(50) NOT NULL), `year` (INTEGER CHECK > 1980), `vin` (VARCHAR(17) UNIQUE), `customer_id` (INTEGER REFERENCES customers ON DELETE CASCADE).
3. **mechanics:** `mechanic_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20)), `specialization` (VARCHAR(100)), `experience_years` (INTEGER CHECK >=0), `status` (VARCHAR(20) CHECK status IN ('active', 'on_leave', 'terminated')).
4. **services:** `service_id` (SERIAL PRIMARY KEY), `service_name` (VARCHAR(100) NOT NULL), `description` (TEXT), `standard_cost` (NUMERIC(10,2) NOT NULL CHECK >0), `standard_duration_hours` (NUMERIC(5,2) NOT NULL CHECK >0).
5. **orders:** `order_id` (SERIAL PRIMARY KEY), `car_id` (INTEGER REFERENCES cars), `order_date` (DATE DEFAULT CURRENT_DATE), `planned_completion_date` (DATE), `actual_completion_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('new', 'in_progress'), `total_cost` (NUMERIC(12,2) DEFAULT 0) — можно вычислять как сумму cost из order_services.
6. **order_services:** `order_service_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders ON DELETE CASCADE), `service_id` (INTEGER REFERENCES services), `mechanic_id` (INTEGER REFERENCES mechanics) — кто выполнил, `quantity` (INTEGER DEFAULT 1 CHECK >0), `price` (NUMERIC(10,2) NOT NULL) — цена за единицу, `duration_hours` (NUMERIC(5,2)) — фактическая длительность, `execution_date` (DATE) — дата выполнения.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что цена, длительность и количество положительные, статусы из допустимых списков, а также уникальность госномера и VIN.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового клиента.**
2. **Регистрация автомобиля** для клиента.
3. **Добавление нового механика.**
4. **Создание нового заказа** (вставка в orders) с указанием автомобиля, даты начала и планового завершения.
5. **Добавление услуги в заказ** (вставка в order_services) с указанием механика, цены, количества и длительности.
6. **Обновление статуса заказа** (например, с 'in_progress' на 'completed') и установка фактической даты завершения.
7. **Расчёт общей стоимости заказа** — можно сделать через триггер или вручную, но нужно обновить total_cost в orders.

8. **Выборка всех заказов за период** с указанием клиента, машины, услуг, механиков и стоимости.
9. **Топ-3 самых востребованных услуг** за месяц (по количеству заказов или по сумме).

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Эффективность мастерской и загруженность механиков», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому механику (или по дням) за месяц (или за день):

- `mechanic_id`, `mechanic_name`;
- `report_month` (месяц отчёта);
- `total_orders` — количество заказов, в которых участвовал механик (уникальные `order_id`);
- `total_services` — общее количество выполненных услуг (сумма `quantity`);
- `total_hours` — общее количество отработанных часов (сумма `duration_hours`);
- `total_revenue` — общая выручка, принесённая механиком (сумма `price * quantity`);
- `avg_order_cost` — средняя стоимость заказа, в котором участвовал механик;
- `avg_service_duration` — средняя длительность выполнения одной услуги.

Также можно добавить агрегацию по услугам (популярность) и по клиентам.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`orders`, `order_services`, `services`, `mechanics`, `cars`, `customers`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение `order_services` с заказами, механиками, услугами;
 - фильтрация по статусу заказа (только `'completed'` или `'in_progress'` для текущей оценки);
 - агрегация по механикам и по месяцам;
 - вычисление средних значений.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `mechanic_id`, `report_month`). Предложите механизм контроля качества (проверка, что `total_hours` ≥ 0 , `avg_service_duration` ≥ 0 и т.д.).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для управляющего автомастерской, который позволит отслеживать доходы и загруженность. Дашборд должен включать следующие элементы:

1. **KPI:** общая выручка за месяц, количество завершённых заказов, средняя стоимость заказа, количество услуг, выполненных за месяц, загруженность механиков (среднее количество часов в день).
2. **Динамика выручки** — линейный график по дням за последние 30 дней.
3. **Распределение выручки по услугам** — круговая диаграмма.
4. **Загруженность механиков** — столбчатая диаграмма (количество часов или выручка по каждому механику).
5. **Топ-5 услуг по популярности** — гистограмма.
6. **Клиенты с наибольшими суммами** — таблица топ-клиентов.
7. **Список заказов в работе** (статус 'in_progress') с указанием планового завершения.
8. **Фильтры:** по дате, по механику, по типу услуги.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по механику открывает детальный отчёт по его заказам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 24. Страховая компания (полисы)

Описание кейса

Страховая компания предлагает различные виды страхования: автострахование (ОСАГО, КАСКО), страхование жизни, здоровья, имущества, ответственности. Ведётся учёт клиентов, страховых полисов, страховых случаев (событий) и выплат. Каждый полис имеет тип, срок действия, страховую сумму, премию (взнос), статус. Клиенты могут иметь несколько полисов. При наступлении страхового случая оформляется заявка (claim), содержащая информацию о событии, дате, описании, сумме ущерба и статусе рассмотрения. После принятия решения о выплате производится выплата (полностью или частично). Компания хочет анализировать:

- Частоту страховых случаев по типам полисов, регионам, клиентам.
- Суммы выплат и убыточность видов страхования.
- Оценку рисков для определения тарифов и условий.
- Эффективность работы страховых агентов (если есть).
- Прогнозирование будущих выплат на основе истории.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководителя страховой компании.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта клиентов, полисов, страховых случаев и выплат. Модель должна включать как минимум следующие сущности:

- **clients** — клиенты (физические или юридические лица) — ФИО/название, контакты, адрес, дата регистрации, тип клиента;
- **policy_types** — типы страховых продуктов (название, описание, стандартные условия);
- **policies** — полисы (клиент, тип, страховая сумма, премия, дата начала, дата окончания, статус, регион, агент (опционально));
- **claims** — страховые случаи (полис, дата события, описание, оценка ущерба, статус заявки, дата подачи);
- **payments** — выплаты по страховым случаям (claim, сумма выплаты, дата выплаты, статус).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **clients**: `client_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50)), `last_name` (VARCHAR(50)), `company_name` (VARCHAR(100)) — для юрлиц, `client_type` (VARCHAR(20) CHECK client_type IN ('individual','company')), `phone`, `email` (UNIQUE), `address` (TEXT), `registration_date` (DATE DEFAULT CURRENT_DATE).

2. **policy_types**: **type_id** (SERIAL PRIMARY KEY), **type_name** (VARCHAR(100) UNIQUE NOT NULL), **description** (TEXT), **standard_rate** (NUMERIC(5,2)) — стандартный тариф (процент от страховой суммы).
3. **policies**: **policy_id** (SERIAL PRIMARY KEY), **client_id** (INTEGER REFERENCES clients), **type_id** (INTEGER REFERENCES policy_types), **sum_insured** (NUMERIC(12,2) NOT NULL CHECK >0), **premium** (NUMERIC(12,2) NOT NULL CHECK >0) — страховой взнос, **start_date** (DATE NOT NULL), **end_date** (DATE NOT NULL, CHECK end_date >= start_date), **status** (VARCHAR(20) CHECK status IN ('active', 'expired', 'cancelled')), **region** (VARCHAR(100)).
4. **claims**: **claim_id** (SERIAL PRIMARY KEY), **policy_id** (INTEGER REFERENCES policies), **event_date** (DATE NOT NULL), **report_date** (DATE DEFAULT CURRENT_DATE), **description** (TEXT), **estimated_loss** (NUMERIC(12,2) NOT NULL CHECK >=0), **status** (VARCHAR(20) CHECK status IN ('submitted', 'under_review', 'approved', 'rejected', 'paid')), **review_notes** (TEXT).
5. **payments**: **payment_id** (SERIAL PRIMARY KEY), **claim_id** (INTEGER REFERENCES claims), **payment_amount** (NUMERIC(12,2) NOT NULL CHECK >0), **payment_date** (DATE DEFAULT CURRENT_DATE), **status** (VARCHAR(20) CHECK status IN ('scheduled', 'completed', 'failed')).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что страховая сумма и премия положительные, даты корректны, статусы из допустимых списков.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового клиента.**
2. **Создание нового полиса** для клиента с указанием типа, страховой суммы, премии, сроков.
3. **Регистрация страхового случая** (вставка в claims) с указанием полиса, даты события, описания и предполагаемого ущерба.
4. **Обновление статуса страхового случая** (например, с 'submitted' на 'approved').
5. **Выплата по страховому случаю** (вставка в payments) и обновление статуса claim на 'paid'.
6. **Отказ в выплате** (обновление статуса claim на 'rejected').
7. **Выборка всех полисов клиента** с истёкшим сроком действия (expired).
8. **Выборка страховых случаев за период** с группировкой по типам полисов и расчётом общей суммы выплат.
9. **Топ-5 клиентов по сумме выплат** за последний год.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Убыточность и риски по видам страхования», которая обновляется ежемесячно. Целевая витрина должна содержать следующие показатели по каждому типу полиса (или по регионам) за месяц:

- `policy_type_id`, `type_name`;
- `report_month` (месяц отчёта);
- `total_policies_active` — количество активных полисов на начало/конец месяца (или среднее);
- `total_premium_collected` — сумма собранных страховых взносов за месяц (для новых или активных полисов);
- `total_claims_submitted` — количество поданных заявок;
- `total_claims_approved` — количество одобренных заявок;
- `total_claims_paid` — количество оплаченных заявок;
- `total_payments` — общая сумма выплат;
- `loss_ratio` — отношение выплат к собранным премиям ($\text{total_payments} / \text{total_premium_collected} * 100$);
- `avg_claim_amount` — средняя сумма выплаты на одну заявку.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`policies`, `claims`, `payments`, `clients`, `policy_types`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший месяц).
2. **Transform:** опишите преобразования:
 - соединение полисов, заявок, выплат и типов;
 - фильтрация по датам (например, дата платежа или дата заявки);
 - расчёт премий, собранных за месяц (можно использовать полисы, начавшиеся в этом месяце, или активные, но лучше использовать фактическую оплату премий, если есть таблица оплат; для упрощения можно использовать премию по полисам, действовавшим в месяце);
 - агрегация по типу полиса и месяцу;
 - вычисление `loss_ratio` и средней суммы.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `type_id`, `report_month`). Предложите механизм контроля качества (проверка, что `loss_ratio` между 0 и 1, `total_payments` ≥ 0 и т.д.).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя страховой компании, который позволит отслеживать убыточность и риски. Дашборд должен включать следующие элементы:

1. **KPI**: общая сумма собранных премий за месяц, общая сумма выплат, коэффициент убыточности (loss ratio), количество страховых случаев, средняя выплата.
2. **Динамика выплат и премий** — линейный график по месяцам (с двумя линиями: выплаты и премии) для оценки тренда.
3. **Убыточность по типам полисов** — столбчатая диаграмма (loss ratio по каждому типу).
4. **Распределение страховых случаев по статусам** — круговая диаграмма (submitted, under review, approved, paid, rejected).
5. **Топ-5 клиентов по сумме выплат** — таблица.
6. **Карта регионов** (если есть данные) — тепловая карта с убыточностью по регионам.
7. **Фильтры**: по дате, по типу полиса, по региону, по статусу заявки.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по типу полиса открывает детальный отчёт по этому типу.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 25. Агрохолдинг (поля, урожай)

Описание кейса

Агрохолдинг занимается выращиванием сельскохозяйственных культур на нескольких полях. Ведётся учёт полей (их местоположение, площадь, тип почвы), выращиваемых культур (название, сорт, средняя урожайность, оптимальные условия), а также данных об урожаях за каждый сезон (год, фактическая урожайность с гектара, общий сбор, затраты на выращивание, погодные условия, применение удобрений и т.п.). Компания хочет анализировать урожайность, выявлять факторы, влияющие на урожайность (почва, погода, удобрения), прогнозировать будущие урожаи и оптимизировать использование полей. Бизнес-цели:

- Мониторинг урожайности по полям и культурам в разрезе лет.
- Выявление полей с наилучшей и наихудшей урожайностью для оптимизации севооборота.
- Анализ влияния погодных условий (температура, осадки) на урожайность.
- Прогнозирование урожайности на основе исторических данных и метеопрогнозов.
- Оценка экономической эффективности выращивания различных культур.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для агрономов и руководства холдинга.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных для учёта полей, культур, урожаев и сопутствующих данных. Модель должна включать как минимум следующие сущности:

- **fields** — поля (название, площадь, местоположение, тип почвы, координаты для карты);
- **crops** — сельскохозяйственные культуры (название, сорт, средняя урожайность, рекомендуемые условия);
- **harvests** — урожаи (поле, культура, год, фактическая урожайность с гектара, общий сбор, затраты, дата сбора);
- **weather** — погодные данные (дата, поле (или регион), температура, осадки, влажность) — опционально, можно хранить отдельно для анализа.

Определите атрибуты, типы данных, ограничения целостности, связи.

Примерная структура таблиц (можете расширить):

1. **fields**: `field_id` (SERIAL PRIMARY KEY), `field_name` (VARCHAR(100) NOT NULL), `area_ha` (NUMERIC(10,2) NOT NULL CHECK >0), `location` (VARCHAR(200)), `soil_type` (VARCHAR(50)), `coordinates` (TEXT) — например, геометрия для карт.
2. **crops**: `crop_id` (SERIAL PRIMARY KEY), `crop_name` (VARCHAR(100) NOT NULL), `variety` (VARCHAR(50)), `avg_yield` (NUMERIC(10,2)) — средняя урожайность в ц/га, `optimal_temp` (NUMERIC(5,2)), `optimal_rainfall` (NUMERIC(10,2)) — мм осадков.

3. `harvests`: `harvest_id` (SERIAL PRIMARY KEY), `field_id` (INTEGER REFERENCES fields), `crop_id` (INTEGER REFERENCES crops), `year` (INTEGER NOT NULL), `actual_yield` (NUMERIC(10,2) NOT NULL) — урожайность в ц/га, `total_harvest` (NUMERIC(12,2)) — общий сбор, `cost` (NUMERIC(12,2)) — затраты на выращивание, `harvest_date` (DATE), `fertilizer_used` (VARCHAR(100)), `notes` (TEXT).
4. `weather_data` (опционально): `weather_id` (SERIAL PRIMARY KEY), `field_id` (INTEGER REFERENCES fields), `date` (DATE), `temperature` (NUMERIC(5,2)), `rainfall` (NUMERIC(10,2)), `humidity` (NUMERIC(5,2)).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц с комментариями и ограничениями (например, проверка, что урожайность > 0 , год > 2000). Добавьте также пример вставки данных для тестирования (по желанию).

Задание 2. CRUD-операции

Разработайте SQL-скрипты для следующих операций:

1. Добавление нового поля.
2. Добавление новой культуры.
3. Запись урожая для поля и культуры за конкретный год.
4. Обновление данных об урожае (например, корректировка урожайности).
5. Удаление записи об урожае (если ошибка).
6. Выборка всех урожаев за определённый год с указанием поля, культуры, урожайности.
7. Расчёт средней урожайности по культурам за последние 5 лет.
8. Выборка полей с урожайностью ниже средней по культуре для принятия решений.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Анализ урожайности», которая обновляется ежегодно после сбора урожая. Целевая витрина должна содержать по каждому полю, культуре и году следующие показатели:

- `field_id`, `field_name`, `crop_id`, `crop_name`;
- `year`;
- `actual_yield` — фактическая урожайность;
- `avg_yield_reference` — средняя урожайность по культуре (из справочника);
- `yield_deviation` — отклонение фактической урожайности от средней (в процентах);
- `total_harvest` — общий сбор;
- `cost_per_ha` — затраты на гектар;
- `profitability` — рентабельность (если есть цена реализации, можно добавить).

Также можно добавить показатели по погоде, если есть данные.

Опишите ETL-процесс поэтапно: из каких таблиц извлекаются данные, какие преобразования выполняются (расчёт отклонений, агрегация, присоединение справочников), стратегия загрузки (полная перезагрузка или инкрементальная). Приведите псевдокод или фрагменты SQL.

Задание 4. Дашборд

Разработайте макет дашборда для агронома, который позволит отслеживать урожайность и анализировать факторы. Дашборд должен включать:

- KPI: средняя урожайность по всем полям за текущий год, общий сбор, средние затраты, количество культур.
- Карта полей с цветовой индикацией урожайности (тепловая карта на основе координат).
- Линейный график динамики урожайности по годам для выбранной культуры или поля.
- Столбчатая диаграмма сравнения урожайности по полям за текущий год.
- Таблица отклонений урожайности от средней (для выявления проблемных полей).
- Диаграмма распределения затрат по культурам.
- Фильтры: по году, по культуре, по полю.

Опишите визуализации и источники данных. Предложите интерактивные взаимодействия.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда.

Вариант 26. Муниципалитет (население, платежи)

Описание кейса

Муниципалитет ведёт учёт жителей, предоставляемых услуг (коммунальные, налоги, сборы) и платежей от населения. Каждый житель (физическое лицо) может иметь льготы. Услуги имеют тарифы и единицы измерения (например, руб./кв.м, руб./Гкал). Ежемесячно или ежеквартально производятся начисления (accruals) для каждого жителя по каждой услуге, исходя из потребления (например, площадь квартиры, показания счётчиков) или фиксированной ставки. Жители вносят платежи, которые могут покрывать начисления полностью или частично. Муниципалитет хочет анализировать:

- Общую задолженность населения по видам услуг и в разрезе районов/адресов.
- Собираемость платежей (отношение оплаченных сумм к начисленным) за периоды.
- Выявление злостных неплательщиков и прогнозирование поступлений.
- Эффективность работы служб по взысканию задолженности.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководителей муниципалитета и финансового отдела.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных для учёта жителей, услуг, начислений и платежей. Модель должна включать как минимум следующие сущности:

- **residents** — жители (ФИО, адрес, телефон, ИНН, категория льгот, дата регистрации);
- **services** — услуги (название, единица измерения, тариф, описание);
- **accruals** — начисления (житель, услуга, период, сумма начисления, дата начисления);
- **payments** — платежи (начисление, сумма оплаты, дата оплаты, статус).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить, например, добавить справочник адресов или районов):

1. **residents**: `resident_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `address` (TEXT NOT NULL), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `inn` (VARCHAR(12) UNIQUE) — ИНН, `benefit_category` (VARCHAR(30) CHECK benefit_category IN ('none', 'veteran', 'disabled', 'registration_date' (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('active', 'moved', 'deceased'))).
2. **services**: `service_id` (SERIAL PRIMARY KEY), `service_name` (VARCHAR(100) NOT NULL), `unit` (VARCHAR(20) CHECK unit IN ('кв.м', 'Гкал', 'чел', 'м³', 'руб')), `rate` (NUMERIC(10,2) NOT NULL CHECK >0), `description` (TEXT).

3. **accruals:** `accrual_id` (SERIAL PRIMARY KEY), `resident_id` (INTEGER REFERENCES residents ON DELETE CASCADE), `service_id` (INTEGER REFERENCES services), `period` (VARCHAR(7) NOT NULL) — например, '2025-01', `amount` (NUMERIC(12,2) NOT NULL CHECK >0), `accrual_date` (DATE DEFAULT CURRENT_DATE).
4. **payments:** `payment_id` (SERIAL PRIMARY KEY), `accrual_id` (INTEGER REFERENCES accruals ON DELETE CASCADE), `payment_amount` (NUMERIC(12,2) NOT NULL CHECK >0), `payment_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('pending', 'completed', 'failed')).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что сумма начисления и платежа положительная, периоды имеют формат YYYY-MM, статусы из допустимых списков.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового жителя.**
2. **Добавление новой услуги.**
3. **Создание начисления** для жителя по услуге за период (вставка в `accruals`).
4. **Внесение платежа** (вставка в `payments`) с привязкой к конкретному начислению.
5. **Обновление статуса платежа** (например, с 'pending' на 'completed' при подтверждении).
6. **Удаление начисления** (если ошибка, но только если нет платежей).
7. **Выборка всех жителей с задолженностью** (сумма начислений минус сумма платежей > 0) с указанием общей суммы долга.
8. **Расчёт собираемости платежей за месяц** — отношение суммы оплат к сумме начислений за тот же период.
9. **Выборка топ-10 должников** по сумме задолженности.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Задолженность и собираемость», которая обновляется ежедневно (или ежемесячно). Целевая витрина должна содержать следующие показатели по каждому жителю (или по районам/адресам) за месяц (или на текущую дату):

- `resident_id`, `full_name`, `address`, `benefit_category`;
- `total_accrued` — общая сумма начислений за период (или накопленная);
- `total_paid` — общая сумма оплат;

- `debt` — задолженность ($\text{total_accrued} - \text{total_paid}$);
- `payment_ratio` — процент оплаты ($\text{total_paid} / \text{total_accrued} * 100$);
- `last_payment_date` — дата последнего платежа.

Также можно добавить агрегацию по услугам и периодам для анализа динамики.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`residents`, `accruals`, `payments`). Укажите, загружаются ли данные за весь период или инкрементально (например, только новые начисления и платежи).
2. **Transform:** опишите преобразования:
 - объединение начислений и платежей по жителям и периодам;
 - вычисление сумм начислений и оплат для каждого жителя (группировка);
 - расчёт задолженности и процента оплаты;
 - определение даты последнего платежа.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `resident_id`). Предложите механизм контроля качества (проверка, что `debt` ≥ 0 , `payment_ratio` между 0 и 100).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя финансового отдела муниципалитета, который позволит отслеживать задолженность и собираемость платежей. Дашборд должен включать следующие элементы:

1. **KPI:** общая сумма начислений за месяц, общая сумма платежей, общая задолженность, процент собираемости, количество должников (с долгом > 0).
2. **Динамика задолженности** — линейный график (сумма задолженности по месяцам за последние 12 месяцев).
3. **Собираемость по услугам** — столбчатая диаграмма (процент оплаты по каждому виду услуг).
4. **Задолженность по районам** (если адрес содержит район) — географическая карта или тепловая карта с цветовой индикацией.
5. **Топ-10 должников** — таблица с ФИО, адресом, суммой долга и датой последнего платежа.
6. **Количество жителей по категориям льгот** — круговая диаграмма.
7. **Фильтры:** по периоду, по услуге, по району, по категории льгот.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по должнику открывает детальную историю его начислений и платежей.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 27. Управление проектами (задачи)

Описание кейса

Компания ведёт управление проектами с использованием системы учёта задач. Каждый проект имеет название, описание, даты начала и планового окончания, бюджет, ответственного менеджера. Проект разбивается на задачи, которые могут иметь подзадачи. Каждая задача характеризуется названием, описанием, статусом (новая, в работе, на проверке, завершена, отменена), датами начала и завершения (плановыми и фактическими), оценкой трудоёмкости (человеко-часы), приоритетом, а также ответственным исполнителем (сотрудником). Сотрудники имеют квалификацию, должность, отдел, загрузку. Компания хочет анализировать:

- Ход выполнения проектов (отклонения от плановых сроков, бюджета).
- Загрузку сотрудников и распределение задач.
- Оценку точности планирования (сравнение плановых и фактических сроков/трудоёмкости).
- Выявление проблемных задач и проектов.
- Прогнозирование сроков завершения проектов на основе текущего статуса.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд с диаграммой Ганта для руководителей проектов и менеджеров.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных для учёта проектов, задач и сотрудников. Модель должна включать как минимум следующие сущности:

- **projects** — проекты (название, описание, дата начала, плановая дата окончания, фактическая дата окончания, бюджет, статус, менеджер проекта);
- **employees** — сотрудники (ФИО, должность, отдел, квалификация, ставка (часовая), контакты);
- **tasks** — задачи (название, описание, статус, приоритет, плановые даты начала и окончания, фактические даты, оценка трудоёмкости в часах, фактически затраченное время, проект, ответственный сотрудник, родительская задача для иерархии);
- **task_dependencies** — зависимости между задачами (предшествующая задача, последующая задача, тип зависимости) — опционально, для построения критического пути.

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **projects**: `project_id` (SERIAL PRIMARY KEY), `project_name` (VARCHAR(100) NOT NULL), `description` (TEXT), `start_date` (DATE NOT NULL), `planned_end_date` (DATE NOT NULL, CHECK `planned_end_date >= start_date`), `actual_end_date` (DATE), `budget` (NUMERIC(12,2) NOT NULL CHECK `>0`), `status` (VARCHAR(20) CHECK `status IN ('planning', 'active', 'paused', 'completed', 'cancelled')`), `manager_id` (INTEGER REFERENCES `employees`) — ответственный менеджер.

2. **employees:** `employee_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `position` (VARCHAR(100)), `department` (VARCHAR(100)), `hourly_rate` (NUMERIC(10,2) CHECK >0), `phone`, `email` (UNIQUE), `hire_date` (DATE).
3. **tasks:** `task_id` (SERIAL PRIMARY KEY), `project_id` (INTEGER REFERENCES projects ON DELETE CASCADE), `parent_task_id` (INTEGER REFERENCES tasks) — для иерархии, `task_name` (VARCHAR(200) NOT NULL), `description` (TEXT), `status` (VARCHAR(20) CHECK status IN ('new', 'in_progress', 'review', 'completed', 'cancelled')), `priority` (VARCHAR(20) CHECK priority IN ('low', 'medium', 'high', 'critical')), `planned_start` (DATE), `planned_end` (DATE, CHECK planned_end >= planned_start), `actual_start` (DATE), `actual_end` (DATE, CHECK actual_end >= actual_start), `estimated_hours` (NUMERIC(8,2) CHECK >0), `actual_hours` (NUMERIC(8,2) DEFAULT 0), `assigned_employee_id` (INTEGER REFERENCES employees).
4. **task_dependencies** (опционально): `dependency_id` (SERIAL PRIMARY KEY), `task_id` (INTEGER REFERENCES tasks ON DELETE CASCADE), `depends_on_task_id` (INTEGER REFERENCES tasks ON DELETE CASCADE), `dependency_type` (VARCHAR(20) CHECK dependency_type IN ('finish_to_start', 'start_to_start', 'finish_to_finish')), `lag` (INTEGER DEFAULT 0) — задержка в днях.

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что статусы из допустимых списков, даты корректны, оценки трудоёмкости положительные.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Создание нового проекта.**
2. **Добавление нового сотрудника.**
3. **Добавление задачи в проект** с указанием родительской задачи (если есть), ответственного сотрудника, плановых дат и оценки трудоёмкости.
4. **Обновление статуса задачи** (например, на 'in_progress', 'completed').
5. **Фиксация фактических затрат времени** по задаче (обновление actual_hours) и фактических дат (actual_start/end).
6. **Выборка всех задач, находящихся в работе, по проекту.**
7. **Расчёт процента выполнения проекта** на основе статуса задач (например, доля завершённых задач).
8. **Выборка сотрудников с наибольшей загрузкой** (сумма estimated_hours или actual_hours по незавершённым задачам).

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Сводка по проектам и задачам», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому проекту (и по задачам) на текущую дату:

- `project_id`, `project_name`, `manager`;
- `start_date`, `planned_end_date`, `actual_end_date`;
- `total_tasks` — общее количество задач;
- `completed_tasks` — количество завершённых задач;
- `completion_percent` — процент завершения ($\text{completed_tasks} / \text{total_tasks} * 100$);
- `planned_hours` — суммарная плановая трудоёмкость;
- `actual_hours` — суммарные фактические затраты;
- `deviation_hours` — отклонение ($\text{actual} - \text{planned}$);
- `days_to_deadline` — дней до планового окончания (если проект ещё не завершён);
- `status` — статус проекта.

Также можно добавить агрегацию по сотрудникам (загрузка, количество задач) и по задачам с отклонениями для выявления проблем.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`projects`, `tasks`, `employees`). Укажите, загружаются ли данные за весь период или инкрементально (изменения за день).
2. **Transform:** опишите преобразования:
 - подсчёт количества задач и завершённых задач по проектам;
 - суммирование плановых и фактических часов;
 - вычисление процента выполнения;
 - расчёт дней до дедлайна (если проект активен);
 - объединение с данными о менеджере.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `project_id`). Предложите механизм контроля качества (проверка, что `completion_percent` между 0 и 100, отклонение может быть отрицательным).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя проектного офиса, который позволит отслеживать ход проектов и загрузку сотрудников. Дашборд должен включать следующие элементы:

1. **KPI**: общее количество активных проектов, процент завершённых задач в целом, отклонение по срокам (среднее), перегрузка сотрудников.
2. **Диаграмма Ганта** — визуализация задач по проектам с плановыми и фактическими датами, цветовое кодирование статусов.
3. **Динамика выполнения проектов** — линейный график (процент завершения по дням для выбранного проекта).
4. **Загрузка сотрудников** — столбчатая диаграмма (количество назначенных задач или сумма часов по каждому сотруднику).
5. **Отклонения от плана** — гистограмма (отклонение по часам или дням для каждого проекта).
6. **Список просроченных задач** — таблица с задачами, где `planned_end` < текущая дата, а статус не завершён.
7. **Фильтры**: по проекту, по сотруднику, по статусу задачи, по дате.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по задаче на диаграмме Ганта открывает детали задачи.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 28. Фриланс-платформа (заказы, исполнители)

Описание кейса

Фриланс-платформа соединяет заказчиков (клиентов) и исполнителей (фрилансеров). Заказчики размещают заказы (проекты) с описанием, бюджетом, сроками, категорией. Фрилансеры могут откликаться на заказы, предлагая свои услуги. Заказчик выбирает исполнителя, после чего заказ переходит в статус «в работе». По завершении работы заказчик может оставить отзыв и оценку (рейтинг) исполнителю. Платформа зарабатывает на комиссии с каждого заказа. Бизнес-цели:

- Анализ доходности платформы (выручка от комиссий).
- Мониторинг активности фрилансеров и заказчиков.
- Оценка рейтинга фрилансеров на основе отзывов и завершённых проектов.
- Выявление наиболее востребованных категорий услуг.
- Прогнозирование загрузки платформы и сезонности.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководителей платформы.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта заказчиков, фрилансеров, заказов, откликов и отзывов. Модель должна включать как минимум следующие сущности:

- **customers** — заказчики (ФИО, контакты, дата регистрации, статус);
- **freelancers** — фрилансеры (ФИО, контакты, навыки, рейтинг, статус);
- **orders** — заказы (заказчик, название, описание, бюджет, категория, срок, статус);
- **bids** — отклики (фрилансер, заказ, предложенная цена, комментарий, дата);
- **reviews** — отзывы (заказ, оценка (1–5), текст, дата).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **customers**: `customer_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `registration_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('active', 'inactive')).
2. **freelancers**: `freelancer_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `phone`, `email` (UNIQUE), `skills` (TEXT), `rating` (NUMERIC(3,2) DEFAULT 0) — средняя оценка, `registration_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('active', 'inactive', 'blocked')), `hourly_rate` (NUMERIC(10,2)).

3. **orders:** `order_id` (SERIAL PRIMARY KEY), `customer_id` (INTEGER REFERENCES customers), `title` (VARCHAR(200) NOT NULL), `description` (TEXT), `budget` (NUMERIC(12,2) CHECK >0), `category` (VARCHAR(100)), `deadline` (DATE), `status` (VARCHAR(20) CHECK status IN ('open', 'in_progress', 'completed', 'cancelled')), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `platform_fee` (NUMERIC(5,2) DEFAULT 10.0)
— комиссия платформы в процентах.
4. **bids:** `bid_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders ON DELETE CASCADE), `freelancer_id` (INTEGER REFERENCES freelancers), `bid_amount` (NUMERIC(12,2) NOT NULL CHECK >0), `bid_comment` (TEXT), `bid_date` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `status` (VARCHAR(20) CHECK status IN ('pending', 'accepted', 'rejected')).
5. **reviews:** `review_id` (SERIAL PRIMARY KEY), `order_id` (INTEGER REFERENCES orders ON DELETE CASCADE), `freelancer_id` (INTEGER REFERENCES freelancers), `rating` (INTEGER CHECK rating BETWEEN 1 AND 5), `comment` (TEXT), `review_date` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что бюджет и ставки положительные, статусы из допустимых списков, оценка в диапазоне 1–5.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Регистрация нового заказчика и нового фрилансера.**
2. **Создание заказа** заказчиком.
3. **Отклик на заказ** (вставка в bids) с указанием предложенной цены.
4. **Выбор исполнителя** (обновление статуса заказа на 'in_progress' и обновление статуса выбранного bid на 'accepted', остальные — 'rejected').
5. **Завершение заказа** (обновление статуса на 'completed').
6. **Оставление отзыва** (вставка в reviews) с расчётом обновлённого рейтинга фрилансера (можно сделать через триггер или вручную).
7. **Выборка всех открытых заказов** с указанием категории, бюджета и количества откликов.
8. **Топ-5 фрилансеров по рейтингу и по количеству завершённых заказов.**

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Доходность платформы и эффективность фрилансеров», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому фрилансеру и по дням (или месяцам):

- `freelancer_id`, `freelancer_name`;

- `date` (отчётная дата);
- `total_orders_completed` — количество завершённых заказов;
- `total_revenue` — общая сумма заработанных средств (сумма `budget` заказов);
- `avg_rating` — средний рейтинг;
- `total_bids` — количество откликов;
- `conversion_rate` — доля откликов, перешедших в заказы ($\text{completed_orders} / \text{total_bids} * 100$);
- `platform_revenue` — доход платформы от комиссий с заказов ($\text{total_revenue} * \text{platform_fee} / 100$);
- `avg_order_budget` — средний бюджет заказов.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`orders`, `bids`, `reviews`, `freelancers`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение заказов с откликами, отзывами и фрилансерами;
 - фильтрация по статусу `'completed'` и датам;
 - расчёт комиссии платформы;
 - агрегация по фрилансерам и датам;
 - вычисление средних значений и конверсии.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `freelancer_id`, `date`). Предложите механизм контроля качества (проверка, что `conversion_rate` ≤ 100 , `avg_rating` между 1 и 5).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя фриланс-платформы, который позволит отслеживать доходы и эффективность исполнителей. Дашборд должен включать следующие элементы:

1. **KPI:** общая выручка платформы за период, количество завершённых заказов, количество активных фрилансеров, средний рейтинг, средняя конверсия откликов.
2. **Динамика выручки платформы** — линейный график по дням/месяцам.
3. **Рейтинг фрилансеров** — столбчатая диаграмма (топ-10 по рейтингу или по доходу).

4. **Распределение заказов по категориям** — круговая диаграмма.
5. **Конверсия фрилансеров** — диаграмма рассеяния (количество откликов vs количество завершённых заказов).
6. **Список заказов с высоким бюджетом** — таблица для анализа.
7. **Фильтры:** по дате, по категории, по статусу заказа.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по фрилансеру открывает детальный отчёт по его заказам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 29. Стоматологическая клиника (пациенты, услуги)

Описание кейса

Стоматологическая клиника предоставляет широкий спектр услуг: терапия, хирургия, ортопедия, ортодонтия, гигиена, имплантация, отбеливание и другие. Клиника ведёт учёт пациентов, врачей-стоматологов (специалистов), услуг, записей на приём и оплат. Каждый приём связан с пациентом, врачом и одной или несколькими услугами (за один визит может быть выполнено несколько процедур). Для каждой услуги определена стандартная цена и длительность. Врачи специализируются на определённых направлениях. Клиника хочет анализировать:

- Доходность по видам услуг, по врачам, по периодам.
- Загруженность врачей (количество приёмов, длительность, количество пациентов).
- Эффективность работы клиники (конверсия записей в приёмы, средний чек, повторные визиты).
- Популярность услуг и сезонные тренды.
- Планирование графика работы врачей на основе спроса.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководителя клиники и администраторов.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта пациентов, врачей, услуг, записей на приём и платежей. Модель должна включать как минимум следующие сущности:

- **patients** — пациенты (ФИО, дата рождения, контакты, дата регистрации);
- **doctors** — врачи (ФИО, специализация, контакты, дата найма, статус);
- **services** — услуги (название, описание, цена, стандартная длительность в минутах, категория);
- **appointments** — записи на приём (пациент, врач, дата и время, фактическая длительность, сумма, статус);
- **appointment_services** — список услуг, выполненных в рамках одного приёма (связь многие-ко-многим с указанием количества и цены);
- **payments** — оплаты (приём, сумма, дата, способ, статус).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **patients**: `patient_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `date_of_birth` (DATE), `phone` (VARCHAR(20)), `email` (VARCHAR(100) UNIQUE), `address` (TEXT), `registration_date` (DATE DEFAULT CURRENT_DATE), `status` (VARCHAR(20) CHECK status IN ('active','inactive')).

2. **doctors:** `doctor_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `specialization` (VARCHAR(100) NOT NULL), `phone`, `email` (UNIQUE), `hire_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('active','on_leave','terminated')).
3. **services:** `service_id` (SERIAL PRIMARY KEY), `service_name` (VARCHAR(100) NOT NULL), `category` (VARCHAR(50) CHECK category IN ('therapy','surgery','orthopedics','orthopedics')), `description` (TEXT), `price` (NUMERIC(10,2) NOT NULL CHECK >0), `duration_minutes` (INTEGER NOT NULL CHECK >0).
4. **appointments:** `appointment_id` (SERIAL PRIMARY KEY), `patient_id` (INTEGER REFERENCES patients ON DELETE CASCADE), `doctor_id` (INTEGER REFERENCES doctors), `appointment_datetime` (TIMESTAMP NOT NULL), `actual_duration_minutes` (INTEGER), `total_amount` (NUMERIC(10,2) DEFAULT 0), `status` (VARCHAR(20) CHECK status IN ('scheduled','in_progress','completed','cancelled','no_show')), `created_at` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP).
5. **appointment_services:** `appointment_service_id` (SERIAL PRIMARY KEY), `appointment_id` (INTEGER REFERENCES appointments ON DELETE CASCADE), `service_id` (INTEGER REFERENCES services), `quantity` (INTEGER DEFAULT 1 CHECK >0), `unit_price` (NUMERIC(10,2) NOT NULL) — цена на момент выполнения.
6. **payments:** `payment_id` (SERIAL PRIMARY KEY), `appointment_id` (INTEGER REFERENCES appointments), `payment_amount` (NUMERIC(10,2) NOT NULL CHECK >0), `payment_date` (TIMESTAMP DEFAULT CURRENT_TIMESTAMP), `payment_method` (VARCHAR(30) CHECK payment_method IN ('cash','card','online','insurance')), `status` (VARCHAR(20) CHECK status IN ('pending','completed','failed')).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что цена и длительность положительные, статусы из допустимых списков.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового пациента и нового врача.**
2. **Добавление новой услуги** в справочник.
3. **Создание записи на приём** (вставка в appointments) с указанием пациента, врача, даты и времени.
4. **Добавление услуг в приём** (вставка в appointment_services) с актуальной ценой.
5. **Обновление статуса приёма** (например, на 'completed' или 'cancelled') и обновление фактической длительности.
6. **Внесение оплаты** за приём.
7. **Выборка всех приёмов на сегодня** с указанием пациента, врача, услуг и статуса.
8. **Расчёт выручки за день** (сумма оплат за завершённые приёмы).

9. Топ-3 самых популярных услуги (по количеству выполненных услуг за месяц).

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Доходы и загрузка врачей», которая обновляется ежедневно. Целевая витрина должна содержать следующие показатели по каждому врачу и по каждому дню (или неделе):

- `doctor_id`, `doctor_name`, `specialization`;
- `date` (отчётная дата);
- `total_appointments` — количество приёмов (завершённых);
- `total_patients` — количество уникальных пациентов;
- `total_services` — общее количество оказанных услуг (сумма `quantity`);
- `total_revenue` — общая выручка от приёмов;
- `avg_duration_minutes` — средняя фактическая длительность приёма;
- `occupancy_rate` — коэффициент загрузки (можно рассчитать как отношение суммарной длительности к доступному рабочему времени, например, 8 часов);
- `no_show_count` — количество неявок (статус `'no_show'`).

Также можно добавить агрегацию по услугам для анализа их доходности.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`appointments`, `appointment_services`, `services`, `doctors`, `patients`). Укажите, загружаются ли данные за весь период или инкрементально (за прошедший день).
2. **Transform:** опишите преобразования:
 - соединение приёмов с услугами, врачами;
 - фильтрация по статусу `'completed'`;
 - расчёт количества услуг, суммы выручки, средней длительности;
 - вычисление загрузки (например, сумма длительности всех приёмов за день / $(8*60) * 100$);
 - подсчёт неявок.
3. **Load:** определите стратегию загрузки в витрину (полная перезагрузка или инкрементальная с обновлением по ключу `doctor_id`, `date`). Предложите механизм контроля качества (проверка, что `occupancy_rate` между 0 и 100, `total_revenue` ≥ 0).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с `pandas`, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для руководителя клиники, который позволит отслеживать ключевые показатели. Дашборд должен включать следующие элементы:

1. **KPI:** общая выручка за день/неделю/месяц, количество приёмов, количество пациентов, средний чек, процент неявок, загруженность врачей.
2. **Динамика выручки** — линейный график по дням за последние 30 дней.
3. **Доходность по услугам** — столбчатая диаграмма (общая выручка по каждому виду услуг).
4. **Загруженность врачей** — столбчатая диаграмма (количество приёмов или суммарная длительность по врачам) с фильтром по специализации.
5. **Распределение пациентов по возрасту** — гистограмма (для анализа демографии).
6. **Список предстоящих записей** на завтра — таблица.
7. **Фильтры:** по дате, по врачу, по услуге, по статусу приёма.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по врачу открывает детальный отчёт по его приёмам.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.

Вариант 30. Университет (студенты, курсы, оценки)

Описание кейса

Университет ведёт учёт студентов, курсов, преподавателей и успеваемости. Каждый студент обучается по определённой специальности (направлению) и зачислен на конкретный курс (семестр). Университет предлагает множество курсов (дисциплин), каждый из которых ведётся одним или несколькими преподавателями. Студенты записываются на курсы в каждом семестре, и по окончании семестра им выставляются оценки (по шкале, например, от 2 до 5, или буквенные A, B, C, D, F, или числовые баллы). Преподаватели имеют квалификацию, стаж, учёную степень. Бизнес-цели:

- Мониторинг успеваемости студентов (средний балл, процент отчисленных, распределение оценок).
- Выявление слабоуспевающих студентов для своевременной поддержки.
- Анализ эффективности преподавателей (средняя оценка по курсам, количество студентов, отзывы).
- Оценка популярности курсов (количество записей, средний балл).
- Планирование учебной нагрузки преподавателей, анализ загруженности.

Для поддержки принятия решений необходимо создать хранилище данных, ETL-процессы и дашборд для руководства университета, деканатов и учебной части.

Задание 1. ER-диаграмма и SQL-скрипт создания БД

Спроектируйте логическую модель базы данных (ER-диаграмму) для учёта студентов, курсов, преподавателей, специальностей, семестров и оценок. Модель должна включать как минимум следующие сущности:

- **students** — студенты (ФИО, дата рождения, контакты, дата поступления, специальность);
- **specialties** — специальности (название, код, описание);
- **courses** — курсы (название, описание, кредитные часы, семестр, специальность (если курс закреплён за специальностью));
- **instructors** — преподаватели (ФИО, учёная степень, должность, контакты);
- **course_offerings** — предложения курсов в конкретном семестре (связь курса, преподавателя, семестра, года);
- **enrollments** — записи студентов на курс (студент, курс (конкретное предложение), оценка, дата выставления).

Определите атрибуты для каждой сущности, их типы данных и ограничения целостности (первичные и внешние ключи, NOT NULL, CHECK, DEFAULT). Предложите связи между таблицами и обоснуйте их.

Примерная структура таблиц (можете расширить):

1. **specialties**: **specialty_id** (SERIAL PRIMARY KEY), **code** (VARCHAR(20) UNIQUE NOT NULL), **name** (VARCHAR(100) NOT NULL), **description** (TEXT).

2. **students:** `student_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `date_of_birth` (DATE), `phone`, `email` (UNIQUE), `address` (TEXT), `enrollment_year` (INTEGER NOT NULL), `specialty_id` (INTEGER REFERENCES specialties), `status` (VARCHAR(20) CHECK status IN ('active', 'graduated', 'academic_leave', 'expelled')).
3. **instructors:** `instructor_id` (SERIAL PRIMARY KEY), `first_name` (VARCHAR(50) NOT NULL), `last_name` (VARCHAR(50) NOT NULL), `title` (VARCHAR(50) CHECK title IN ('assistant', 'lecturer', 'senior_lecturer', 'associate_professor', 'professor')), `phone`, `email` (UNIQUE), `hire_date` (DATE).
4. **courses:** `course_id` (SERIAL PRIMARY KEY), `course_code` (VARCHAR(20) UNIQUE NOT NULL), `name` (VARCHAR(100) NOT NULL), `credits` (INTEGER CHECK >0), `description` (TEXT), `specialty_id` (INTEGER REFERENCES specialties) — если курс привязан к специальности.
5. **semesters:** `semester_id` (SERIAL PRIMARY KEY), `semester_name` (VARCHAR(20) NOT NULL), `academic_year` (VARCHAR(20)), `start_date` (DATE), `end_date` (DATE).
6. **course_offerings:** `offering_id` (SERIAL PRIMARY KEY), `course_id` (INTEGER REFERENCES courses), `instructor_id` (INTEGER REFERENCES instructors), `semester_id` (INTEGER REFERENCES semesters), `max_students` (INTEGER DEFAULT 30), `room` (VARCHAR(50)).
7. **enrollments:** `enrollment_id` (SERIAL PRIMARY KEY), `student_id` (INTEGER REFERENCES students ON DELETE CASCADE), `offering_id` (INTEGER REFERENCES course_offerings ON DELETE CASCADE), `grade` (NUMERIC(5,2) CHECK (grade BETWEEN 0 AND 100) OR (grade IN (2,3,4,5) — можно задать шкалу), `grade_letter` (CHAR(2) CHECK grade_letter IN ('A', 'B', 'C', 'D', 'F')), `grade_date` (DATE), `status` (VARCHAR(20) CHECK status IN ('registered', 'passed', 'failed', 'incomplete')).

Напишите SQL-скрипт на PostgreSQL для создания всех таблиц в правильном порядке с учётом внешних ключей. Добавьте комментарии к таблицам и столбцам. Предусмотрите ограничения, например, что оценки в допустимом диапазоне, даты корректны, статусы из допустимых списков. Добавьте уникальное ограничение на (`student_id`, `offering_id`) для предотвращения дублирования записей.

Задание 2. CRUD-операции

Разработайте SQL-скрипты для выполнения следующих операций:

1. **Добавление нового студента** с указанием специальности.
2. **Добавление нового курса и нового преподавателя.**
3. **Создание предложения курса** (course offering) на семестр.
4. **Запись студента на курс** (вставка в enrollments) с начальным статусом 'registered' (без оценки).
5. **Выставление оценки** студенту по курсу (обновление поля grade, grade_letter, grade_date, статус на 'passed' или 'failed').
6. **Изменение оценки** (если ошибка) и обновление средней.

7. **Выборка всех студентов с задолженностями** (оценка ниже порога, например, < 60 или F) за текущий семестр.
8. **Расчёт среднего балла по группе** (по специальности, по курсу) — запрос с агрегацией.
9. **Топ-5 курсов по успеваемости** (средняя оценка) за последний семестр.

Все запросы должны быть корректными и учитывать ссылочную целостность. При необходимости используйте транзакции.

Задание 3. ETL-процесс

Разработайте ETL-пайплайн для построения витрины данных «Успеваемость и эффективность курсов», которая обновляется после каждой сессии (или еженедельно). Целевая витрина должна содержать следующие показатели по каждому курсу (или специальности) за семестр:

- `course_id`, `course_name`, `instructor_id`, `instructor_name`;
- `semester_id`, `semester_name`;
- `total_students` — общее количество студентов, записанных на курс;
- `passed_students` — количество студентов, получивших положительную оценку (например, ≥ 60 или D и выше);
- `failed_students` — количество не сдавших;
- `avg_grade` — средняя оценка (в баллах);
- `grade_distribution` — распределение оценок по категориям (A, B, C, D, F) в виде отдельных столбцов или JSON;
- `success_rate` — процент успеваемости ($\text{passed} / \text{total} * 100$);
- `stddev_grade` — стандартное отклонение оценок (для анализа разброса).

Также можно добавить агрегацию по преподавателям для оценки их эффективности.

Опишите ETL-процесс поэтапно:

1. **Extract:** из каких таблиц извлекаются данные (`enrollments`, `students`, `course_offerings`, `courses`, `instructors`, `semesters`). Укажите, загружаются ли данные за весь период или инкрементально (только изменения за семестр).
2. **Transform:** опишите преобразования:
 - соединение записей с курсами, преподавателями, студентами, семестрами;
 - фильтрация по статусу 'passed' или 'failed' (можно использовать `grade`, если она выставлена);
 - вычисление средней оценки, стандартного отклонения;
 - подсчёт количества студентов и распределения по оценкам (например, с помощью CASE или фильтров);
 - расчёт успеваемости.

3. **Load**: определите стратегию загрузки в витрину (полная перезагрузка после каждого семестра или инкрементальная). Предложите механизм контроля качества (проверка, что `success_rate` между 0 и 100, `avg_grade` в допустимом диапазоне).

Приведите псевдокод или SQL-фрагменты основных операций. Опишите, какие инструменты вы бы использовали (например, хранимые процедуры в PostgreSQL, скрипты на Python с pandas, или внешние ETL-средства).

Задание 4. Дашборд

Разработайте макет дашборда для учебного отдела и деканата, который позволит отслеживать успеваемость и эффективность образовательного процесса. Дашборд должен включать следующие элементы:

1. **KPI**: общее количество студентов, средний балл по университету, процент успеваемости, количество должников, количество отчисленных.
2. **Распределение оценок** — гистограмма или столбчатая диаграмма, показывающая количество студентов по каждой оценке (A, B, C, D, F) или по числовому диапазону.
3. **Динамика среднего балла** по семестрам (линейный график) с разбивкой по специальностям или курсам.
4. **Успеваемость по курсам** — столбчатая диаграмма (средний балл и успеваемость) с фильтром по преподавателю или семестру.
5. **Рейтинг преподавателей** — таблица с показателями (средняя оценка, процент успеваемости, количество студентов).
6. **Список студентов с низкой успеваемостью** (например, средний балл < 60) — таблица для принятия мер.
7. **Фильтры**: по семестру, по специальности, по курсу, по преподавателю.

Опишите, какие визуализации используются и из каких данных они строятся (из витрины или напрямую из базы). Предложите интерактивные взаимодействия, например, клик по курсу открывает детальный отчёт по студентам этого курса.

Оформите описание дашборда в виде текста и приложите эскиз расположения панелей.

По результатам выполнения всех четырёх заданий студент предоставляет отчёт с ER-диаграммой, SQL-скриптами, описанием ETL и макетом дашборда, а также файлы с SQL-кодом для проверки.