**Базы данных**

**Замечания:**
1. Если БД используется только как вспомогательная часть (например, для хранения логов или конфигураций), она не считается полноценной БД-частью проекта.

2. СУБД должна соответствовать цели проекта. Выбор NoSQL-СУБД должен быть обоснован – объяснено, почему для данной задачи выбрана определенная модель.

**Требования разбиты по следующим категориям:**

- Реляционные БД:
  - Транзакционные БД (OLTP)
  - Аналитические БД (OLAP)
- Нереляционные БД (NoSQL)

**Транзакционные БД (OLTP)**

**Минимальные требования (на 5 баллов)**

**Документация**

- Словарь данных: описание слоёв (если есть), таблиц и колонок, типов данных, ключей, а также ожиданий по качеству данных.
- Описано как поддерживается целостность данных и как выполняются транзакции.
- ER-диаграмма – опциональна.
- Логическая схема: визуальное представление должно быть понятным: выделены основные блоки и связи, допускается цветовое выделение для улучшения восприятия.
- DDL-скрипт (SQL) для создания структуры базы данных.

**Проектирование**

- Использована современная реляционная СУБД (PostgreSQL, SQL Server и т.д.).
- Проведено проектирование структуры с нормализацией до третьей нормальной формы (3НФ).
- Схема включает достаточное количество таблиц и разные типы связей.
- Для всех таблиц указаны:

- первичные и внешние ключи,
- ограничения (NOT NULL, UNIQUE, CHECK и т.п.),
- осмысленные типы данных (например, numeric для денег, timestamp для дат).

**Развёртывание**

- Структура базы данных создаётся через SQL-скрипт или миграции, а не вручную.
- Все скрипты и миграции хранятся в системе контроля версий (Git).

**Тестовые данные**

- Тестовые данные – достаточное количество тестовых записей для демонстрации работоспособности приложения и edge cases.
- Приведены скрипты для заполнения справочных таблиц (фиксированные данные).

**Использование**

- Настроены роли и права доступа:
  - минимум: app_read, app_write, admin – или другие, которые имеют больший смысл для вашей темы;
  - приложению запрещено работать под superuser;
  - при добавлении новых объектов необходимо проверять и обновлять права пользователей.
- Паролиее хранятся только в зашифрованном виде, но не используются устаревшие механизмы шифрования (MD5, SHA-1 и т.п.).
- Целостность данных обеспечивается с помощью ограничений и транзакций.

**Ограничения**

- Нельзя хранить пароли в открытом виде.
- Нельзя использовать CSV/Excel-файлы как единственное хранилище данных без СУБД.
- Нельзя хранить неструктурированные документы (JSON) в реляционной базе без продуманной модели.

    - Исключение возможно, если структура описана и обоснована в документации.

- Нельзя игнорировать целостность данных (отсутствие ключей, ограничений, транзакций).

**Максимальные требования (на 10 баллов)**

**Архитектура и функциональность + документация**

- При необходимости использованы разные типы СУБД в одном проекте (например, PostgreSQL для транзакционной части и ClickHouse для аналитики) – если это оправдано темой. + приведено обоснование в документации
- Реализованы слои данных (raw → staging → data mart) – при необходимости, в зависимости от темы проекта. + приведено обоснование в документации
- Описана версионность структуры базы – как отслеживаются изменения, как хранится история версий (например, через миграции и Git).
- Реализованы и обоснованы индексы (в зависимости от контекста проекта).
- Использованы триггеры, хранимые процедуры и пользовательские функции, реализующие часть бизнес-логики на уровне базы данных.
- Применяются представления (VIEW) или материализованные представления для упрощения и оптимизации запросов.
- Для полей, содержащих персональные данные (ФИО, e-mail, телефон и т.п.), реализовано маскирование или обезличивание данных.

**Аналитические БД (OLAP)**

**Минимальные требования (на 5 баллов)**

**Документация**

- Словарь данных: описание таблиц и колонок, типов данных, ключей, агрегатов, а также ожиданий по качеству данных.
- Описана архитектура слоёв (например, raw → staging → data mart) и назначение каждого из них.
- Логическая схема: визуальное представление должно быть понятным: выделены основные блоки и связи, допускается цветовое выделение для улучшения восприятия.

- DDL-скрипт (SQL) для создания структуры базы данных.
- Приведена схема потоков данных (data flow): откуда поступают данные, как обрабатываются, куда загружаются.
- Описано, как обеспечивается целостность и актуальность данных (валидация, дедупликация, контроль загрузок).

**Проектирование**

- Использована современная аналитическая СУБД (например, ClickHouse, Snowflake и т.д.).
- Схема включает достаточное количество таблиц-фактов и таблиц-измерений – согласно выбранной модели.
- Определены правила обновления данных (полная загрузка, инкрементальная, CDC).

**Развёртывание**

- Структура базы данных создаётся через SQL-скрипты или миграции и хранится в системе контроля версий (Git).
- Подготовлены скрипты или пайплайны для загрузки данных (ETL/ELT).

**Тестовые данные**

- Подготовлены тестовые источники и данные для демонстрации корректности загрузки и агрегации.

**Использование**

- Реализованы аналитические запросы (GROUP BY, агрегаты, оконные функции, фильтрация по измерениям).
- Продемонстрированы аналитические отчёты или другие визуализации.
- Реализованы проверки качества данных (контроль дубликатов, пустых значений, некорректных типов).

**Ограничения**

- Нельзя использовать транзакционные СУБД, не предназначенные для аналитических нагрузок (например, SQLite).
- Нельзя хранить все данные в одной таблице без выделения измерений.

- Нельзя выполнять загрузку данных вручную – все шаги должны быть воспроизводимы (через скрипты или пайплайн).

**Максимальные требования (на 10 баллов)**

**Архитектура и функциональность + документация**

- Архитектура реализована с разделением слоёв данных (например, raw → staging → ODS → data mart → data mart → semantic), и приведено объяснение их назначения.
- Реализованы ETL или ELT-процессы с инкрементальной загрузкой, контролем ошибок, логированием и отчётами по загрузкам.
- При необходимости использованы оптимизационные механизмы: партиционирование, кластеризация, распределённое хранение, денормализация.
- При необходимости построены индексы для ускорения запросов. + приведено обоснование в документации
- При необходимости реализованы представления и материализованные представления для агрегаций и отчётов.
- Применены механизмы SCD (Slowly Changing Dimensions) для учёта исторических данных.
- В документации описаны источники данных, периодичность обновления, методы контроля качества и мониторинга.

**Нереляционные БД (NoSQL)**

**Минимальные требования (на 5 баллов)**

**Документация**

- Словарь данных: описание назначения базы, коллекций / узлов / таблиц, ключей, типов данных, а также ожиданий по качеству данных.
- Описана модель данных и приведено обоснование выбора конкретной СУБД.
- Пояснено, как обеспечивается целостность данных (валидация схемы, ограничения, каскадные обновления, контроль связей и т.д.).

**Структура и проектирование**

- Использована современная NoSQL-СУБД, подходящая под задачу проекта.

- Продумана структура хранения данных.
    - Для документоориентированных СУБД, например:
        - определены коллекции, документы, поля, вложенные структуры и индексы;
        - указано, какие поля обязательны, какие вложенные, и как выполняется валидация.
- Количество коллекций или сущностей соответствует сложности проекта.

**Надёжность и безопасность**

- Настроены роли и права пользователей:
    - права на чтение/запись, управление схемой и т.д.
    - приложению запрещено работать под superuser.
- Пароли и токены не хранятся в открытом виде.

**Развёртывание и тестовые данные**

- Все данные и структура должны быть воспроизводимы (дамп, экспорт, миграции).
- Набор тестовых документов / узлов должен быть достаточен для демонстрации работы.

**Ограничения**

- Нельзя использовать NoSQL просто как «контейнер JSON’ов» без структуры и валидации.
- Нельзя хранить все данные в одной коллекции (или узле), если есть разные типы сущностей.

**Максимальные требования (на 10 баллов)**

**Архитектура и функциональность + документация**

- Использованы разные типы БД в рамках одного проекта (например, MongoDB для документов, Redis для кэша, Neo4j для связей). + приведено обоснование в документации
- Продемонстрирована грамотная модель данных:

    - для документоориентированных, например:
        - осознанный выбор между встраиванием (embedded) и ссылками (referenced) – для документоориентированных;
        - оптимизация структуры под тип запросов.
- Используются сложные запросы или механизмы агрегации:
    - для документоориентированных, например: агрегирующие пайплайны, фильтры, поиск по текстовым полям;
- Реализованы индексы разных типов (текстовые, составные, геоиндексы, TTL).