1. Характеристика предприятия

Частное Предприятие «Книжный клуб» – белорусская компания, основанная в 2009 году. Позиционирует себя как первый интернет-магазин книг в стране, который поставил перед собой задачу создания книжного клуба. Все члены клуба любителей литературы имеют уникальную возможность оперативно заказать книгу на любой вкус – от фантастики до кулинарии.

Цель – поиск лучших книг, индивидуально для каждого члена клуба исходя из его предпочтений и рекомендаций.

Основная деятельность предприятия – розничная торговля вне торговых объектов, а именно по образцу и через интернет-магазин. Помимо этого, компания осуществляет розничную торговлю сопутствующих непродовольственных товаров повседневного спроса.

Основная часть прибыли предприятия уходит на заработную плату сотрудникам и на развитие интернет-магазина, который нуждается в существенной доработке, чтобы соответствовать современным рыночным требованиям торговли.

Главным конкурентом Частного предприятия «Книжный клуб» является компания ООО «Приносим радость», которая владеет популярным интернет-магазином по продаже литературы – OZ.by.

Данные о годовой выручке Частного предприятия «Книжный клуб, внести в отчет не удалось в связи с коммерческой тайной предприятия.

2. Индивидуальное задание

**«Интернет магазин «Книжный клуб»** — сервис для реализации литературы с доставкой на дом, а также сервис предоставляет возможность оформить клубную подписку, которая дает бонусы и скидки. На рисунке 1 показана структура сервиса.

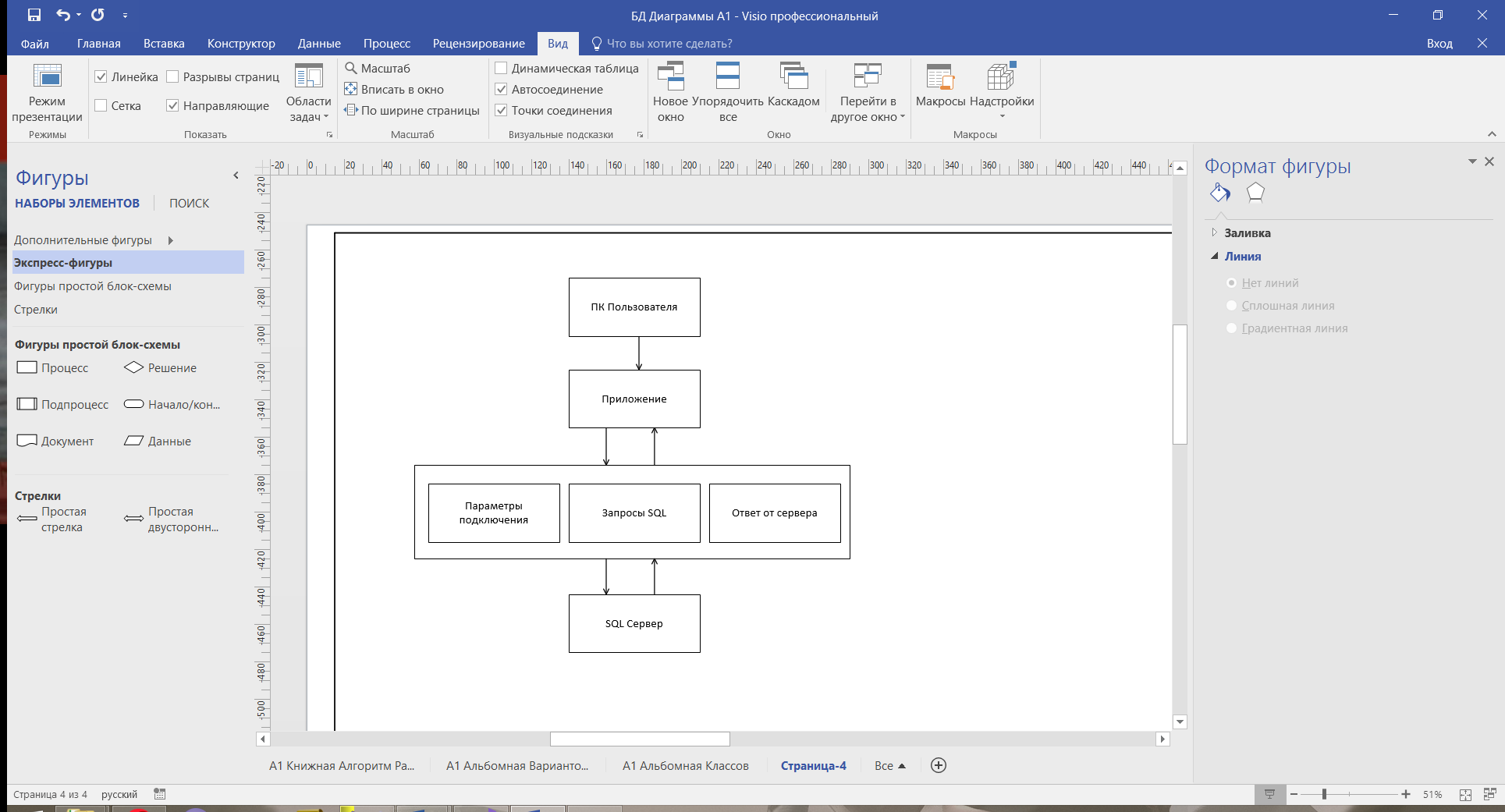


Рисунок 1– Стуктура платформы

Платформа разрабатывается по методологии Agile.

**Гибкая методология разработки** (англ. Agile software development, agile-методы) — серия подходов к разработке программного обеспечения, ориентированных на использование интерактивной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля. Существует несколько методик, относящихся к классу гибких методологий разработки, в частности экстремальное программирование, DSDM, Scrum, FDD.

Основные идеи:

* люди и взаимодействие важнее процессов и инструментов;
* работающий продукт важнее исчерпывающей документации;
* сотрудничество с заказчиком важнее согласования условий контракта;
* готовность к изменениям важнее следования первоначальному плану.

Стурктура команды представлена на рисунке 2.

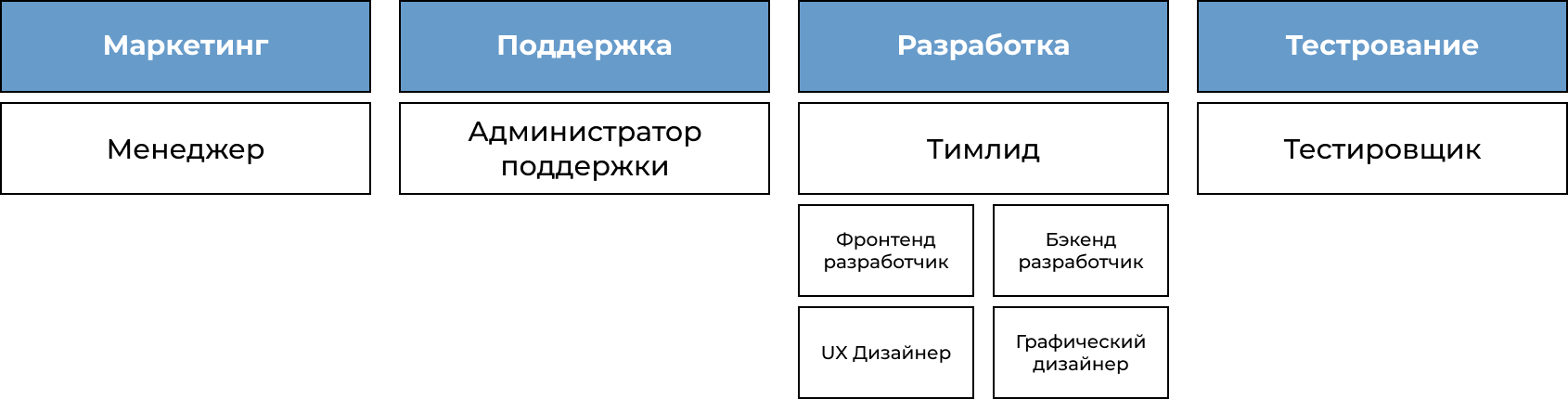


Рисунок 2 – Стуктура команды

При разработке платформы используется язык программирования PHP. Так же такие технологии как:

* SQL Server;
* PostgreSQL;
* Docker;
* Laravel;
* React JS;
* GraphQL.

**PHP** — объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в браузере. С помощью языка PHP можно создавать обычные сайты и приложения "клиент-сервер”.

В области веб-программирования, в частности серверной части, PHP — один из популярных [сценарных языков](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) (наряду с [JSP](https://ru.wikipedia.org/wiki/JSP), [Perl](https://ru.wikipedia.org/wiki/Perl) и языками, используемыми в [ASP.NET](https://ru.wikipedia.org/wiki/ASP.NET)).

Язык автоматически поддерживает HTTP Cookies, в соответствии с стандартами Netscape. Это позволяет проводить установку и чтение небольших сегментов данных на стороне клиента. Работа с Cookies организована посредством сеансов (сессий). У сессий есть срок действия (после его истечения, данные удаляются), в сессиях можно хранить и редактировать разные типы данных, в том числе сериализованные - пропущенные через serialize (процесс происходит автоматически), PHP-объекты.

В настоящее время PHP используется сотнями тысяч разработчиков. Согласно рейтингу корпорации TIOBE, базирующемся на данных поисковых систем, в мае 2016 года PHP находился на 6 месте среди языков программирования. К крупнейшим сайтам, использующим PHP, относятся [Facebook](https://ru.wikipedia.org/wiki/Facebook), [Wikipedia](https://ru.wikipedia.org/wiki/Wikipedia) и др.

Входит в [LAMP](https://ru.wikipedia.org/wiki/LAMP) — распространённый набор программного обеспечения для создания и [хостинга](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B3) [веб-сайтов](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82) ([Linux](https://ru.wikipedia.org/wiki/Linux), [Apache](https://ru.wikipedia.org/wiki/Apache_HTTP_Server), [MySQL](https://ru.wikipedia.org/wiki/MySQL), PHP). [3]

Программа на языке PHP выполняется в браузере – [прикладном программном обеспечении](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для просмотра [веб-страниц](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0), содержания [веб-документов](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82), [компьютерных файлов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB) и их [каталогов](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3_(%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)); управления [веб-приложениями](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5); а также для решения других задач. В [глобальной сети](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0) браузеры используют для запроса, обработки, манипулирования и отображения содержания [веб-сайтов](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82).

PHP крайне прост для освоения, но вместе с тем способен удовлетворить запросы профессиональных программистов. Не нужно пугаться длинного списка возможностей PHP. Программист может быстро начать, и уже в течение нескольких часов сможет создавать простые PHP-скрипты.

Основой для панели администратора стала СMS и по совместительству фреймворк – Laravel.

**Docker** – это программная платформа для быстрой разработки, тестирования и развертывания приложений. Docker упаковывает ПО в стандартизованные блоки, которые называются [контейнерами](https://aws.amazon.com/ru/containers/). Каждый контейнер включает все необходимое для работы приложения: библиотеки, системные инструменты, код и среду исполнения. Благодаря Docker можно быстро развертывать и масштабировать приложения в любой среде и сохранять уверенность в том, что код будет работать.

Использование Docker на AWS дает разработчикам и системным администраторам надежный и экономичный способ сборки, доставки и запуска распределенных приложений любого масштаба. AWS поддерживает обе модели лицензирования Docker: лицензию с открытым кодом Docker Community Edition (CE) и лицензию Docker Enterprise Edition (EE) на основе подписки.

В основе работы Docker лежит стандартизированный способ исполнения кода. Docker – это операционная система для контейнеров. Подобно тому как [виртуальная машина](https://aws.amazon.com/ru/ec2/) создает виртуальное представление аппаратного обеспечения сервера (то есть устраняет необходимость непосредственно управлять таковым), контейнеры создают виртуальное представление серверной операционной системы. После установки на каждый сервер Docker предоставляет доступ к простым командам, необходимым для сборки, запуска или остановки контейнеров.

Такие сервисы AWS, как [AWS Fargate](https://aws.amazon.com/ru/fargate/), [Amazon ECS](https://aws.amazon.com/ru/ecs/), [Amazon EKS](https://aws.amazon.com/ru/eks/) и [AWS Batch](https://aws.amazon.com/ru/batch/), упрощают работу с контейнерами Docker, а также управление ими в любом масштабе.

Использование Docker позволяет быстрее и эффективнее доставлять или перемещать код, стандартизирует выполняемые приложениями операции и в целом экономит средства, оптимизируя использование ресурсов. Благодаря Docker пользователи получают объект, который с высокой надежностью можно запускать на любой платформе. Простой и понятный синтаксис Docker обеспечивает полный контроль над выполняемыми операциями. Повсеместное внедрение контейнеров подразумевает доступ к разнообразным инструментам и готовым приложениям, которые можно использовать с Docker.

Контейнеры Docker можно использовать в качестве основных компонентов для создания современных платформ и приложений. Docker упрощает сборку и запуск распределенных микросервисных архитектур, развертывание кода с помощью стандартизированных конвейеров непрерывной интеграции и доставки, создание высокомасштабируемых систем обработки данных и полностью управляемых платформ для разработчиков.

**Laravel** представляет собой бесплатный веб-[фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), предназначенный для разработки с использованием архитектурной модели [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller). Laravel выпущен под [лицензией MIT](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_MIT).

Исходный код проекта размещается на [GitHub](https://ru.wikipedia.org/wiki/GitHub). В результате опроса в декабре 2013 года о самых популярных [PHP](https://ru.wikipedia.org/wiki/PHP)-фреймворках Laravel занял место самого многообещающего проекта на 2014 год.

В 2015 году в результате опроса по использованию [PHP](https://ru.wikipedia.org/wiki/PHP)-фреймворков среди программистов занял первое место в номинациях:

* Фреймворк корпоративного уровня;
* Фреймворк для личных проектов;

**React JS –** JavaScript-библиотека для создания пользовательских интерфейсов.

**Основные особенности React JS:**

*Декларативный.* Создавать интерактивные пользовательские интерфейсы на React — приятно и просто. Вам достаточно описать, как части интерфейса приложения выглядят в разных состояниях. React будет своевременно их обновлять, когда данные изменяются. Декларативные представления сделают код более предсказуемым и упростят отладку.

*Основан на компонентах.* Создавайте инкапсулированные компоненты с собственным состоянием, а затем объединяйте их в сложные пользовательские интерфейсы. Поскольку логика компонента написана на JavaScript, а не содержится в шаблонах, можно с лёгкостью передавать самые разные данные по всему приложению и держать состояние вне DOM.

В платформе на React реализована собственная библиотека для построения пользовательского итерфейса.

Для обеспечения гибкости данный библиотеки использовался подход разделения логики и отображения программных компонентов библиотеки. Это позволяет быстро создавать стилизованные версии библиотеки для пользователей библиотеки. Рассмотрим данный подход на примере компонента для текстового ввода (Приложение А.1).

Данный компонент содержит в себе исключительно логику работы компоненты и не более. Он отслеживает фокус, проверяет пользовательский ввод, хранит в себе пользовательский ввод, позиционирует иконку.

Далее рассмотрим пример со стилизованным компонентом текстового ввода (см. рисунок 3).

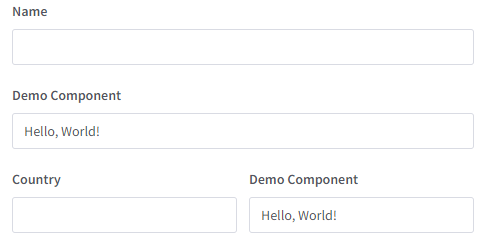


Рисунок 3 – Стилизованное поле ввода

Здесь мы видим, что компонент не содержит в себе какую-либо логику, а работает только со стилями. Таким образом пользователь библиотеки имеет возможность настраивать отображение текстового ввода в приложении под его нужды.

Данный подход очень эффективен. Он позволяет не только быстро создавать новые версии библиотек, но и помогает разделять программный код, что хорошо сказывается на поддержки данной библиотеки.

## **Паттерн «Линза»**

В функциональном программировании широко используются неизменяемые структуры данных. Работа с ними значительно отличается по сравнению с изменяемыми данными.

В основе этого лежит тот факт, что при изменении какой-либо части неизменяемой структуры данных создается ее копия, отличающаяся от оригинала этой самой измененной частью. Полное копирование всей исходной структуры не эффективно, поэтому новая структура как правило использует ссылки на неизмененные части из оригинала.

Пример. Пусть у нас есть структура данных user (Приложение А.3). У нас стоит задача поменять значение имени. Если мы работаем с этой структурой как с изменяемой, то достаточно просто изменить значение в объекте user. Но если мы работаем с этой структурой как с неизменяемой, то мы не имеем право менять данные в исходном объекте. Нам нужно создать новый объект user2 в который поместить все значения из user за исключением нового name (Приложение А.4).

Композиция гетеров осуществляется одинаково как для изменяемых так и для неизменяемых структур, но построение сетеров отличается значительно.

Для построения сетера глубиной n изменяемой структуры данных достаточно использования n – 1 гетеров и одного сетера с последнего уровня.

Для получения сетера неизменяемой структуры глубиной n необходимо n – 1 гетеров и n сетеров, т.к. необходимо обновлять все уровни начиная с 0 (исходный объект).

Для упрощения построения (компоновки) сетеров (и гетеров) неизменяемых структур данных удобно использовать инструмент линзы.

В данном проекте линзы используются для сохранения иммутабельности пользовательских данных. Также линзы позволяют удобно передавать в компонент значение и функцию меняющую это самое знаечение.

lens = Lens.onEditableComponent<models.MentorProgramAvailabilityDateInput[]>(**this**);

renderAvailabilityPeriod(item, index) {

**const** { value, onValueChange } = **this**.lens.index(index).toProps();

**const** from = value.from || **null**;

**const** to = value.to || **null**;

**return** (

<FlexRow vPadding='12' spacing='18' size='36'>

<RangeDatePicker

{ ...**this**.lens.index(index).toProps() }

format={ dateFormat }

size='36'

value={ { from, to } }

onValueChange={ val => onValueChange({ from: val.from, to: val.to, externalId: value.externalId } as models.MentorProgramAvailabilityDateInput) }

/>

<IconButton size='24' icon={ bucketIcon } onClick={ () => **this**.handleRemovePeriod(index) }/>

<FlexSpacer/>

</FlexRow>

);

}

В данном примере хорошо видны преимущества данного подхода:

1. Мы не меняем наши данные напрямую, а меняем саму ссылку на объект с данными.
2. Наш объект линзы позволяет гибко и иммутабельно работать с данными
3. Есть функции-помощники, такие, как toProps, которая позволяет получить значение value и функцию которое это значение изменяет onValueChange

## **React context**

Контекст позволяет передавать данные через дерево компонентов без необходимости передавать пропсы на промежуточных уровнях.

В типичном React-приложении данные передаются сверху вниз (от родителя к дочернему компоненту) с помощью пропсов. Однако, этот способ может быть чересчур громоздким для некоторых типов пропсов (например, выбранный язык, UI-тема), которые необходимо передавать во многие компоненты в приложении. Контекст предоставляет способ делиться такими данными между компонентами без необходимости явно передавать пропсы через каждый уровень дерева.

Контекст разработан для передачи данных, которые можно назвать «глобальными» для всего дерева React-компонентов (например, текущий аутентифицированный пользователь, UI-тема или выбранный язык).

Данный подход позволяет сделать код чище во многих случаях, уменьшая количество пропсов, которые мы должны передавать через наше приложение, и давая больше контроля корневым компонентам. Однако, это решение не является верным в каждом случае. Перемещая больше сложной логики вверх по дереву, мы нагружаем вышестоящие компоненты

В данном проекте контексты играют важную роль. Например, так реализован вызов уведомлений в веб-приложений.

.then(({ data }) => {

**this**.preloadData();

svc.uuiNotifications.show((props) => successNotification('Changes saved', props));

})

То есть вызов уведомлений происходит с помощью использования контекста уведомлений, что позволяет вызывать уведомления в любом участке кода. Такой подход избавляет от большой вложенности данных, проходящих через компоненты.

## **Маршрутизация**

В React имеется своя система маршрутизация, которая позволяет сопоставлять запросы к приложению с определенными компонентами. Ключевым звеном в работе маршрутизации является модуль react-router, который содержит основной функционал по работе с маршрутизацией. Однако если мы собираемся работать в браузере, то нам надо использовать модуль react-router-dom.

Каждый Router создает объект history который хранит путь к текущему location[1] и перерисовывает интерфейс сайта когда происходят какие то изменения пути.

Остальные функции предоставляемые в React Router полагаются на доступность объекта history через context, поэтому они должны рендериться внутри компонента Route.

<Route/> компонент это главный строительный блок React Router'а. В том случае если вам нужно рендерить элемент в зависимости от pathname URL'ов, то следует использовать компонент <Route/>

Компонент Route может быть в любом месте в роутере, но иногда нужно определять, что рендерить в одно и тоже место. В таком случае следует использовать компонент группирования Route'ов – <Switch/>. <Switch/> итеративно проходит по дочерним компонентам и рендерит только первый который подходит под location.pathname.

У Route есть 3 props'a которые описывают каким образом выполнить рендер сопоставляя prop path с location.pathname и только один из prop должен быть представлен в Route:

1. component – React компонент. Когда роут удовлетворяется сопоставление в path, то он возвращает переданный component (используя функцию React.createElement).
2. render – функция которая должна вернуть элемент React. Будет вызвана когда удовлетворится сопоставление в path. Render довольно похож на component, но используется для inline рендеринга и подстановки необходимых для элемента props[5].
3. children – в отличие от предыдущих двух props children будет всегда отображаться независимо от того сопоставляется ли path или нет.

## **CSS препроцессор LESS**

CSS - это простой язык. Он позволяет нам задавать стили для HTML элементов с помощью селекторов. Это просто даже для начинающих.

Однако, со временем размер вашего веб-проекта может вырасти, и вы получите большой объем повторяющегося кода CSS. Если вы столкнулись с этой проблемой, то самое время использовать препроцессор CSS. CSS препроцессоры не заменяют CSS.

На самом деле, препроцессоры просто дают нам дополнительные функциональные возможности, такие как переменные, операторы и функции, с которыми мы можем упростить создание и управление файлами CSS.

Препроцессоры позволяют определить свойства один раз и затем повторно использовать их в нашем проекте, что в гораздо более функционально, чего простой CSS делать не может.

LESS относительно новый препроцессор, ему около 4 лет. Его часто сравнивают с SASS, более старым препроцессором. Как и LESS, так и SASS, имеют свои корни в языке программирования Ruby, но в настоящее время используются гораздо шире. LESS теперь основан на Javascript.

Люди, использующие LESS, могут создать заранее определенный набор цветов для своего сайта. Они могут определить особый стиль один раз и затем многократно использовать его везде, где необходимо.

Чтобы воспользоваться тем, что LESS может предложить, мы сначала должны сохранить наш CSS-код в файл с расширением .less. Использовать LESS очень просто. Он работает как на веб-сервере, так и на стороне клиента.

Самый простой способ начать работу на стороне клиента. Добавьте ссылку на less-файл на вашей веб-странице, так же как добавляете CSS-файл, но с атрибутом rel установленным как "stylesheet/less", а затем файл less.js.

Этот метод работает только в современных браузерах и подходит только для использования на локальном сервере. Для рабочего сайта необходима предварительная компиляция с применением Node на веб-сервере или различные инструменты от сторонних производителей.

Одни из вещей, которые часто повторяются в файле CSS - это цвета. Одни и те же цвета повторяются в файлах CSS для разных элементов, заголовков, ссылок и т.д.

**#header** {

background-color: **#CCCCCC**

}

**.aside** {

background-color: **#CCCCCC**;

color: **#000000**;

**.sidebar** a {

border-bottom: **1px** **solid** **#CCCCCC**;

}

Переменные полезны для повторного использования отдельных значений, таких как цвет, размер шрифта. В LESS с mixins мы можем использовать набор свойств из одного или нескольких правил. Mixins могут быть использованы, например, для определения стиля границ, размер шрифта и т.д.

В приведенном выше коде мы определили верхнюю и нижнюю границу внутри класса border\_top\_bottom. Теперь, когда мы хотим добавить эти стили к другими элементами, мы можем использовать их так:

**#header** {

color: **#000000**;

**.border\_top\_bottom**;

}

**.content** a {

color: **#000000**;

**.border\_top\_bottom**;

}

Одна из вещей, распространеных в CSS, это длинные селекторы, которые нам часто приходится писать для стилей дочерних элементов:

nav { }

nav li { }

nav li a { }

nav li a**:hover** { }

nav li a**.active** { }

nav li a**.visited** { }

И если у вас есть несколько подуровней выпадающего меню, то это становится более сложным для понимания. В LESS это может быть записано как:

nav {

li {

a {

**&**amp;**:hover** { }

**&**amp;**:active** { }

**&**amp;**:visited** { }

}

}

}

LESS также позволяет нам выполнять такие операции как сложение, вычитание, умножение и деление числовых значений, цветов и переменных в таблице стилей. Допустим, мы объявили переменную padding, которая применяется ко всем нашим элементам H1. Но теперь мы хотим добавить дополнительный отступ для нашего заголовка на главной странице. Мы можем сделать это следующим образом:

**@padding:** 5px;

h1 {

padding: @**padding**;

}

h1**.page-title** {

padding: (@**padding** \* **2**);

}

Значение по умолчанию будет умножено на 2, что задаст заголовку на главной странице padding равный 10px. Операции должны выполняться в круглых скобках, однако они могут также работать без скобок.

В данном проекте CSS препроцессор LESS очень полезен при генерации цветов в файлах стилей. Для этого не нужно писать каждый цвет отдельно, достаточно пройтись по ним циклом:

## **Запрос клиента к серверу**

GraphQL - это синтаксис, который описывает как запрашивать данные, и, в основном, используется клиентом для загрузки данных с сервера. GraphQL имеет три основные характеристики:

1. Позволяет клиенту точно указать, какие данные ему нужны.
2. Облегчает агрегацию данных из нескольких источников.
3. Использует систему типов для описания данных.

GraphQL был разработан в Facebook, но даже гораздо более простые приложения могут столкнуться с ограничениями традиционных REST API интерфейсов.

Например представьте, что вам нужно отобразить список записей (posts), и под каждым опубликовать список лайков (likes), включая имена пользователей и аватары. На самом деле, это не сложно, вы просто измените API posts так, чтобы оно содержало массив likes, в котором будут объекты-пользователи (см. рисунок 4).



Рисунок 4 – Пример запроса на языке GraphQL

Но затем, при разработке мобильного приложения, оказалось что из-за загрузки дополнительных данных приложение работает медленнее. Так что вам теперь нужно два endpoint, один возвращающий записи с лайками, а другой без них.

Экстраполируйте этот сценарий на то множество источников данных и клиентских API, с которыми имеет дело Facebook, и вы поймёте почему старый добрый REST API достиг своего предела.

Facebook придумал концептуально простое решение: вместо того, чтобы иметь множество «глупых», endpoint, лучше иметь один «умный», endpoint, который будет способен работать со сложными запросами и придавать данным такую форму, какую запрашивает клиент.

Фактически, слой GraphQL находится между клиентом и одним или несколькими источниками данных; он принимает запросы клиентов и возвращает необходимые данные в соответствии с переданными инструкциями.

Пользоваться старой REST-моделью это как заказывать пиццу, затем заказывать доставку продуктов, а затем звонить в химчистку, чтобы забрать одежду. Три магазина – три телефонных звонка (см. рисунок 5).

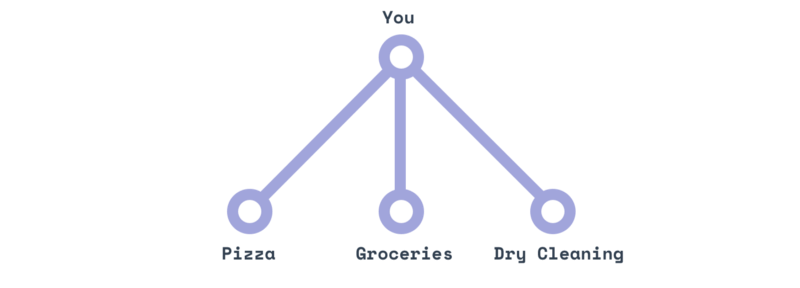


Рисунок 5 – Модель клиент-данные

GraphQL похож на личного помощника: вы можете передать ему адреса всех трех мест, а затем просто запрашивать то, что вам нужно («принеси мне мою одежду, большую пиццу и два десятка яиц») и ждать их получения.

В библиотеке для создания веб-интерфесов GraphQL позволяет оптимизировать работу запросов и не нагружать сервер, так и клиент. Это сильно влияет на производительности веб-приложения.

GraphQL как язык запросов самостоятельно используется редко, вместо этого обычно используют клиенты, которые не только могут получать и отправлять данные с веб-приложения, но и имеют множество дополнительных функций в виде: статуса запроса с сервера, обработку ошибок, удобный API.

В данном проекте таким клиентом служит Apollo Client.

## **Сборка проекта**

Webpack – это инструмент, позволяющий скомпилировать, например, JavaScript модули в единый JS-файл. Webpack также известен как сборщик модулей.

При большом количестве файлов он создает один объемный файл (или несколько файлов) для запуска вашего приложения.

Он также способен выполнять множество иных операций:

* помогает собрать воедино ваши ресурсы
* следит за изменениями и повторно выполняет задачи
* может выполнить транспиляцию JavaScript следующего поколения до более старого стандарта JavaScript (ES5) с помощью Babel, что позволит использовать новейшие функции JavaScript, не беспокоясь о том, поддерживает их браузер или нет
* может выполнить транспиляцию CoffeeScript в JavaScript
* может конвертировать встроенные изображения в data:URI
* позволяет использовать require() для CSS файлов
* может запустить webpack-dev-server (в нём встроен локальный сервер и livereload (“живая перезагрузка браузера”))
* может работать с Hot Module Replacement (замена горячего модуля)
* может разделить выходной файл (output file) на несколько файлов, чтобы избежать медленной загрузки страницы из-за большого размера JS-файла
* может выполнить Tree Shaking
* Webpack не ограничивается одним лишь фронтендом, его также успешно применяют в бэкенд разработке на Node.js.

У Webpack есть предшественники, у которых он перенял многие идеи. Основное различие заключается в том, что те инструменты известны как task runners (такс-раннеры), в то время как Webpack ничто иное, как сборщик модулей.

Webpack – это более целенаправленный инструмент. Вам достаточно указать точку входа в ваше приложение (это может быть даже HTML-файл с тегами <script>), а webpack проанализирует файлы и объединит их в один выходной JavaScript-файл, содержащий все необходимое для запуска приложения (см. рисунок 6).

По умолчанию, Webpack (начиная с 4-й версии) не требует никакой настройки, если вы соблюдаете эти правила:

* точкой входа вашего приложения является ./src/index.js
* вывод (output) размещается в ./dist/main.js
* Webpack работает в production mode (режим производства)

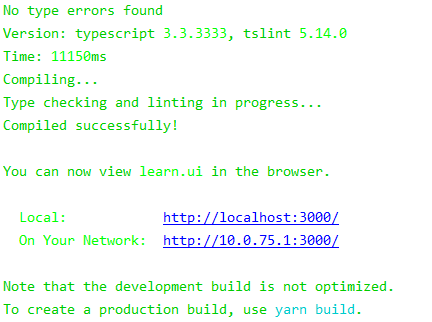


Рисунок 6 – Лог компиляции Webpack

Конечно, если понадобится, вы сможете настроить каждую мельчайшую деталь в Webpack. Конфигурационный файл Webpack -webpack.config.js хранится в корневой директории проекта.

По умолчанию, точкой входа является ./src/index.js. Нижеприведенный пример использует файл ./index.js в качестве входной точки.

module.exports = {

/\*...\*/

entry: './index.js'

/\*...\*/

}

По умолчанию, вывод размещается в ./dist/main.js. В нижеприведенном примере, результат работы в Webpack генерируется в файле app.js:

module.exports = {

/\*...\*/

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'app.js'

}

/\*...\*/

}

С помощью Webpack можно использовать оператор import или require в своем JavaScript коде для того, чтобы подключать файлы любого типа (например, CSS).

В Webpack загрузчики являются аналогами задач (tasks) в Grunt и Gulp. Они принимают содержимое файлов, а затем преобразуют его необходимым образом и включают результат преобразования в общую сборку. Например, они могут компилировать TypeScript, загружать компоненты Vue.js и многое другое.

Например, в своем коде вы можете использовать:

**import** 'style.css'

Плагины  –  это почти то же самое, что и загрузчики. Они могут сделать то, что не могут загрузчики. Ко всему прочему, Webpack построен на системе плагинов, которые вы используете в своем файле конфигурации.

module.exports = {

/\*...\*/

plugins: [

**new** CleanWebpackPlugin(['dist']),

]

/\*...\*/

}

Режимы (появившиеся в 4-й версии Webpack) настраивают среду, в которой будет работать Webpack. Режим может быть настроен на development или production (по умолчанию стоит production).

module.exports = {

entry: './index.js',

mode: 'development',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'app.js'

}

}

Режим development:

* менее оптимизирован, чем production
* работает быстрее
* не удаляет комментарии
* предоставляет более подробные сообщения об ошибках и способы их решения
* сильно облегчает отладку

Режим production работает медленнее, чем development, так как ему нужно создать более оптимизированный бандл. Полученный JavaScript файл меньше по размеру, поскольку многое из режима development в нем отсутствует.

**Система управления базами данных (СУБД)** — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Основные функции СУБД:

* Управление данными во внешней памяти (на дисках);

Управление данными в оперативной памяти с использованием дискового кэша;

* История изменений, резервное копирование и восстановление базы данных после сбоев;
* Поддержка языков БД (язык определения данных, язык манипулирования данными).

Обычно современная СУБД содержит следующие компоненты:

* Ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию,
* Процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода,
* Подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД, а также сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

Существует 4 основных типа отношений между таблицами:

* Один к одному. В этом случае каждой записи одной таблицы соответствует только одна запись другой таблицы;
* Один ко многим. Это когда одной записи главной таблицы (master) соответствует несколько записей подчиненной таблицы (detail). То есть, каждой записи, которая есть первичным ключом одной таблицы, соответствует несколько записей связанной таблицы;
* Многие к одному. Это когда нескольким записям главной таблицы отвечает одна запись подчиненной таблицы;
* Многие ко многим. Это когда в обоих таблицах существует несколько взаимосвязанных записей.

Классификации СУБД:

1. По модели данных
   1. Иерархические

Используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможна ситуация, когда объект-предок не имеет потомков или имеет их несколько, тогда как у объекта-потомка обязательно только один предок. Объекты, имеющие общего предка, называются близнецами (в программировании применительно к структуре данных дерево устоялось название братья).

Иерархической базой данных является файловая система, состоящая из корневого каталога, в котором имеется иерархия подкаталогов и файлов.

Примеры: Caché, Google App Engine Datastore API.

* 1. Сетевые

Сетевые базы данных подобны иерархическим, за исключением того, что в них имеются указатели в обоих направлениях, которые соединяют родственную информацию.

* 1. Реляционные

Практически все разработчики современных приложений, предусматривающих связь с системами баз данных, ориентируются на реляционные СУБД.

* 1. Объектно-ориентированные

Управляют базами данных, в которых данные моделируются в виде объектов, их атрибутов, методов и классов.

Этот вид СУБД позволяет работать с объектами баз данных так же, как с объектами в программировании в объектно-ориентированных языках программирования. ООСУБД расширяет языки программирования, прозрачно вводя долговременные данные, управление параллелизмом, восстановление данных, ассоциированные запросы и другие возможности.

* 1. Объектно-реляционные

Этот тип СУБД позволяет через расширенные структуры баз данных и язык запросов использовать возможности объектно-ориентированного подхода: объекты, классы и наследование.

Зачастую все те СУБД, которые называются реляционными, являются, по факту, объектно-реляционными.

1. По степени распределённости
   1. Локальные СУБД (все части локальной СУБД размещаются на одном компьютере)
   2. Распределённые СУБД (части СУБД могут размещаться на двух и более компьютерах).
2. По способу доступа к БД
   1. Файл-серверные

В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок. Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера. Недостатки: потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

* 1. Клиент-серверные

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно. Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу. Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность.

* 1. Встраиваемые

Встраиваемая СУБД — СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети. Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL либо через специальные программные интерфейсы (API).

**Стратегии работы с внешней памятью**

* СУБД с непосредственной записью — это СУБД, в которых все измененные блоки данных незамедлительно записываются во внешнюю память при поступлении сигнала подтверждения любой транзакции. Такая стратегия используется только при высокой эффективности внешней памяти.
* СУБД с отложенной записью — это СУБД, в которых изменения аккумулируются в буферах внешней памяти до наступления любого из следующих событий:
* контрольной точки;
* конец пространства во внешней памяти, отведенное под журнал. СУБД выполняет контрольную точку и начинает писать журнал сначала, затирая предыдущую информацию;
* останов. СУБД ждёт, когда всё содержимое всех буферов внешней памяти будет перенесено во внешнюю память, после чего делает отметки, что останов базы данных выполнен корректно;
* при нехватке оперативной памяти для буферов внешней памяти.
* Такая стратегия позволяет избежать частого обмена с внешней памятью и значительно увеличить эффективность работы СУБД.

Одним из основополагающих понятий в технологии баз данных является понятие *целостности*. В общем случае это понятие прежде всего связано с тем, что *база данных* отражает в информационном виде некоторый *объект* реального мира или совокупность взаимосвязанных объектов реального мира. В реляционной модели объекты реального мира представлены в виде совокупности взаимосвязанных отношений. Целостность - соответствие *информационной модели предметной области*, хранимой в базе данных, объектам реального мира и их взаимосвязям в каждый момент времени. Любое изменение в *предметной области*, значимое для построенной модели, должно отражаться в базе данных, и при этом должна сохраняться однозначная *интерпретация* информационной модели в терминах *предметной области*.

Мы отметили, что только существенные или значимые изменения *предметной области* должны отслеживаться в информационной модели. Действительно, модель всегда представляет собой некоторое упрощение реального объекта, в модели мы отражаем только то, что нам важно для решения конкретного набора задач. В модели данных должны быть предусмотрены средства и методы, которые позволят нам обеспечивать динамическое отслеживание в базе данных согласованных действий, связанных с согласованным изменением информации.

**Общие понятия и определения целостности**

*Поддержка* целостности в реляционной модели данных в ее классическом понимании включает в себя 3 аспекта.

Во-первых, это *поддержка* *структурной целостности*, которая трактуется как то, что реляционная *СУБД* должна допускать работу только с *однородными структурами* данных типа "реляционное *отношение*". При этом понятие "реляционного отношения" должно удовлетворять всем ограничениям, накладываемым на него в классической теории реляционной *БД* (отсутствие дубликатов кортежей, соответственно обязательное наличие первичного ключа, отсутствие понятия упорядоченности кортежей).

В *дополнение* к структурной целостности необходимо рассмотреть проблему неопределенных *Null* значений. Неопределенное *значение* интерпретируется в реляционной модели как *значение*, неизвестное на данный момент времени. Это *значение* при появлении дополнительной информации в любой момент времени может быть заменено на некоторое конкретное *значение*. При сравнении неопределенных значений не действуют стандартные правила сравнения: одно неопределенное *значение* никогда не считается равным другому неопределенному значению. Для выявления равенства значения некоторого атрибута неопределенному применяют специальные стандартные предикаты:

<имя атрибута>IS NULL и <имя атрибута> IS NOT NULL.

Если в данном кортеже (в данной строке) указанный *атрибут* имеет неопределенное *значение*, то *предикат* IS NULL принимает *значение* TRUE (*Истина*), а *предикат* IS NOT NULL — FALSE (*Ложь*), в противном случае *предикат* IS NULL принимает *значение* FALSE, а *предикат* IS NOT NULL принимает *значение* TRUE.

Ведение *Null* значений вызвало необходимость модификации классической двузначной логики и превращения ее в трехзначную. Все *логические операции*, производимые с неопределенными значениями, подчиняются этой логике в соответствии с заданной таблицей истинности.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 1. Таблица истинности для логических операций с неопределенными значениями | | | | |
| **А** | **В** | **Not A** | **A & B** | **A \vee  В** |
| TRUE | TRUE | FALSE | TRUE | TRUE |
| TRUE | FALSE | FALSE | FALSE | TRUE |
| TRUE | Null | FALSE | Null | TRUE |
| FALSE | TRUE | TRUE | FALSE | TRUE |
| FALSE | FALSE | TRUE | FALSE | FALSE |
| FALSE | Null | TRUE | FALSE | Null |
| Null | TRUE | Null | Null | TRUE |
| Null | FALSE | Null | FALSE | Null |
| Null | Null | Null | Null | Null |

В стандарте *SQL2* появилась возможность сравнивать не только конкретные значения атрибутов с неопределенным значением, но и результаты логических выражений сравнивать с неопределенным значением, для этого введена специальная логическая константа UNKNOWN. В этом случае операция сравнения выглядит как:

Логическое выражение > IS {TRUE | FALSE | UNKNOWN}

Во-вторых, это *поддержка* *языковой целостности*, которая состоит в том, что реляционная *СУБД* должна обеспечивать языки описания и манипулирования данными не ниже стандарта *SQL*. Не должны быть доступны иные низкоуровневые средства манипулирования данными, не соответствующие стандарту.

Именно поэтому *доступ* к информации, хранимой в базе данных, и любые изменения этой информации могут быть выполнены только с использованием операторов языка *SQL*.

В-третьих, это *поддержка* *ссылочной целостности*(Declarative *Referential Integrity*, DRI), означает обеспечение одного из заданных принципов взаимосвязи между экземплярами кортежей взаимосвязанных отношений:

* кортежи подчиненного отношения уничтожаются при удалении кортежа основного отношения, связанного с ними.
* кортежи основного отношения модифицируются при удалении кортежа основного отношения, связанного с ними, при этом на месте ключа родительского отношения ставится неопределенное Null значение.

Ссылочная *целостность* обеспечивает поддержку непротиворечивого состояния *БД* в процессе модификации данных при выполнении операций добавления или удаления.

Кроме указанных ограничений целостности, которые в общем виде не определяют семантику *БД*, вводится понятие *семантической поддержки целостности*.

Структурная, языковая и ссылочная *целостность* определяют правила работы *СУБД* с реляционными структурами данных. Требования поддержки этих трех видов целостности говорят о том, что каждая *СУБД* должна уметь это делать, а разработчики должны это учитывать при построении баз данных с использованием реляционной модели. И эти требования поддержки целостности достаточно абстрактны, они определяют допустимую форму представления и обработки информации в реляционных базах данных. Но, с другой стороны, эти аспекты никак не касаются содержания *базы данных*. Для определения некоторых ограничений, которые связаны с содержанием *базы данных*, требуются другие методы. Именно эти методы и сведены в поддержку *семантической целостности*.

То, что мы можем построить схему *базы данных* или ее концептуальную модель только из совокупности нормализованных таблиц, определяет структурную *целостность*.

Семантическая *поддержка* может быть обеспечена двумя путями: декларативным и процедурным путем. Декларативный *путь* связан с наличием механизмов в рамках *СУБД*, обеспечивающих проверку и выполнение ряда декларативно заданных правил-ограничений, называемых чаще всего "бизнес-правилами" (*Business Rules*) или декларативными ограничениями целостности.

Выделяются следующие виды декларативных ограничений целостности:

* Ограничения целостности атрибута: значение по умолчанию, задание обязательности или необязательности значений (Null), задание условий на значения атрибутов.
* Задание значения по умолчанию означает, что каждый раз при вводе новой строки в отношение, при отсутствии данных в указанном столбце этому атрибуту присваивается именно значение по умолчанию.
* В конкретных СУБД это значение будет формироваться с использованием специальных встроенных функций СУБД.
* Ограничения целостности, задаваемые на уровне доменов, при поддержке доменной структуры. Эти ограничения удобны, если в базе данных присутствуют несколько столбцов разных отношений, которые принимают значения из одного и того же множества допустимых значений. Некоторые СУБД поддерживают подобную доменную структуру, то есть разрешают определять отдельно домены, задавать тип данных для каждого домена и задавать соответственно ограничения в виде бизнес-правил для доменов. А для атрибутов задается не примитивный первичный тип данных, а их принадлежность тому или другому домену. Иногда доменная структура выражена неявно и в ряде СУБД применяется специальная терминология для этого. Так, например, в MS SQL Server вместо понятия домена вводится понятие типа данных, определенных пользователем, но смысл этого типа данных фактически эквивалентен смыслу домена. В этом случае действительно удобно задать ограничение на значение прямо на уровне домена, тогда оно автоматически будет выполняться для всех атрибутов, которые принимают значения из этого домена. А почему удобно задать это ограничение на уровне домена? А если мы зададим это ограничение для каждого атрибута, входящего в домен, разве наша система будет работать неправильно? Нет, конечно, она будет работать правильно, но представьте себе, что у вас в организации изменились правила работы, которые выражены в виде декларативных ограничений на значения.
* Да, это действительно легче, тем более что в процессе работы схема базы данных разрастается и начинает содержать более сотни отношений, и задача нетривиальная — найти все отношения, в которых ранее установлено это ограничение и исправить его.
* Одним из основных правил при разработке проекта базы данных, как мы уже упоминали раньше, является минимизация избыточности, а это означает, что если возможно информацию о чем-то, в том числе и об ограничениях, хранить в одном месте, то это надо делать обязательно.
* Ограничения целостности, задаваемые на уровне отношения. Некоторые семантические правила невозможно преобразовать в выражения, которые будут применимы только к одному столбцу. Каждый из атрибутов является в общем случае необязательным и может принимать неопределенные значения.
* Ограничения целостности, задаваемые на уровне связи между отношениями: задание обязательности связи, принципов каскадного удаления и каскадного изменения данных, задание поддержки ограничений по мощности связи. Эти виды ограничений могут быть выражены заданием обязательности или необязательности значений внешних ключей во взаимосвязанных отношениях.

Декларативные ограничения целостности относятся к ограничениям, которые являются немедленно проверяемыми. Есть ограничения целостности, которые являются откладываемыми. Эти ограничения целостности поддерживаются механизмом транзакций и триггеров.

Выделяют такие виды SQL запросов:

**DDL** (**Data Definition Language) -**язык определения данных. Задачей DDL запросов является создание БД и описание ее структуры. Запросами такого вида устанавливаются правила того, в каком виде различные данные будут размещаться в БД.

**DML** (Data Manipulation Language) - язык манипулирования данными. В число запросов этого типа входят различные команды, используя которые непосредственно производятся некоторые манипуляции с данными. DML-запросы нужны для добавления изменений в уже внесенные данные, для получения данных из БД, для их сохранения, для обновления различных записей и для их удаления из БД. В число элементов DML-обращений входит основная часть SQL операторов.

**DCL** (Data Control Language) - язык управления данными. Включает в себя запросы и команды, касающиеся разрешений, прав и других настроек СУБД.

**TCL** (Transaction Control Language) - язык управления транзакциями. Конструкции такого типа применяют чтобы управлять изменениями, которые производятся с использованием DML запросов. Конструкции TCL позволяют нам производить объединение DML запросов в наборы транзакций.

**Основные типы SQL запросов по их видам:**

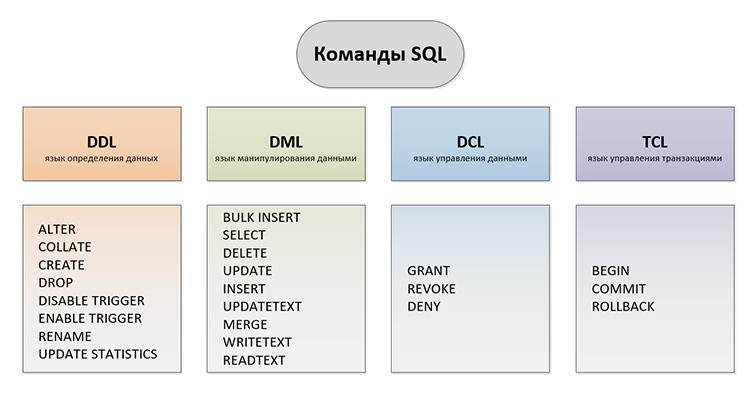


Рисунок 7 – Команды SQL

**PostgreSQL** — [свободная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) [объектно-реляционная](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) [система управления базами данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) (СУБД).

**Основные возможности**

### Функции

Функции являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом SQL, реализация дополнительной логики, например, [условных переходов](https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4) и [циклов](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), выходит за рамки SQL и требует использования некоторых языковых расширений. Функции могут писаться с использованием одного из следующих языков:

* Встроенный процедурный язык [PL/pgSQL](https://ru.wikipedia.org/wiki/PL/pgSQL), во многом аналогичный языку [PL/SQL](https://ru.wikipedia.org/wiki/PL/SQL), используемому в СУБД Oracle;
* Скриптовые языки — [PL/Lua](https://ru.wikipedia.org/w/index.php?title=PL/Lua&action=edit&redlink=1), [PL/LOLCODE](https://ru.wikipedia.org/w/index.php?title=PL/LOLCODE&action=edit&redlink=1), [PL/Perl](https://ru.wikipedia.org/wiki/PL/Perl), [PL/PHP](https://ru.wikipedia.org/w/index.php?title=PL/PHP&action=edit&redlink=1), [PL/Python](https://ru.wikipedia.org/wiki/PL/Python), [PL/Ruby](https://ru.wikipedia.org/w/index.php?title=PL/Ruby&action=edit&redlink=1), [PL/sh](https://ru.wikipedia.org/w/index.php?title=PL/sh&action=edit&redlink=1), [PL/Tcl](https://ru.wikipedia.org/wiki/PL/Tcl), [PL/Scheme](https://ru.wikipedia.org/wiki/PL/Scheme), [PL/v8](https://ru.wikipedia.org/w/index.php?title=PL/v8&action=edit&redlink=1) ([Javascript](https://ru.wikipedia.org/wiki/Javascript));
* Классические языки — [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Java](https://ru.wikipedia.org/wiki/Java) (через модуль [PL/Java](https://ru.wikipedia.org/w/index.php?title=PL/Java&action=edit&redlink=1));
* Статистический язык [R](https://ru.wikipedia.org/wiki/R_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) (через модуль [PL/R](https://ru.wikipedia.org/w/index.php?title=PL/R&action=edit&redlink=1)).

PostgreSQL допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса.

Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя.

Иногда функции отождествляются с [хранимыми процедурами](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0), однако между этими понятиями есть различие. С девятой версии возможно написание автономных блоков, которые позволяют выполнять код на процедурных языках без написания функций, непосредственно в клиенте.

### Триггеры

[Триггеры](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D0%B3%D0%B3%D0%B5%D1%80_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) определяются как функции, инициируемые [DML](https://ru.wikipedia.org/wiki/Data_Manipulation_Language)-операциями. Например, операция [INSERT](https://ru.wikipedia.org/wiki/Insert_(SQL)) может запускать триггер, проверяющий добавленную запись на соответствия определённым условиям. При написании функций для триггеров могут использоваться различные языки программирования (см. выше).

Триггеры ассоциируются с таблицами. Множественные триггеры выполняются в алфавитном порядке.

### Правила и представления

Механизм правил представляет собой механизм создания пользовательских обработчиков не только [DML](https://ru.wikipedia.org/wiki/DML)-операций, но и операции выборки. Основное отличие от механизма триггеров заключается в том, что правила срабатывают на этапе разбора запроса, до выбора оптимального плана выполнения и самого процесса выполнения. Правила позволяют переопределять поведение системы при выполнении SQL-операции к таблице. Хорошим примером является реализация механизма [представлений](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *views*): при создании представления создается правило, которое определяет, что вместо выполнения операции выборки к представлению система должна выполнять операцию выборки к базовой таблице/таблицам с учётом условий выборки, лежащих в основе определения представления. Для создания представлений, поддерживающих операции обновления, правила для операций вставки, изменения и удаления строк должны быть определены пользователем.

### Индексы

В PostgreSQL имеется поддержка [индексов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) следующих типов: [B-дерево](https://ru.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), [хеш](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0), [GiST](https://ru.wikipedia.org/wiki/GiST), [GIN](https://ru.wikipedia.org/wiki/GIN), [BRIN](https://ru.wikipedia.org/wiki/BRIN), [Bloom](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%BB%D1%8C%D1%82%D1%80_%D0%91%D0%BB%D1%83%D0%BC%D0%B0). При необходимости можно создавать новые типы индексов. Индексы в PostgreSQL обладают следующими свойствами:

* возможен просмотр индекса не только в прямом, но и в обратном порядке — создание отдельного индекса для работы конструкции ORDER BY ... DESC не нужно;
* возможно создание индекса над несколькими столбцами таблицы, в том числе над столбцами различных типов данных;
* индексы могут быть функциональными, то есть строиться не на базе набора значений некоего столбца/столбцов, а на базе набора значений функции от набора значений;
* индексы могут быть частичными, то есть строиться только по части таблицы (по некоторой её проекции); в некоторых случаях это помогает создавать намного более компактные индексы или достигать улучшения производительности за счёт использования разных типов индексов для разных (например, с точки зрения частоты обновления) частей таблицы;
* [планировщик запросов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D1%89%D0%B8%D0%BA_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2) может использовать несколько индексов одновременно для выполнения сложных запросов.

### Многоверсионность ([MVCC](https://ru.wikipedia.org/wiki/MVCC))

PostgreSQL поддерживает одновременную модификацию БД несколькими пользователями с помощью механизма Multiversion Concurrency Control ([MVCC](https://ru.wikipedia.org/wiki/MVCC)). Благодаря этому соблюдаются требования [ACID](https://ru.wikipedia.org/wiki/ACID) и практически отпадает нужда в блокировках чтения.

### Типы данных

PostgreSQL поддерживает большой набор встроенных типов данных:

* Численные типы
  + Целые
  + С фиксированной точкой
  + С плавающей точкой
  + Денежный тип (отличается специальным форматом вывода, а в остальном аналогичен числам с фиксированной точкой с двумя знаками после запятой)
* Символьные типы произвольной длины
* Двоичные типы (включая [BLOB](https://ru.wikipedia.org/wiki/BLOB))
* Типы «дата/время» (полностью поддерживающие различные форматы, точность, форматы вывода, включая последние изменения в часовых поясах)
* Булев тип
* Перечисление
* Геометрические примитивы
* Сетевые типы
  + [IP](https://ru.wikipedia.org/wiki/IP) и [IPv6](https://ru.wikipedia.org/wiki/IPv6)-адреса
  + [CIDR](https://ru.wikipedia.org/wiki/CIDR)-формат
  + [MAC-адрес](https://ru.wikipedia.org/wiki/MAC-%D0%B0%D0%B4%D1%80%D0%B5%D1%81)
* [UUID](https://ru.wikipedia.org/wiki/UUID)-идентификатор
* [XML](https://ru.wikipedia.org/wiki/XML)-данные
* [Массивы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))
* [JSON](https://ru.wikipedia.org/wiki/JSON)
* Идентификаторы объектов БД
* Псевдотипы

Более того, пользователь может самостоятельно создавать новые требуемые ему типы и программировать для них механизмы индексирования с помощью [GiST](https://ru.wikipedia.org/wiki/GiST).

### Пользовательские объекты

PostgreSQL может быть расширен пользователем для собственных нужд практически в любом аспекте. Есть возможность добавлять собственные:

* Преобразования типов
* Типы данных
* Домены (пользовательские типы с изначально наложенными ограничениями)
* Функции (включая агрегатные)
* Индексы
* Операторы (включая переопределение уже существующих)
* Процедурные языки

### Наследование и [партицирование](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

Таблицы могут наследовать характеристики и наборы полей от других таблиц (родительских). При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице.

В PostgreSQL 10 был добавлен механизм [партицирования](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) таблиц. [Партицирование](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) предназначено для разделения одной таблицы на несколько, так называемые партиции. [Партицирование](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) схоже с наследованием, но имеет более дружелюбный к пользователю синтаксис и более строгие ограничения, что позволяет выполнять дополнительные оптимизации при планировании запросов.

Заключение

Во время прохождения практики в Частное Предприятие «Книжный клуб», я подкрепил полученные во время учебы теоретические знания практическими навыками. Познакомился с рабочими процессами в организации, структурой и принципами функционирования предприятия. Развил коммуникативные навыки. Получил опыт работы в коллективе.

Во время прохождения практики выполнял все поставленные задачи. Индивидуальное задание, полученное на предприятии, было выполнено полностью и в срок.

Во время прохождения практики расширил свои знания в области информационных систем, компьютерных сетей.

В данной практике перед нами было представлено множество целей:

- ознакомление со структурой предприятия, организацией и управлением;

- изучить основные информационные потоки и документооборот на предприятии;

- углубить и закрепить знания, полученные при изучении специальных дисциплин.

В результате прохождения практики я научился работать с современными системами программного обеспечения, операционными системами, облачными системами, обслуживающими сервисными программами.

Список использованной литературы

1 JavaScript для профессионалов. СПБ: «Вильямс», 2016

2 Компьютерные сети - Таненбаум Эндрю, Уэзеролл Дэвид, издание Санкт-Петербург, 2014

3 www.ru.wikipedia.org

4 «Компьютерные сети. Принципы, технологии, протоколы.» /Олифер, В.Г., Олифер, Н.А./ СПб.:Питер, 2002.-384c.

5 «Компьютерные сети» /Танненбаум, Э./ – СПб.:Питер, 2002. – 459c.

6 «Защита компьютерной информации». /Анин, Б./ – СПб.: БЧВ, 2000. -384c.

7 «Вычислительные системы, сети и телекоммуникации.» /Пятибратов, М.Д./ – ФИС, 1998. – 279с.

8 «TCP/IP. Архитектура, протоколы, реализация.» /Фейт, С., Лори, М./ СПб : БЧВ, 2000. – 562с.

9 «Сети ЭВМ: протоколы, стандарты, интерфейсы.» /Блэк, Ю./ М.: Мир, 1990. – 321с.

10 «Стандарты по локальным вычислительным сетям: Справочник» /Под ред. Шерба, В.К., Киреичев, В.М., Самойленко, С.И./ М.: Радио и связь, 1990 – 356c.

11 «Проектирование распределенных информационно-вычислительных систем.» / Решетняк, В.Н., Гузик, В.Ф., Сидоренко, В.Г./ Учеб. пособие. Таганрог: ТРТУ,1996.–284c