**Выбор PostgreSQL в качестве базы данных для веб-приложения с каталогом книг**

При разработке веб-приложения, работающего с каталогом книг и управляемого персоналом (пользователями и администраторами), выбор системы управления базой данных (СУБД) становится ключевым шагом. Этот выбор определяет производительность, удобство разработки и способность приложения масштабироваться. PostgreSQL была выбрана в качестве основной базы данных благодаря её функциональным преимуществам, соответствующим требованиям веб-приложения. Далее приводится расширенное обоснование выбора PostgreSQL и сравнение с другими популярными СУБД.

1. **1. Описание проекта**

Веб-приложение предоставляет функциональность для работы с каталогом книг, включая:

* хранение данных о книгах (название, автор, жанр, издатель и другие метаданные);
* добавление, изменение, удаление записей (загрузка, выгрузка бд – пока неясно);
* поиск по названию, автору и другим полям;
* фильтрацию и сортировку записей;
* разграничение доступа для различных ролей (администраторы, пользователи);
* обработку сложных транзакций (например, изменение связанных записей).

**Основные требования к базе данных:**

1. Надёжность хранения данных и обеспечение их целостности.
2. Возможность работы как с реляционными, так и с нереляционными данными.
3. Масштабируемость для работы с растущими объёмами данных и нагрузками.
4. Гибкая и надёжная интеграция с серверной частью веб-приложения.

**2. Основные кандидаты для базы данных**

**2.1 MySQL**

* **Описание:** Популярная реляционная СУБД с открытым исходным кодом, активно применяемая в веб-разработке.
* **Преимущества:**
  + Простота установки и использования.
  + Высокая производительность для простых операций чтения.
  + Большая документация и сообщество.
* **Недостатки:**
  + Ограниченные возможности работы с JSON-данными.
  + Менее строгая поддержка ACID[1] в некоторых движках (например, MyISAM).
  + Меньший набор встроенных функций для сложных операций (по сравнению с PostgreSQL).

**2.2 SQLite**

* **Описание:** Лёгкая встраиваемая база данных, популярная для небольших и мобильных приложений.
* **Преимущества:**
  + Не требует установки сервера.
  + Высокая производительность на малых объёмах данных.
  + Простота в использовании.
* **Недостатки:**
  + Не подходит для масштабируемых веб-приложений с высокой нагрузкой.
  + Ограниченный функционал для сложных транзакций и анализа данных.

**2.3 MongoDB**

* **Описание:** Документо-ориентированная NoSQL база данных.
* **Преимущества:**
  + Гибкость структуры данных.
  + Хорошо подходит для приложений с часто изменяющейся схемой данных.
  + Высокая производительность при частых операциях записи.
* **Недостатки:**
  + Сложности в реализации реляционных связей (например, книги и авторы).
  + Ограниченные возможности транзакционной обработки.
  + Необходимость дополнительных усилий для построения сложных аналитических запросов.

**2.4 Microsoft SQL Server**

* **Описание:** Коммерческая реляционная СУБД от Microsoft, используемая в корпоративных приложениях.
* **Преимущества:**
  + Интеграция с экосистемой Microsoft.
  + Высокая производительность и поддержка аналитики.
  + Гибкие инструменты управления базами данных.
* **Недостатки:**
  + Высокая стоимость лицензирования.
  + Сложность настройки.
  + Ограниченная поддержка сторонних платформ по сравнению с PostgreSQL.

**2.5 PostgreSQL**

* **Описание:** Реляционная СУБД с открытым исходным кодом, известная своей гибкостью и мощными функциями.
* **Преимущества:**
  + Полная поддержка ACID.
  + Богатый функционал: работа с JSON, полнотекстовый поиск, сложные транзакции.
  + Расширяемость за счёт модулей и плагинов.
  + Бесплатна и открыта для модификации.
* **Недостатки:**
  + Более сложная настройка по сравнению с MySQL.
  + Более требовательна к ресурсам при высоких нагрузках.

**3. Почему выбран PostgreSQL?**

**3.1 Соответствие требованиям веб-приложения**

1. **Хранение реляционных и нереляционных данных:**
   * PostgreSQL поддерживает реляционные связи между таблицами (например, книги связаны с авторами, жанрами).
   * Возможность работы с JSON и JSONB позволяет эффективно хранить метаданные и динамическую информацию, такую как пользовательские отзывы.
2. **Надёжность транзакций:**
   * Полная поддержка ACID гарантирует целостность данных при выполнении операций, что особенно важно для веб-приложения с одновременным доступом множества пользователей.
3. **Поиск и фильтрация данных:**
   * PostgreSQL предоставляет встроенные инструменты для полнотекстового поиска, которые упрощают реализацию функций поиска и фильтрации в каталоге книг.
4. **Масштабируемость:**
   * Поддержка репликации и шардинга[2] позволяет масштабировать приложение по мере роста данных и нагрузки.

**3.2 Технические преимущества PostgreSQL**

1. **Сложные запросы и аналитика:**
   * PostgreSQL поддерживает оконные функции, CTE (Common Table Expressions), триггеры и процедуры, что делает его универсальным инструментом для сложных аналитических задач.
2. **Расширяемость:**
   * Возможность подключения расширений (например, PostGIS для работы с геоданными) и пользовательских функций.
3. **Универсальность:**
   * PostgreSQL подходит как для оперативной обработки данных (OLTP), так и для аналитической обработки (OLAP).

**3.3 Интеграция с веб-приложением**

* PostgreSQL имеет драйверы для всех популярных серверных языков, включая Python, Java (используется в данном проекте), Node.js.
* Хорошо работает с ORM[3] (например, SQLAlchemy, Django ORM), что упрощает взаимодействие с базой данных.
* Имеет средства для тонкой настройки производительности в зависимости от потребностей приложения.

**4. Преимущества PostgreSQL для веб-приложения**

1. **Надёжность и безопасность:**
   * Гибкая настройка прав доступа и шифрование соединений.
   * Возможности резервного копирования и восстановления данных.
2. **Масштабируемость:**
   * PostgreSQL поддерживает горизонтальное масштабирование через репликацию (стриминг репликация, логическая репликация).
   * Подходит для работы в кластерных архитектурах, таких как Kubernetes[4], обеспечивая стабильность под высокой нагрузкой.
3. **Поддержка современных веб-функций:**
   * Встроенная работа с JSON/JSONB позволяет эффективно реализовать API для веб-приложений (например, REST или GraphQL), минимизируя преобразования данных.
   * Полнотекстовый поиск — полезная возможность для веб-приложений с поисковым функционалом. PostgreSQL предоставляет гибкие средства поиска с учётом языка, синонимов и морфологии.
4. **Обработка больших объёмов данных:**
   * PostgreSQL оптимизирован для работы с большими базами данных, что важно для веб-приложений с растущим количеством записей, таких как каталоги книг.
   * Индексация (например, B-tree, GIN, GiST) помогает ускорить операции поиска, фильтрации и сортировки.
5. **Открытый исходный код и бесплатность:**
   * PostgreSQL бесплатна, что снижает затраты на разработку и эксплуатацию веб-приложения.
   * Активное сообщество и частые обновления обеспечивают доступ к современным функциям и улучшениям.

**5. Сравнение PostgreSQL с альтернативами для веб-приложений**

1. **MySQL:**
   * Хотя MySQL проще в настройке и использовании, она уступает PostgreSQL в поддержке JSON, полнотекстового поиска и сложных транзакций, что критично для гибких веб-приложений.
2. **MongoDB:**
   * MongoDB хорошо подходит для динамических данных, но сложнее в реализации связей между документами, что важно для реляционных структур (например, связи между книгами, авторами и жанрами).
3. **SQLite:**
   * SQLite не масштабируется для веб-приложений с высоким уровнем одновременных запросов и большими объёмами данных, в отличие от PostgreSQL.
4. **Microsoft SQL Server:**
   * SQL Server обладает схожими с PostgreSQL возможностями, но имеет высокую стоимость лицензирования, что делает PostgreSQL предпочтительным выбором для проектов с открытым исходным кодом.

**Итог**

PostgreSQL — это идеальное решение для веб-приложения с каталогом книг благодаря следующим причинам:

1. **Универсальность и гибкость:**
2. **Масштабируемость и производительность:**
3. **Современные функции:**
4. **Экономическая эффективность:**
5. **Активное сообщество и поддержка:**

Благодаря этим преимуществам PostgreSQL предоставляет мощную, надёжную и гибкую основу для веб-приложения с каталогом книг, обеспечивая отличные условия для текущей разработки и будущего роста.

**II)** Разработка ER-диаграмм

**III)** Работа с бд  
**NOT NULL**: Это ограничение, которое говорит, что поле должно содержать значение, и не может быть пустым.

**NULL**: Когда поле не ограничено NOT NULL, оно может содержать пустое значение, что означает, что это поле не обязательно для заполнения.

Создание таблиц:

1)В данном SQL-запросе создаётся таблица authors, которая представляет собой сущность для хранения данных об авторах:

CREATE TABLE IF NOT EXISTS authors(

id serial primary key,

fio varchar not null,

birth\_date date,

country varchar,

nickname varchar

);

В запросе указаны следующие поля:

1. **id (serial primary key)**:
   * **Не может быть NULL**.
   * Это поле является уникальным идентификатором каждой записи в таблице. Поскольку оно является **первичным ключом (PRIMARY KEY)**, это означает, что его значение должно быть уникальным и не может быть пустым (NULL).
2. **fio (varchar not null)**:
   * **Не может быть NULL**.
   * Это поле предназначено для хранения полного имени автора (например, "Иванов Иван Иванович"). Ключевое слово NOT NULL означает, что это поле обязательно для заполнения, и оно не может содержать NULL-значение.
3. **birth\_date (date)**:
   * **Может быть NULL**.
   * Это поле предназначено для хранения даты рождения автора. Поскольку оно не имеет ограничения NOT NULL, оно может содержать значение NULL. Это означает, что если дата рождения автора неизвестна, можно оставить поле пустым.
4. **country (varchar)**:
   * **Может быть NULL**.
   * Это поле для указания страны, в которой родился автор. Поле не ограничено NOT NULL, поэтому оно может быть пустым, если информация о стране неизвестна.
5. **nickname (varchar)**:
   * **Может быть NULL**.
   * Это поле для хранения псевдонима автора, если он есть. Поскольку оно не ограничено NOT NULL, оно также может быть пустым (NULL), если у автора нет псевдонима.

Таким образом в таблице обязательными для заполнения являются поля fio и id. Остальные поля (birth\_date, country, nickname) могут быть пустыми, если соответствующая информация отсутствует.  
  
  
2) В данном SQL-запросе создаётся таблица publishing\_companies, которая представляет собой сущность для хранения данных о издательствах:

CREATE TABLE IF NOT EXISTS publishing\_companies(

name varchar primary key,

establishment\_year INTEGER,

contact\_info VARCHAR,

city VARCHAR

);

1. **name (varchar primary key)**:
   * **Не может быть NULL**.
   * Это поле содержит название издательства и является **первичным ключом (PRIMARY KEY)**. Поскольку первичный ключ не может быть пустым, это поле обязано содержать уникальное значение и не может быть NULL.
2. **establishment\_year (INTEGER)**:
   * **Может быть NULL**.
   * Это поле хранит год основания издательства. Оно не имеет ограничения NOT NULL, поэтому может быть пустым (NULL), если год основания неизвестен или не указан.
3. **contact\_info (VARCHAR)**:
   * **Может быть NULL**.
   * Это поле хранит контактную информацию издательства, такую как email. Оно не ограничено NOT NULL, что позволяет оставлять его пустым, если такая информация не предоставлена.
4. **city (VARCHAR)**:
   * **Может быть NULL**.
   * Это поле для хранения города, в котором расположено издательство. Так как поле не имеет ограничения NOT NULL, оно может быть пустым, если информация о городе не указана.

Таким образом поле **name** обязательно для заполнения и не может быть пустым. Поля **establishment\_year**, **contact\_info**, и **city** могут быть пустыми (NULL), если информация по ним отсутствует.

3) В данном SQL-запросе создаётся таблица books, которая представляет собой сущность для хранения данных о книгах:

CREATE TABLE IF NOT EXISTS books(

isbn varchar primary key,

name varchar NOT NULL,

publication\_year date,

age\_limit real,

publishing\_company varchar references publishing\_companies (name),

page\_count int,

language varchar,

cost REAL,

count\_of\_books int

);

1. **isbn (varchar primary key)**:
   * **Не может быть NULL**.
   * Это поле является уникальным идентификатором книги, и поскольку оно указано как **первичный ключ (PRIMARY KEY)**, оно должно быть уникальным и обязательно для заполнения. Поле isbn не может содержать значение NULL.
2. **name (varchar)**:
   * **Не может быть NULL**.
   * Это поле для хранения названия книги. Поскольку оно ограничено NOT NULL, оно не может содержать значение NULL. Это означает, что поле обязательно для заполнения.
3. **publication\_year (date)**:
   * **Может быть NULL**.
   * Это поле для хранения года публикации книги. Оно не ограничено NOT NULL, и если год публикации книги неизвестен, это поле может быть оставлено пустым (NULL).
4. **age\_limit (real)**:
   * **Может быть NULL**.
   * Это поле для хранения возрастного ограничения книги. Поскольку оно не ограничено NOT NULL, оно может содержать NULL в случае, если возрастное ограничение не указано.
5. **publishing\_company (varchar references publishing\_companies (name))**:
   * **Не может быть NULL**.
   * Это поле хранит название издательства, которое связано с таблицей publishing\_companies через внешний ключ. Поскольку оно ограничено правилами базы данных, оно не может быть пустым (NULL).
6. **page\_count (int)**:
   * **Может быть NULL**.
   * Это поле хранит количество страниц в книге. Поскольку оно не ограничено NOT NULL, оно может быть пустым (NULL), если информация о количестве страниц отсутствует.
7. **language (varchar)**:
   * **Может быть NULL**.
   * Это поле для хранения языка книги. Так как оно не ограничено NOT NULL, оно может содержать NULL, если язык книги не указан.
8. **cost (REAL)**:
   * **Может быть NULL**.
   * Это поле хранит стоимость книги. Оно не ограничено NOT NULL, и если цена книги неизвестна или не указана, это поле может быть пустым (NULL). Например поступила новая книга на склад и её ещё не выставили на продажу.
9. **count\_of\_books (int)**:
   * **Может быть NULL**.
   * Это поле для хранения количества экземпляров книги. Оно не ограничено NOT NULL, поэтому также может быть пустым (NULL), если информация о количестве экземпляров книги отсутствует.

### Таким образом п**оля**isbn, name и **publishing\_company** обязательны для заполнения и не могут быть пустыми. Все остальные поля могут быть пустыми, если соответствующая информация отсутствует.

4) В данном SQL-запросе создаётся таблица authorships, которая представляет собой связь между книгами и авторами, связывая их по полям book\_isbn и author\_id:

CREATE TABLE IF NOT EXISTS authorships(

book\_isbn VARCHAR NOT NULL references books(isbn),

author\_id INTEGER NOT NULL references authors(id),

constraint pk\_authorships PRIMARY KEY (book\_isbn, author\_id)

);

1. **book\_isbn (VARCHAR NOT NULL references books(isbn))**:
   * **Не может быть NULL**.
   * Это поле содержит ISBN книги, которое является внешним ключом, ссылающимся на поле isbn таблицы books. Ограничение NOT NULL означает, что это поле должно обязательно содержать значение, и оно не может быть пустым (NULL). Каждая запись в таблице authorships должна быть связана с существующей книгой.
2. **author\_id (INTEGER NOT NULL references authors(id))**:
   * **Не может быть NULL**.
   * Это поле содержит ID автора, которое является внешним ключом, ссылающимся на поле id таблицы authors. Также как и для поля book\_isbn, ограничение NOT NULL означает, что это поле должно содержать значение, и оно не может быть пустым (NULL). Каждая запись в таблице authorships должна быть связана с существующим автором.
3. **constraint pk\_authorships PRIMARY KEY (book\_isbn, author\_id)**:
   * **Не может быть NULL**.
   * Это составной (композитный) первичный ключ, состоящий из полей book\_isbn и author\_id. Так как первичный ключ по определению не может содержать NULL-значений, оба этих поля обязательно должны быть заполнены для каждой записи. Это также гарантирует, что одна и та же книга не может быть связана с одним и тем же автором более одного раза, то есть каждая пара (book\_isbn, author\_id) будет уникальной.

Таким образом поля **book\_isbn** и **author\_id** не могут быть пустыми (NULL), так как они обязательно должны быть заполнены, и связаны с другими таблицами через внешние ключи. Это обязательные поля для каждой записи в таблице authorships. Составной первичный ключ из этих двух полей гарантирует, что каждое сочетание книги и автора будет уникальным.

5) В данном SQL-запросе создаётся таблица styles, которая предназначена для хранения стилей (например, литературных, художественных и т.д.):

CREATE TABLE IF NOT EXISTS styles(

id serial primary key,

name varchar not null

);

1. **id (serial primary key)**:
   * **Не может быть NULL**.
   * Это поле представляет уникальный идентификатор стиля и использует тип serial, что означает, что значения будут автоматически генерироваться (прибавляться на 1 к предыдущему значению). Это поле является **первичным ключом (PRIMARY KEY)**, что требует его уникальности и обязательного заполнения. Таким образом, значение в поле id всегда будет уникальным и не может быть пустым (NULL).
2. **name (varchar not null)**:
   * **Не может быть NULL**.
   * Это поле хранит название стиля (например, "Реализм", "Импрессионизм", "Фантастика" и т. д.). Поскольку в запросе указано NOT NULL, это поле обязательно должно быть заполнено. Стиль обязательно должен иметь имя.

Таким образом, оба поля в таблице имеют обязательные значения: **id** автоматически генерируемое уникальное значение, не может быть пустым и **name** обязательное строковое значение, не может быть пустым.

6) В данном SQL-запросе создаётся таблица book\_styles, которая представляет собой связь между книгами и стилями:

CREATE TABLE IF NOT EXISTS book\_styles(

book\_isbn varchar not null references books(isbn),

style serial not null references styles(id),

constraint pk\_book\_styles PRIMARY KEY (book\_isbn, style)

);

1. **book\_isbn (varchar not null references books(isbn))**:
   * **Не может быть NULL**.
   * Это поле содержит ISBN книги и является внешним ключом, ссылающимся на поле isbn в таблице books. Ограничение NOT NULL гарантирует, что это поле должно содержать значение, и оно не может быть пустым (NULL). Каждая запись в таблице book\_styles должна быть связана с существующей книгой.
2. **style (serial not null references styles(id))**:
   * **Не может быть NULL**.
   * Это поле содержит идентификатор стиля, который является внешним ключом, ссылающимся на поле id в таблице styles. Поле типа serial автоматически генерирует уникальные значения. Ограничение NOT NULL указывает, что это поле также должно быть заполнено и не может быть пустым (NULL). Каждая запись в таблице book\_styles должна быть связана с существующим стилем.
3. **constraint pk\_book\_styles PRIMARY KEY (book\_isbn, style)**:
   * **Не может быть NULL**.
   * Это составной (композитный) первичный ключ, состоящий из полей book\_isbn и style. Составной первичный ключ гарантирует уникальность каждой пары книги и стиля. Это означает, что книга не может быть связана с одним и тем же стилем более одного раза. Поскольку первичный ключ не может содержать NULL, оба поля book\_isbn и style должны быть заполнены для каждой записи.

**Поле book\_isbn** обязательно для заполнения (не может быть NULL) и связано с таблицей books через внешний ключ.

**Поле style** также обязательно для заполнения (не может быть NULL) и связано с таблицей styles через внешний ключ.

**Составной первичный ключ** (состоящий из полей book\_isbn и style) гарантирует уникальность каждой записи в таблице и исключает возможность того, чтобы одна книга была связана с одним стилем несколько раз.

Таким образом, оба поля (book\_isbn и style) в таблице **book\_styles** должны быть обязательными для заполнения, и их комбинация будет уникальной для каждой записи.

7) В данном SQL-запросе создаётся таблица administrators, которая предназначена для хранения данных об администраторах:

CREATE TABLE IF NOT EXISTS administrators(

id serial PRIMARY KEY, -- Уникальный идентификатор администратора (автоматически генерируется)

login varchar NOT NULL UNIQUE, -- Логин администратора, без пробелов

password varchar NOT NULL -- Пароль администратора

);

1. **id (serial PRIMARY KEY)**:
   * **Не может быть NULL**.
   * Это поле является уникальным идентификатором администратора, которое автоматически генерируется с помощью типа serial. Это поле является **первичным ключом (PRIMARY KEY)**, и поэтому оно обязано содержать уникальное значение. Поскольку оно является первичным ключом, оно не может быть пустым (NULL).
2. **login (varchar NOT NULL UNIQUE)**:
   * **Не может быть NULL**.
   * Это поле содержит логин администратора, который должен быть уникальным. Поскольку ограничение NOT NULL указано на это поле, оно обязательно должно быть заполнено, и не может быть пустым (NULL). Также указано, что логин должен быть уникальным в таблице с помощью ограничения UNIQUE. Это гарантирует, что два администратора не могут иметь одинаковый логин.
3. **password (varchar NOT NULL)**:
   * **Не может быть NULL**.
   * Это поле предназначено для хранения пароля администратора. Поскольку оно ограничено NOT NULL, это поле обязательно для заполнения, и оно не может быть пустым (NULL). Важно заметить, что здесь нет ограничения UNIQUE, потому что один и тот же пароль может быть использован разными администраторами.

**Поле id** обязательно для заполнения, оно уникально и не может быть пустым, так как это первичный ключ.

**Поле login** также обязательно для заполнения (не может быть NULL) и должно быть уникальным для каждого администратора.

**Поле password** обязательно для заполнения (не может быть NULL), но не имеет ограничения уникальности.

**[1]ACID (от англ. atomicity, consistency, isolation, durability) — это набор требований к транзакционной системе**, обеспечивающий наиболее надёжную и предсказуемую её работу. Эти требования включают в себя **атомарность, согласованность, изоляцию и устойчивость**

**[2] Шардинг — это принцип проектирования базы данных, при котором части одной таблицы размещаются на разных шардах. Каждый шард представляет собой отдельный узел внутри кластера, который может состоять из одной или нескольких реплик — серверов, на которых дублируются данные в рамках шарда.**

[3] **ORM** (англ. **Object-Relational Mapping**, рус. объектно-реляционное отображение, или преобразование) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

[4] Kubernetes — **это портативная расширяемая платформа с открытым исходным кодом для управления контейнеризованными рабочими нагрузками и сервисами**. Она облегчает как декларативную настройку, так и автоматизацию. Kubernetes помогает упростить управление сложными и распределёнными приложениями. Он автоматизирует многие аспекты управления контейнерами, такие как развёртывание, масштабирование и обновление.