**Содержание**

[Введение 5](#_Toc167055316)

[1. Анализ требований к программному средству. 6](#_Toc167055317)

[1.1. Аналитический обзор аналогов. 6](#_Toc167055318)

[1.1.1. Аналог «TicketPro». 6](#_Toc167055319)

[1.1.2. Аналог «Квiткi.by». 8](#_Toc167055320)

[1.1.3. Аналог «Bezkassira.by». 11](#_Toc167055321)

[1.3. Вывод. 15](#_Toc167055322)

[2. Разработка архитектуры проекта. 16](#_Toc167055323)

[2.1. Диаграммы UML, взаимосвязь всех компонентов. 16](#_Toc167055324)

[2.2. Описание информационных объектов и ограничений целостности. 18](#_Toc167055325)

[2.2.1. Таблица Users. 18](#_Toc167055326)

[2.2.2. Таблица Events. 18](#_Toc167055327)

[2.2.3. Таблица Locations. 19](#_Toc167055328)

[2.2.4. Таблица SectorRows. 19](#_Toc167055329)

[2.2.5. Таблица Categories. 19](#_Toc167055330)

[2.2.6. Таблица Subcategories. 19](#_Toc167055331)

[2.2.7. Таблица Tickets. 20](#_Toc167055332)

[2.2.8. Таблица Sales. 20](#_Toc167055333)

[2.2.9. Таблица Organizers. 20](#_Toc167055334)

[2.2.10. Таблица Comments. 21](#_Toc167055335)

[2.2.11. Таблица Managers. 21](#_Toc167055336)

[2.2.12. Таблица UserQuestions. 21](#_Toc167055337)

[2.2.13. Таблица OrganizerQuestions. 22](#_Toc167055338)

[2.2.14. Таблица UserBlocks. 22](#_Toc167055339)

[2.2.15. Таблица OrganizerBlocks. 22](#_Toc167055340)

[2.2.16. Таблица Roles. 23](#_Toc167055341)

[2.2.17. Таблица ShoppingCart. 23](#_Toc167055342)

[2.2.18. Таблица CheckingOrganizers. 23](#_Toc167055343)

[2.2.19. Таблица EventsSchedule. 24](#_Toc167055344)

[2.2.20. Таблица TicketRefund. 24](#_Toc167055345)

[2.3. Вывод. 24](#_Toc167055346)

[3. Разработка модели базы данных. 25](#_Toc167055347)

[3.1. Создание необходимых объектов. 25](#_Toc167055348)

[3.1.1. Таблицы. 25](#_Toc167055349)

[3.1.2. Процедуры. 25](#_Toc167055350)

[3.1.3. Функции. 31](#_Toc167055351)

[3.1.4. Триггеры. 32](#_Toc167055352)

[3.1.5. Синонимы. 35](#_Toc167055353)

[3.1.6. Индексы. 35](#_Toc167055354)

[3.2. Описание используемой технологии. 36](#_Toc167055355)

[3.3. Вывод. 38](#_Toc167055356)

[4. Установка, настройка и использование Oracle. 39](#_Toc167055357)

[4.1. Подключение базы данных к приложению WPF. 39](#_Toc167055358)

[4.2. Создание ролей для разграничения доступа. 40](#_Toc167055359)

[4.3. Описание процедур экспорта и импорта. 41](#_Toc167055360)

[4.4. Тестирование производительности базы данных. 42](#_Toc167055361)

[4.5. Вывод. 43](#_Toc167055362)

[5. Тестирование, проверка работоспособности и анализ полученных результатов. 44](#_Toc167055363)

[5.1. Тестирование гостевой части. 44](#_Toc167055364)

[5.2. Тестирование пользовательской части. 47](#_Toc167055365)

[5.3. Тестирование области работы организатора. 55](#_Toc167055366)

[5.4. Тестирование области работы менеджера. 56](#_Toc167055367)

[5.3. Вывод. 60](#_Toc167055368)

[Заключение 61](#_Toc167055369)

[Список использованных литературных источников 62](#_Toc167055370)

[Приложение А. Листинг таблиц 63](#_Toc167055371)

[Приложение Б. Листинг процедур 65](#_Toc167055372)

[Приложение В. Листинг триггеров 73](#_Toc167055373)

# Введение

Современные технологии значительно упростили доступ к различным культурным и развлекательным событиям, таким как концерты, спортивные соревнования, театральные постановки, выставки и фестивали. Они предлагают широкий спектр впечатлений и возможностей для досуга, способствуя развитию культурного и духовного потенциала общества.

Вместе с тем, индустрия продажи билетов также столкнулась с новыми вызовами и проблемами, требующими решения. Необходимость в эффективном управлении данными, обеспечении безопасности информации и создании удобного пользовательского опыта становится все более актуальной.

Разработка базы данных для сервиса по продаже билетов на мероприятия становится ключевой задачей. Правильно спроектированная и реализованная база данных обеспечит эффективное хранение и управление информацией о мероприятиях, пользователях, заказах и других аспектах сервиса, что позволит обеспечить стабильную и безопасную работу в целом.

Цель курсовой работы заключается в разработке такой базы данных, которая сможет эффективно поддерживать функционирование сервиса по продаже билетов на мероприятия. Это включает в себя создание системы управления каталогом мероприятий, реализацию функционала покупки и возврата билетов, а также создание системы поддержки пользователей.

Развертывание данной базы данных на платформе Oracle и тестирование ее производительности позволят выявить и устранить возможные проблемы, обеспечивая стабильную и бесперебойную работу сервиса. Такой подход позволит сделать доступ к мероприятиям еще более удобным и доступным для широкой аудитории, способствуя развитию культурного и развлекательного потенциала общества.

# 1. Анализ требований к программному средству.

## 1.1. Аналитический обзор аналогов.

Аналитический обзор аналогов является неотъемлемой частью исследовательского процесса при создании базы данных для сервиса по продаже билетов на мероприятия. В данном разделе рассматриваются существующие аналогичные решения на рынке, их особенности, преимущества и недостатки. Целью данного анализа является выявление успешных практик и ошибок конкурентов, что позволит оптимизировать и улучшить проект.

### 1.1.1. Аналог «TicketPro».

Первый аналог – TicketPro [1]. При регистрации на веб-сайте в качестве пользователя, запрашивается информация об электронной почте, имени и фамилии, телефоне, поле. На рисунке 1.1 представлен личный аккаунт, который включает возможность изменения личных данных, подписку на рассылку, подробную историю покупок.

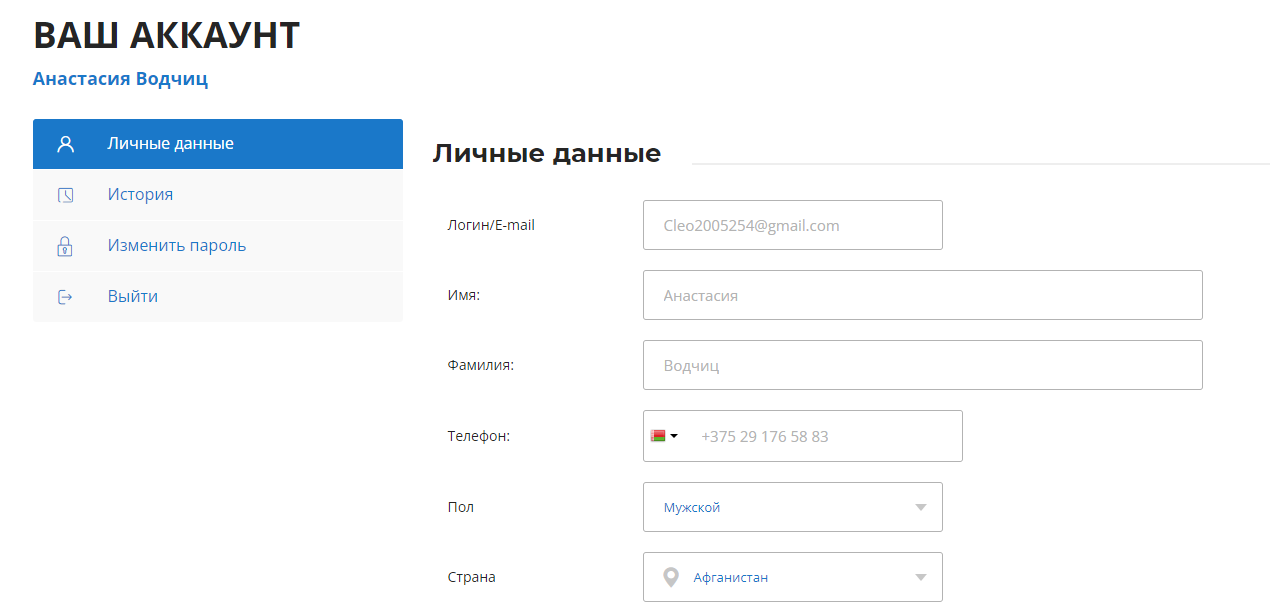


Рисунок 1.1 – Личный аккаунт веб-сайта TicketPro

Мероприятия на сайте разделены на несколько категорий, включая спорт, концерты, театр, для детей и разное. В соответствии с рисунком 1.2, основные категории представлены отдельными ячейками на главной странице, подкатегории можно просмотреть в бургер-меню при наведении на основные. В шапке веб-сайта располагается кнопка для поиска мероприятий по названию.

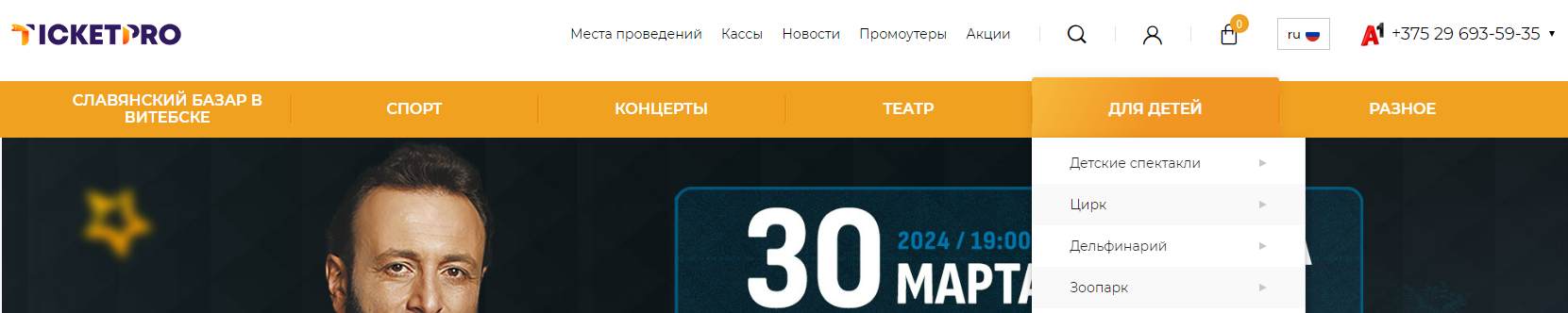


Рисунок 1.2 – Разделение мероприятий на категории веб-сайта TicketPro

Каждая категория мероприятий предоставляет возможность фильтрации по периоду, месту проведения и подкатегории. Присутствует сортировка по дате и по названию (как в убывающем, так и в возрастающем порядке). Фильтрация и сортировка представлены на рисунке 1.3.

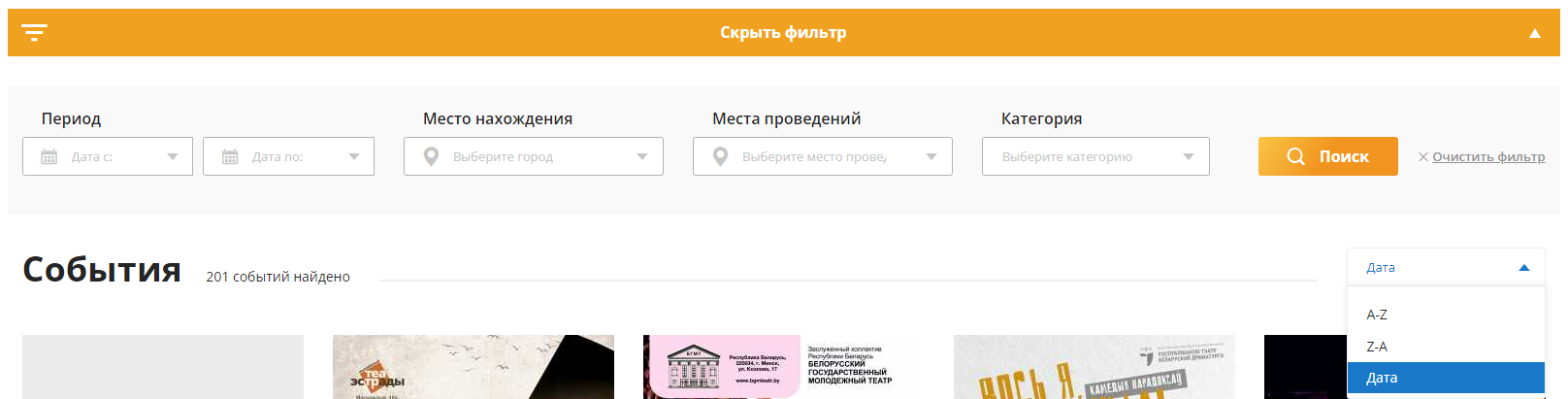


Рисунок 1.3 – Фильтрация и сортировка веб-сайта TicketPro

Для каждого мероприятия предоставлена отдельная веб-страница, продемонстрированная на рисунке 1.4, с полной информацией о дате, времени, месте проведения, организаторе. Отсутствуют комментарии к мероприятиям.

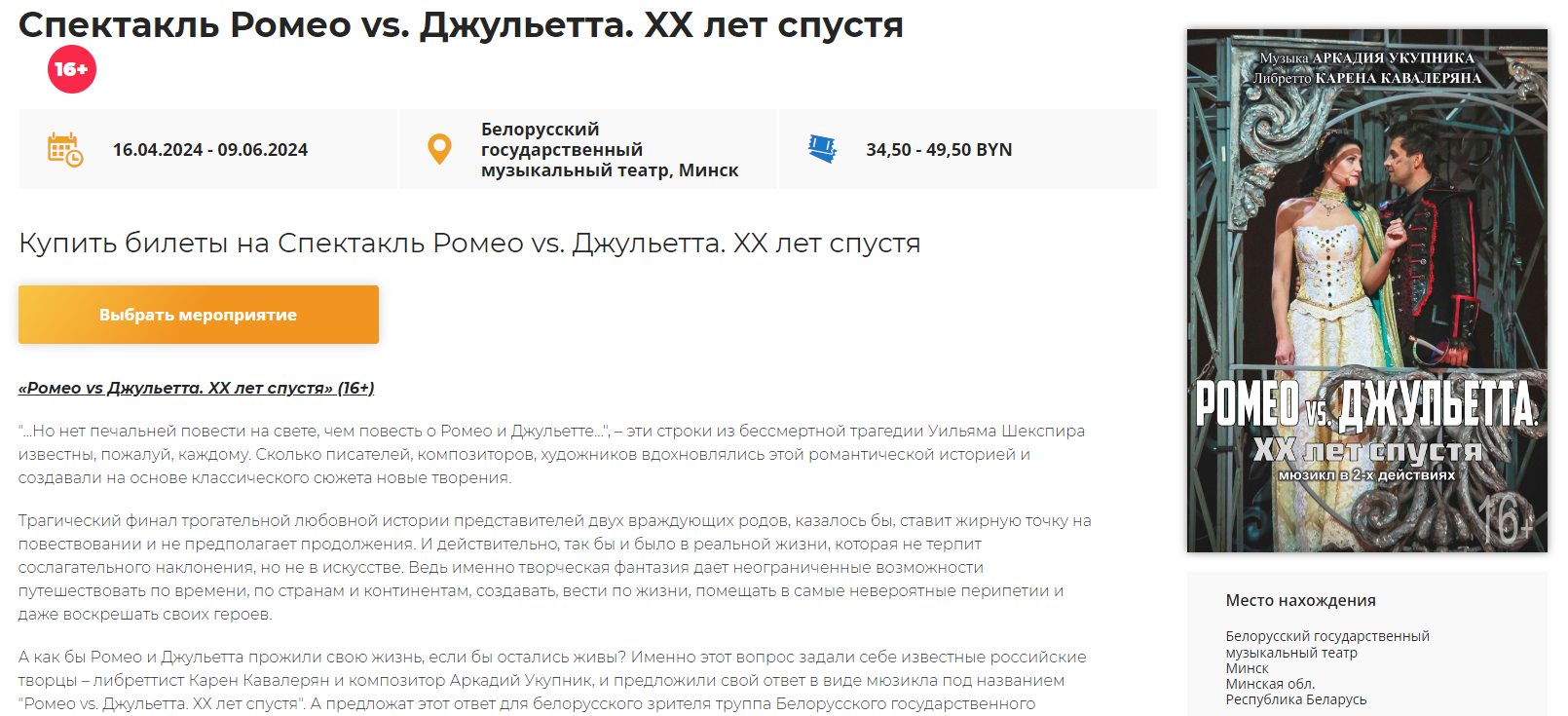


Рисунок 1.4 – Страница мероприятия веб-сайта TicketPro

Страница выбора билета на плане зала, которая представлена на рисунке 1.5, реализовано удобно, чётко приписаны цены, номера мест, рядом располагается корзина, из которой можно перейти к оформлению заказа. Присутствует возможность как очистить всю корзину, так и удалить отдельный билет из неё, прописана итоговая стоимость. Если выйти из страницы выбора билета при работе с корзиной, она не очищается, выбранные билеты остаются в корзине, к которой можно перейти через шапку веб-сайта в любой момент.

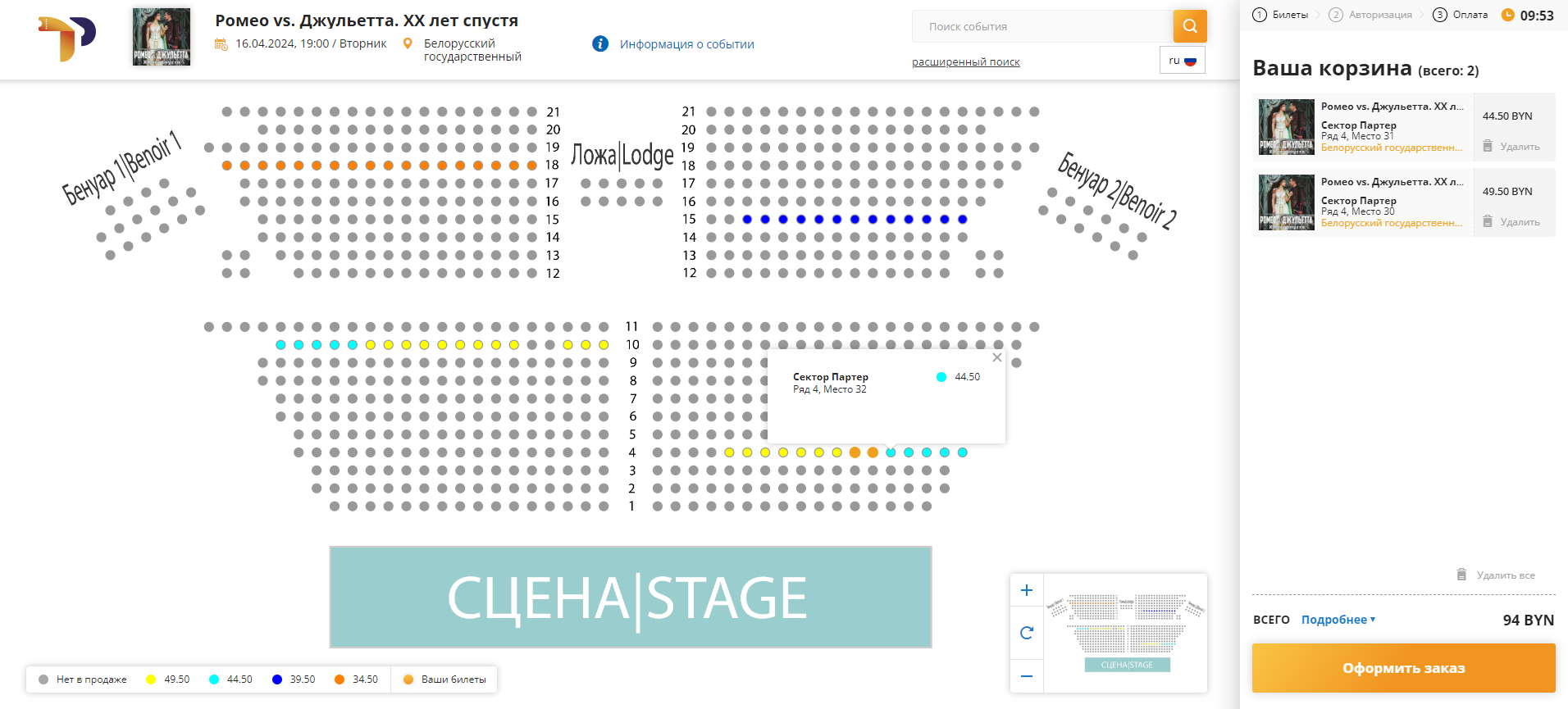


Рисунок 1.5 – Страница выбора билетов веб-сайта TicketPro

Во время оформления заказа, можно авторизоваться и работать через свой аккаунт, а также можно продолжить анонимно (в таком случае придётся отдельно указывать имя, фамилию, телефон, электронный адрес). Финальным этапом при оформлении заказа является выбор способа оплаты. Оформление заказа представлено на рисунке 1.6.

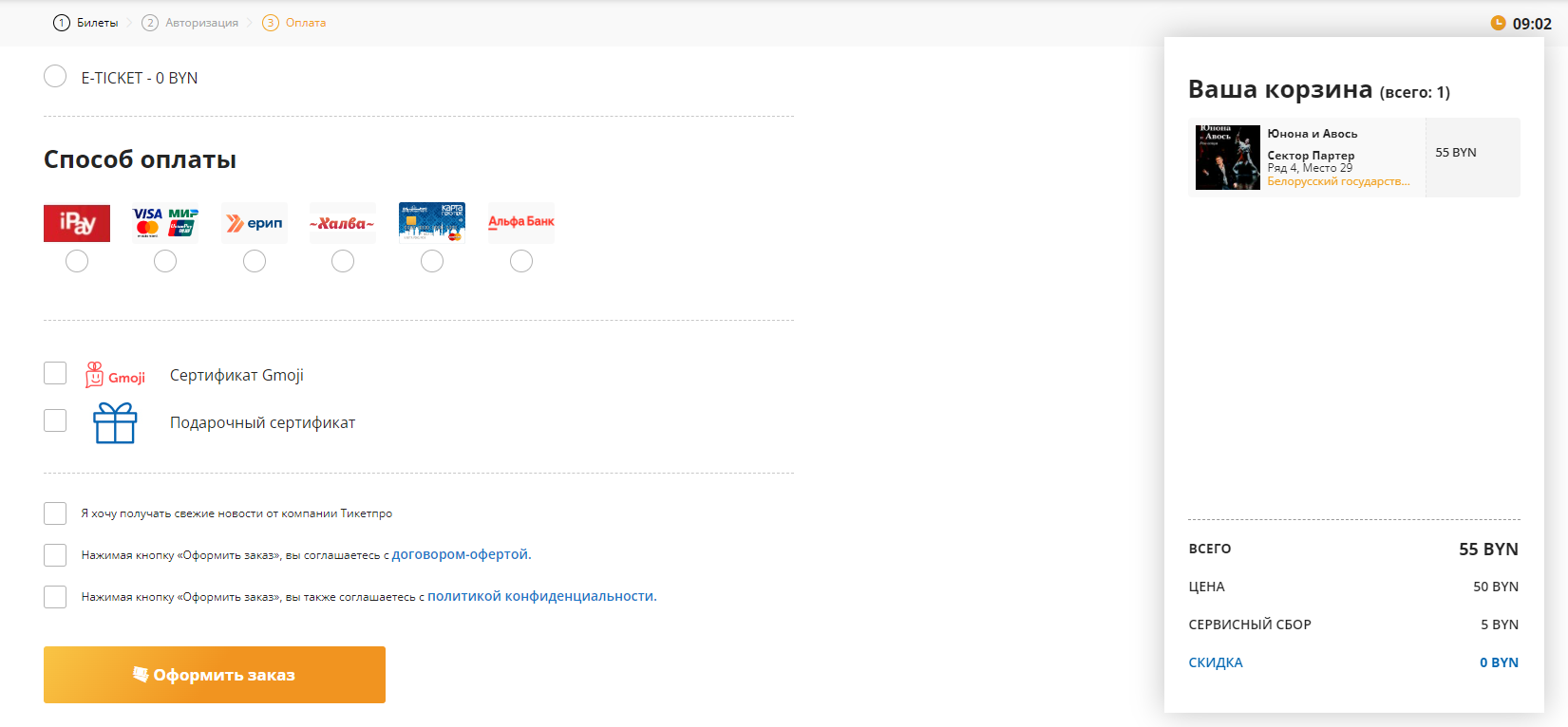


Рисунок 1.6 – Оформление заказа на веб-сайте TicketPro

На веб-сайте отсутствует удобная система регистрации для организаторов мероприятий. Есть вся контактная информация, включая социальные сети, а также форма для отправки сообщений службе поддержки. Возврат билетов осуществляется только по поручению и от имени организатора мероприятий. Для возврата необходимо написать заявку на почту, указанную в контактах веб-сайта.

### 1.1.2. Аналог «Квiткi.by».

Следующий аналог – Квiткi.by [2]. При регистрации на веб-сайте в качестве пользователя, запрашивается электронная почта, имя, фамилия, телефон. Личный кабинет, который представлен на рисунке 1.7, содержит текущие билеты, историю покупок, настройку аккаунта и условия договора.

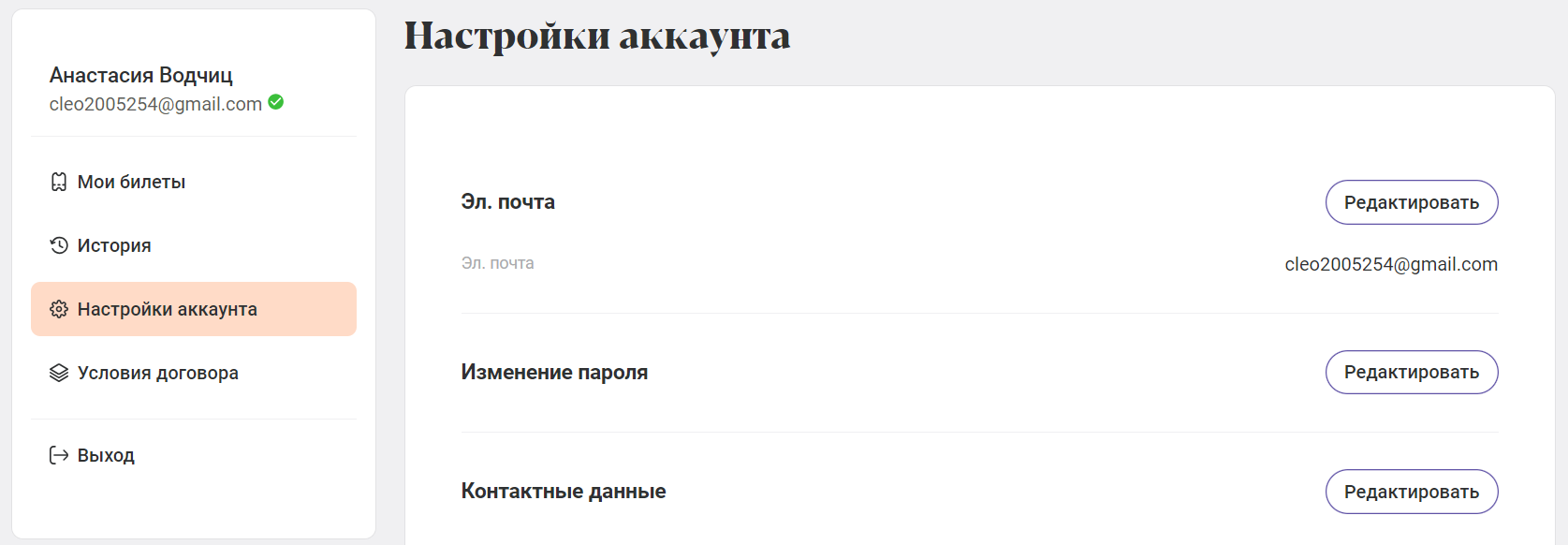


Рисунок 1.7 – Личный аккаунт веб-сайта Квiткi.by

Деление мероприятий на музыку, театр, большой театр, цирк, спорт, фестивали и выставки продемонстрировано на рисунке 1.8. Также имеется общий каталог, содержащий все существующие мероприятия. В шапке веб-сайта расположена строка для поиска мероприятий.

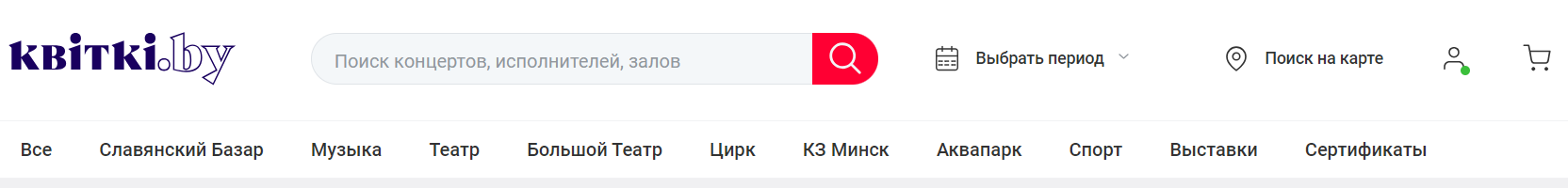


Рисунок 1.8 – Разделение мероприятий на категории веб-сайта Квiткi.by

Каждый раздел мероприятий можно отфильтровать по периоду и месту проведения, статусу мероприятия (в продаже, возврат, продано), подкатегории мероприятия. Присутствует сортировка по дате, цене, название (как по возрастанию, так и по убыванию). Фильтрация и сортировка представлены на рисунке 1.9.

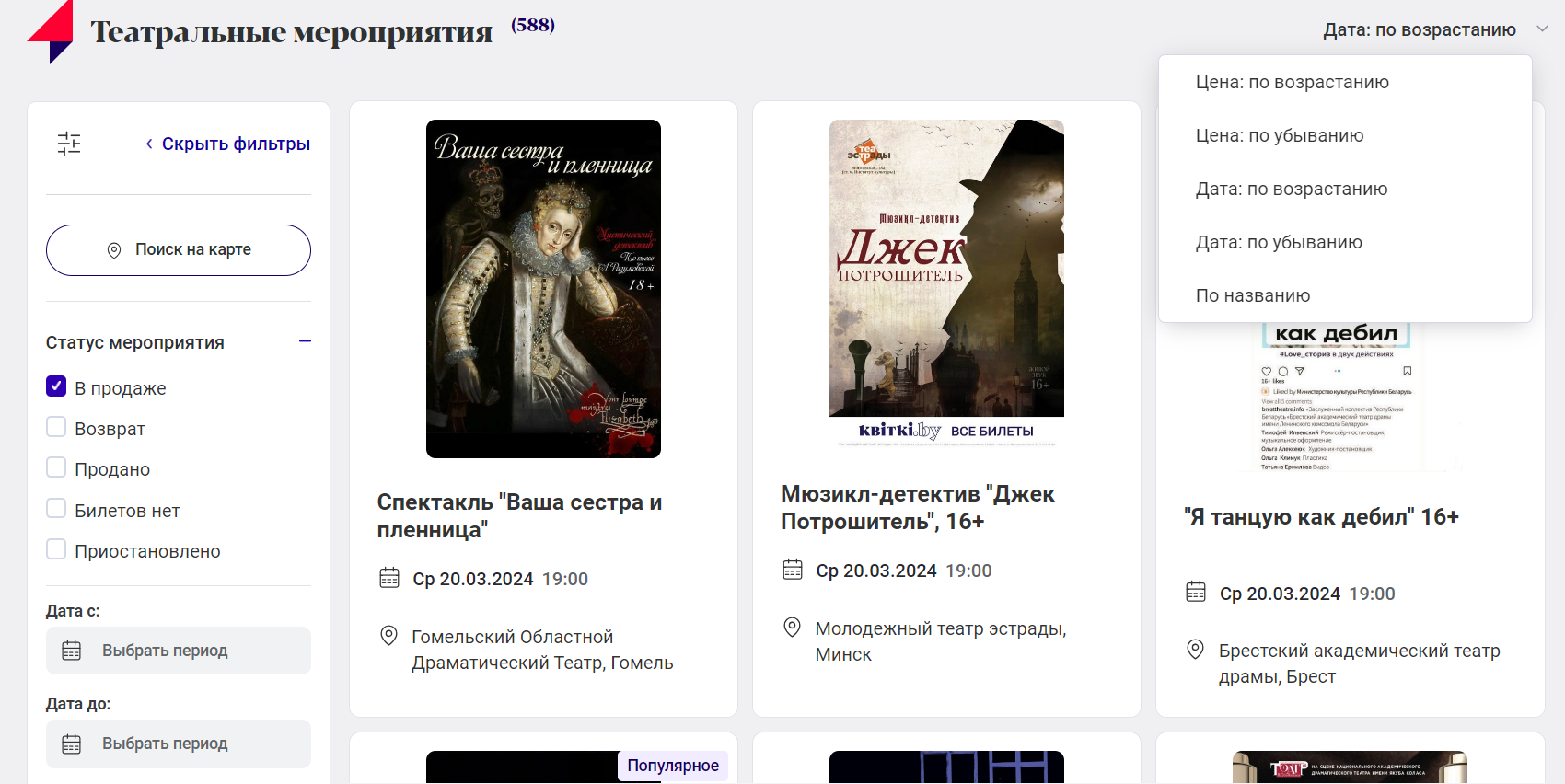


Рисунок 1.9 – Фильтрация и сортировка веб-сайта Квiткi.by

В соответствии с рисунком 1.10, по каждому мероприятию имеется отдельная веб-страница с полной информацией, включая название, дату, время, место проведения, цену, описание, организатора, продолжительность. Отсутствуют комментарии.

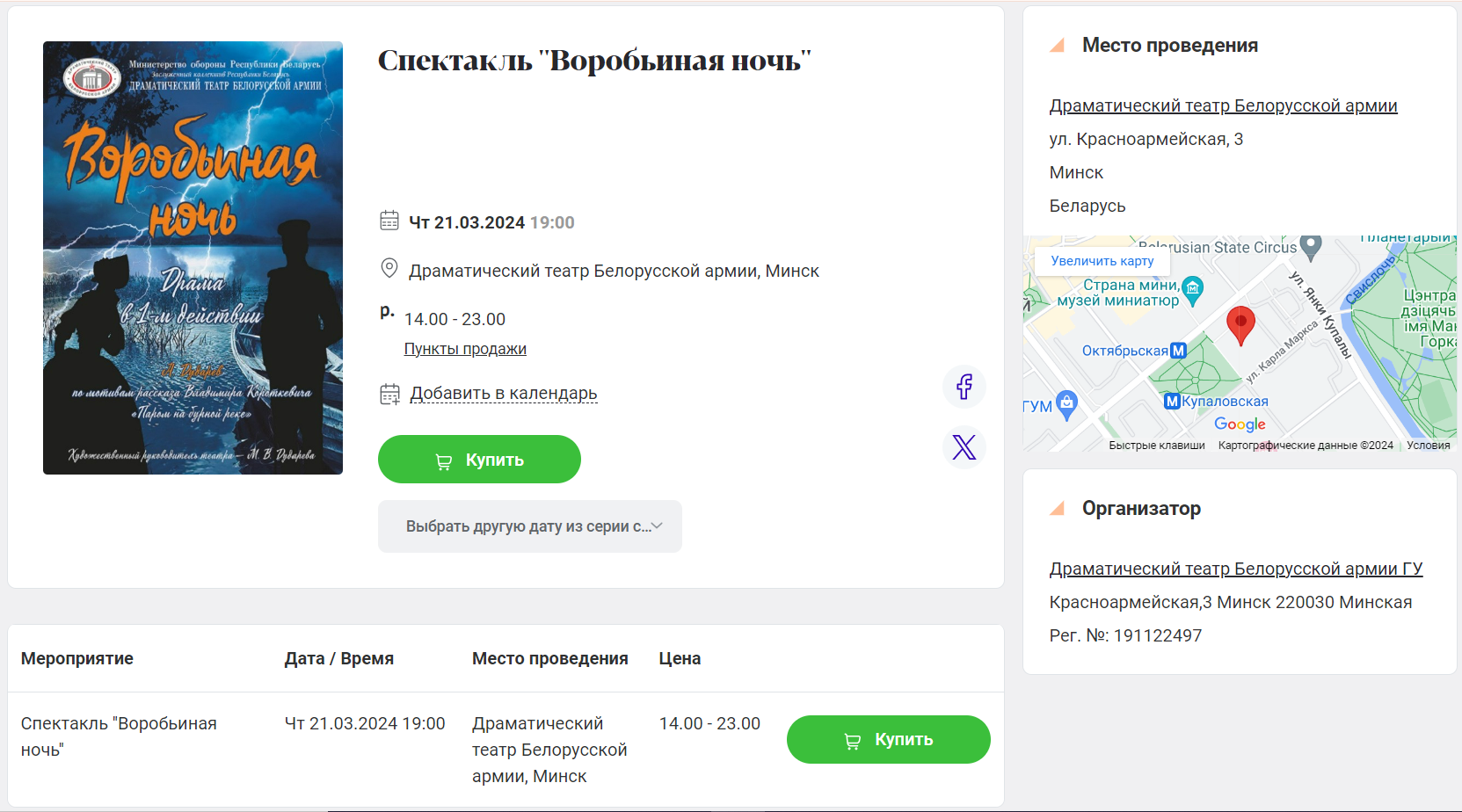


Рисунок 1.10 – Страница мероприятия веб-сайта Квiткi.by

Страница покупки билетов, показанная на рисунке 1.11, имеет подробное описание, дублирующее информацию с веб-страницы мероприятия, а также полную информацию по билетам: цена, количество оставшихся билетов. Предварительный просмотр корзины включает в себя список всех билетов с подробной информацией, итоговая сумма и количество билетов. Присутствует удаление отдельного билета и очистка всей корзины.

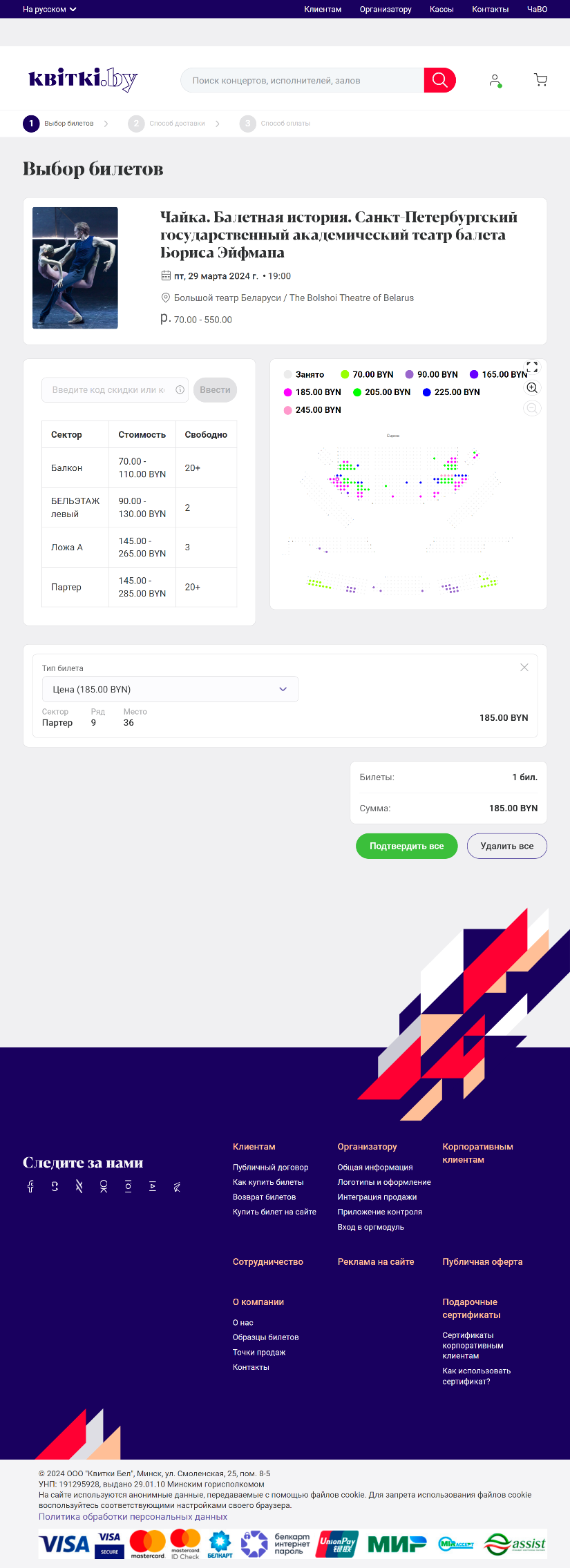


Рисунок 1.11 – Страница выбора билетов веб-сайта Квiткi.by

Если выйти из предварительного просмотра корзины, то зайти в неё повторно и посмотреть содержимое можно будет только в двух случаях: пользователь на данный момент авторизован или пользователь не авторизован, но при входе в корзину вводит контактные данные, без этого посмотреть на добавленные билеты нельзя. Далее при оформлении покупки, в соответствии с рисунком 1.12, необходимо ввести способ оплаты, имеется поле для ввода кода подарочного сертификата. Пишется итоговая цена, не прописано общее количество билетов.

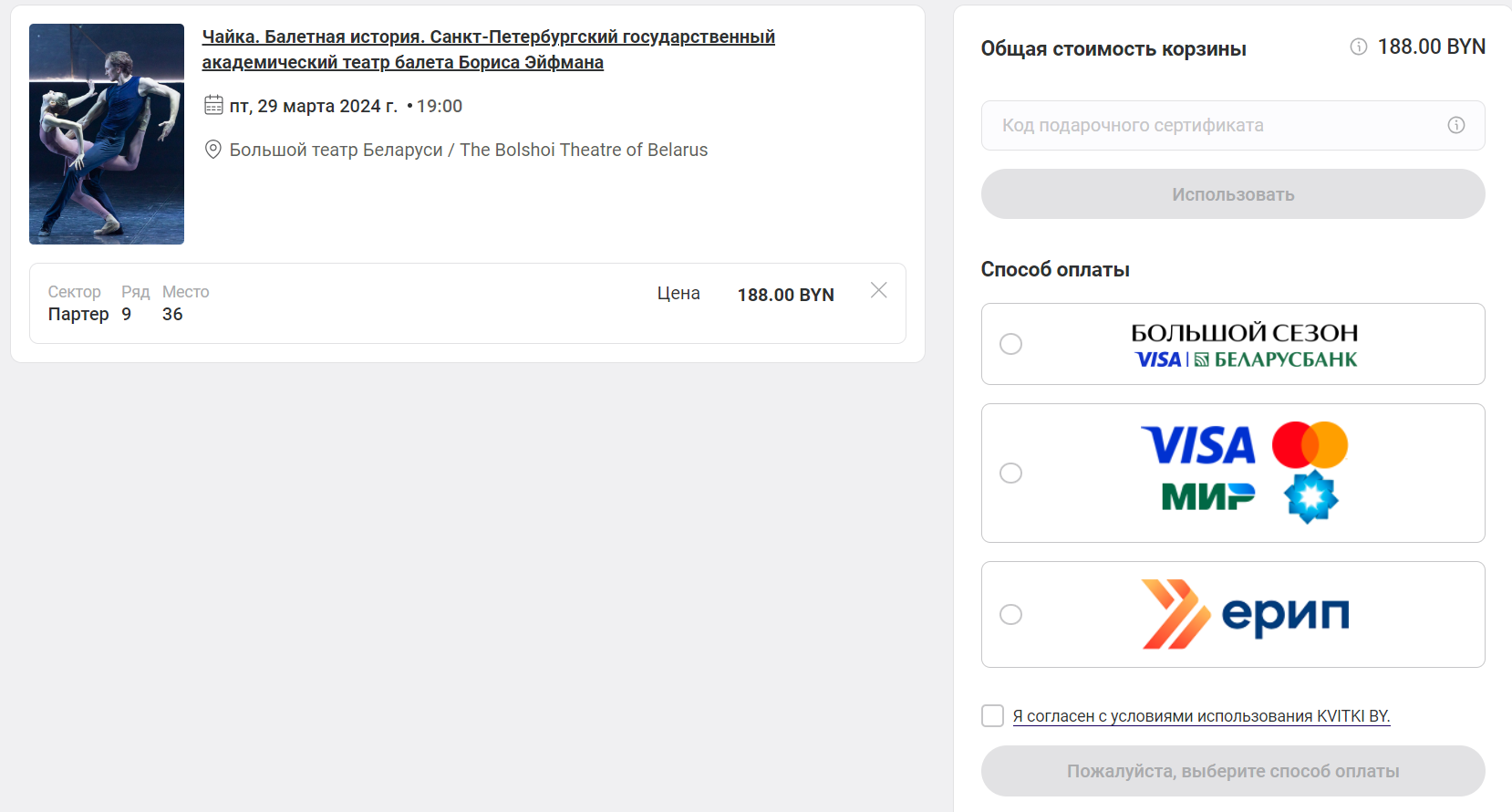


Рисунок 1.12 – Оформление заказа на веб-сайте Квiткi.by

Присутствует подробная инструкция регистрации организаторам. На сайте есть контактная информация, включая социальные сети, однако отсутствует форма для отправки сообщений службе поддержки. Возврат билетов осуществляется только по поручению и от имени организатора мероприятий. Для возврата необходимо написать заявку на почту, указанную в контактах веб-сайта.

### 1.1.3. Аналог «Bezkassira.by».

Последний аналог – Bezkassira.by [3]. Регистрация на веб-сайте осуществляется вводом электронной почты в соответствующее поле, далее на почту приходит письмо с подтверждением регистрации, после чего пользователь попадает в личный кабинет. В кабинете можно поменять информацию об фамилии и имени, номере телефона, подписаться на рассылку, просмотреть информацию о купленных билетах, а также сменить пароль. Личный кабинет представлен на рисунке 1.13.



Рисунок 1.13 – Личный аккаунт веб-сайта Bezkassira.by

Пароль пользователь изначально не задаёт самостоятельно, он приходит на электронную почту отдельным письмом. Если зайти в личный кабинет, выйти из него получится только при помощи выхода из учётной записи, так что каждый раз при необходимости просмотреть информацию в личном кабинете, нужно будет авторизироваться заново. На главной странице пользователи могут найти подборки мероприятий. В соответствии с рисунком 1.14, мероприятия разделены на следующие категории: концерты, кино, спектакли, бизнес, юмор, ИТ и интернет, спорт, для детей, вечеринки, фестивали, выставки, красота и здоровье, образование, квесты, искусство, еда, психология, экскурсия, то есть имеется большое количество разделов.

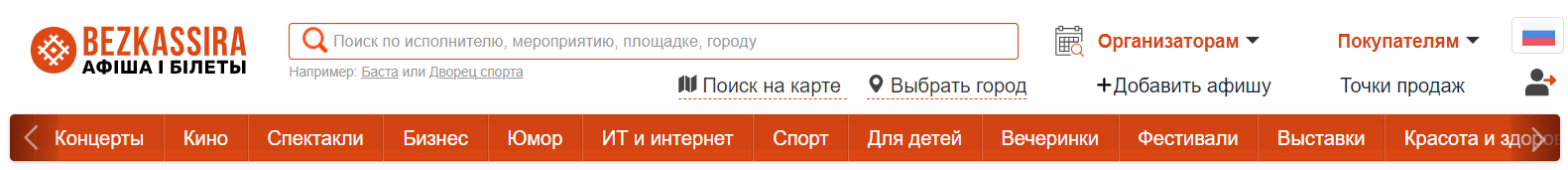


Рисунок 1.14 – Разделение мероприятий на категории веб-сайта Bezkassira.by

Внутри категорий присутствует фильтрация по подкатегориям, месту проведения, дате (нельзя выбрать промежуток даты – только конкретное число), отсутствует сортировка, в шапке веб-сайта располагается поле для поиска. Фильтрация представлена на рисунке 1.15.

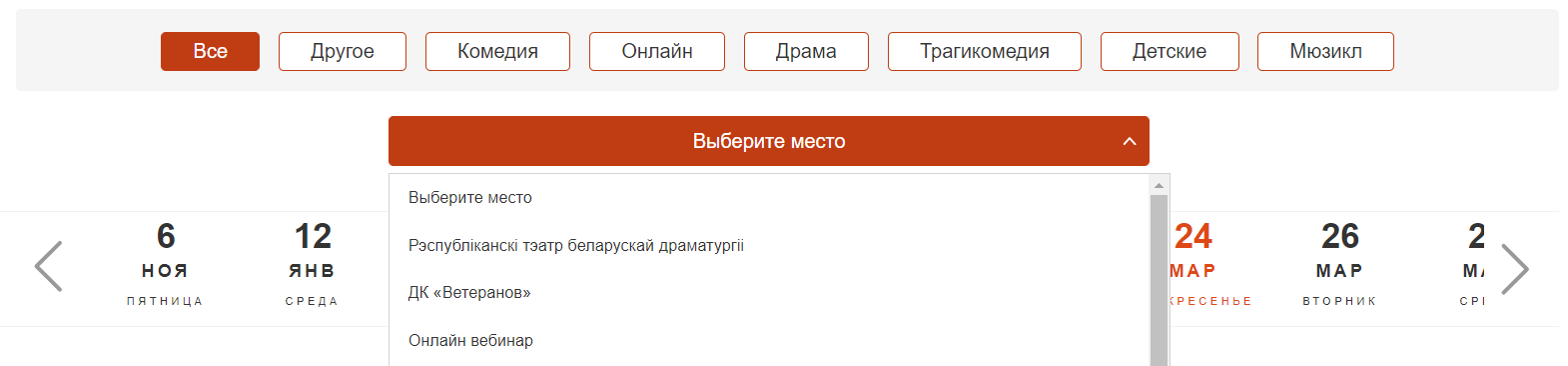


Рисунок 1.15 – Фильтрация и сортировка веб-сайта Bezkassira.by

На веб-сайте есть отдельные страницы с акциями и покупкой сертификатов. Отсутствует удобная регистрация организаторов мероприятий и отзывы. На отдельных веб-страницах по мероприятиям присутствует информация о названии, дате, времени и месте проведения, описание, цена билетов, в соответствии с рисунком 1.16.

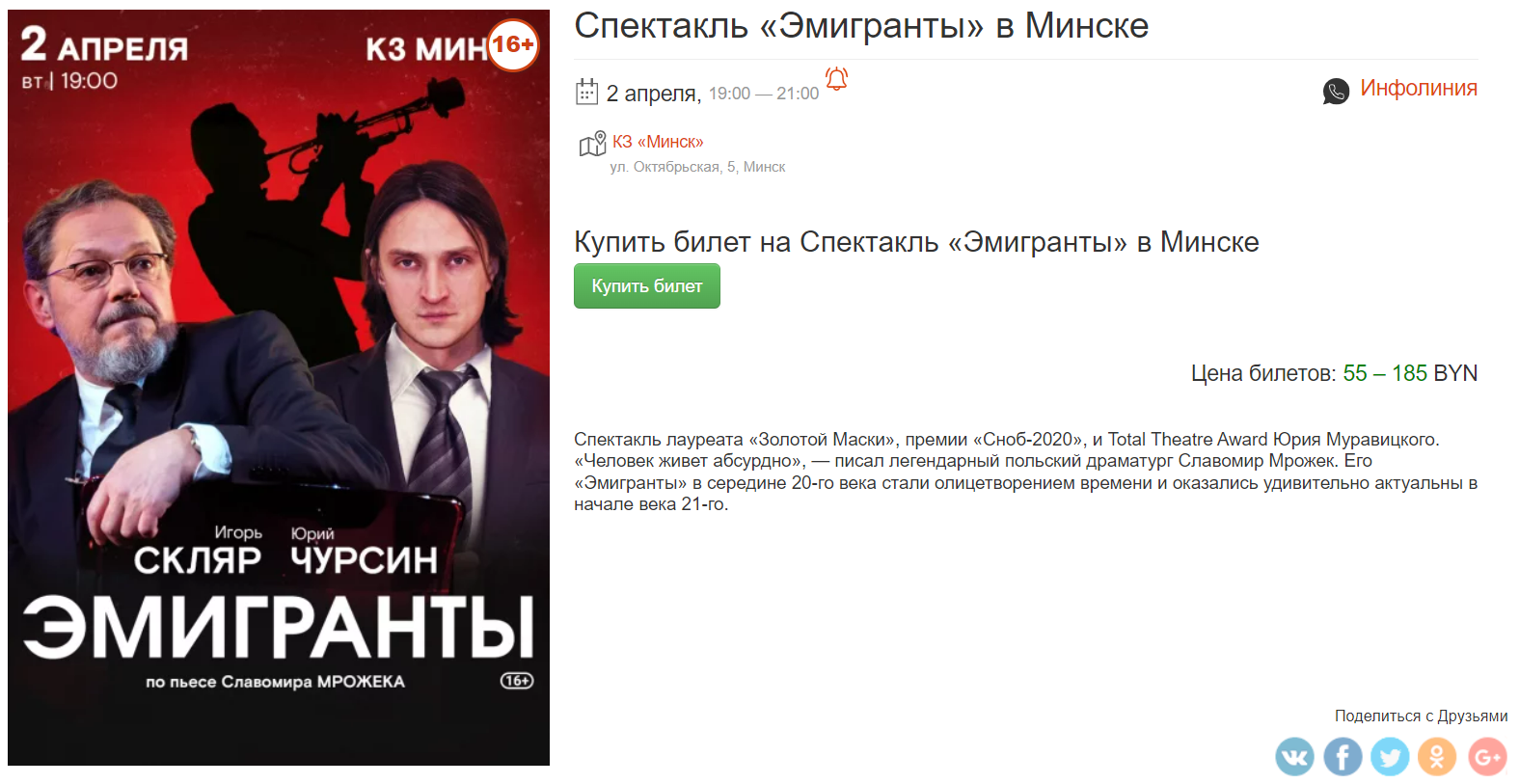


Рисунок 1.16 – Страница мероприятия веб-сайта Bezkassira.by

Страница покупки билетов кратко дублирует информацию о мероприятии, план сцены, каждый билет имеет информацию о стоимости, названии сектора, номере ряда и места. Предварительный просмотр корзины позволяет удалить выбранные места по отдельности, единовременной очистки всей корзины нет, присутствует цена каждого отдельного билета, но отсутствует информация о количестве билетов и итоговой сумме. Страница выбора билетов представлена на рисунке 1.17.

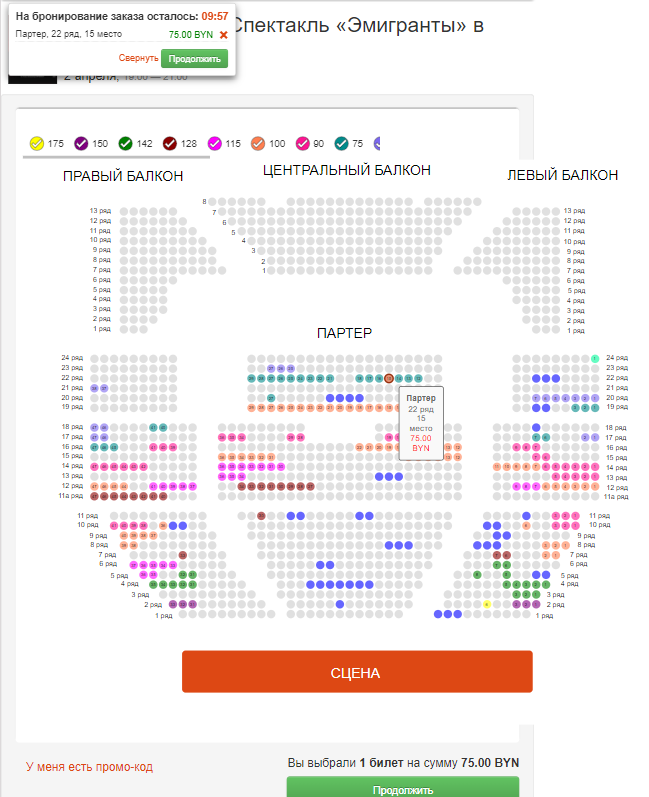


Рисунок 1.17 – Страница выбора билетов веб-сайта Bezkassira.by

При продолжении оформления заказа необходимо указать способ оплаты и ввести данные покупателя, на этом моменте начинает отображаться информация об общей сумме и количестве билетов, что продемонстрированно на рисунке 1.18.

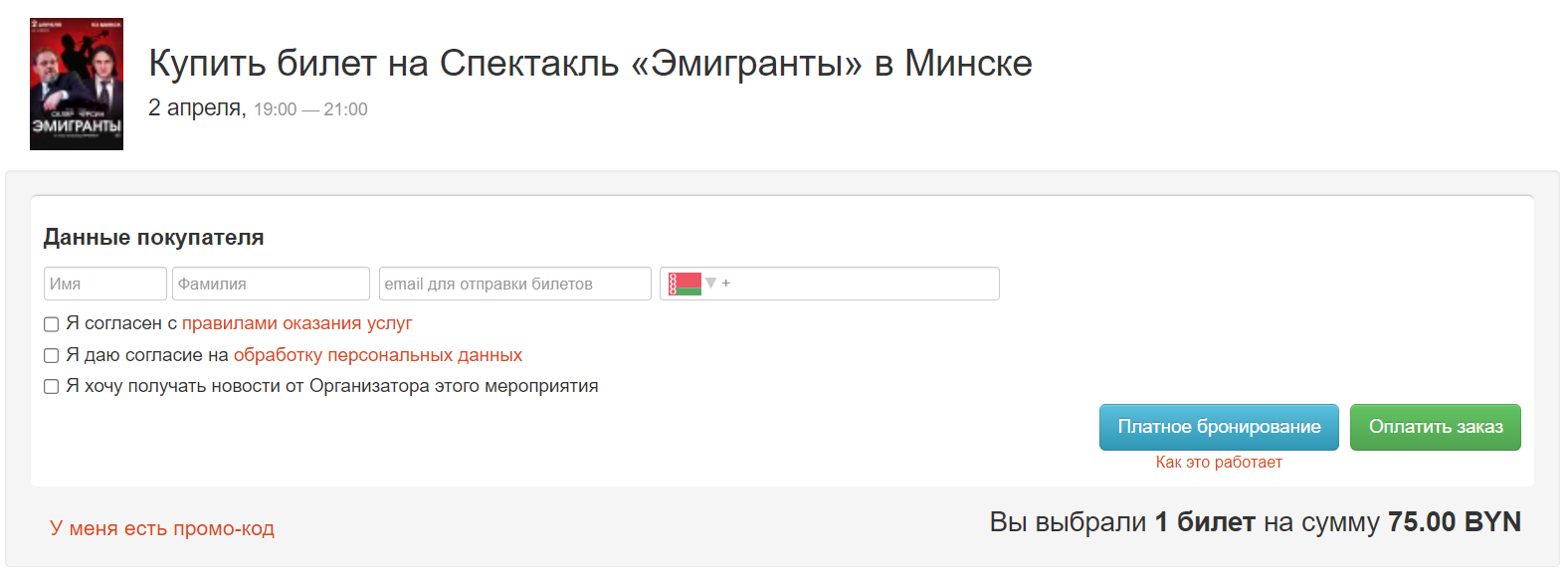


Рисунок 1.18 – Оформление заказа на веб-сайте Bezkassira.by

Купленные билеты в личном кабинете будут отображаются через связку электронных почт учётных записей и почт, указанных при покупках. Имеется форма для заполнения сообщений в службу поддержки. Для возврата денежных средств необходимо отправить заявку на странице заказа соответствующую кнопку, ссылку на страницу заказа можно найти в письме, которое приходит после оформления заказа. Возврат билетов осуществляется только по поручению и от имени организатора мероприятий.

**1.2. Разработка функциональных требований, определение вариантов использования.**

После проведения обзора аналогов и изучения существующих решений в области организации мероприятий, можно сформулировать функциональные требования:

1. Регистрация и авторизация пользователей.

2. При регистрации у пользователя запрашивают информацию об электронной почте, имени, фамилии, номере телефона.

3. Личный кабинет пользователя должен включать историю покупок.

4. Мероприятия должны разделяться на категории. Также должен быть общий каталог всех мероприятий. Основные категории мероприятий должны разбиваться на подкатегории.

5. Должна присутствовать возможность фильтрации по периоду времени, месту проведения, диапазону стоимости.

6. Должна присутствовать возможность сортировки по названию, дате проведения в убывающем и возрастающем порядке.

7. Необходимо наличие поля для поиска мероприятий по введённому названию.

8. Должны присутствовать комментарии к каждому мероприятию, с оценкой по пятибалльной шкале.

9. Для каждого мероприятия должна существовать отдельная страница с подробной информацией.

10. При переходе к процессу выбора билета на мероприятие, пользователь должен иметь возможность выбрать конкретное место. При просмотре каждого места должна иметься информация о ряде и стоимости. Имеется информация о количестве оставшихся билетов.

11. Гости, как и пользователи, имеют возможность просматривать каталог мероприятий, используя фильтрацию, сортировку и поиск, просматривать отдельные веб-страницы мероприятий, а также изучать доступные билеты.

12. Во время добавления билета в корзину необходимо авторизоваться и работать через свой аккаунт.

13. Возврат билетов осуществляется через личный кабинет.

14. Должна присутствовать удобная система регистрации для организаторов.

15. Организаторы также имеют личный кабинет, могут просматривать личный каталог мероприятий и работать с ними через специальный конструктор.

16. Организаторы могут создавать новые мероприятия, изменять или удалять уже существующие. При создании мероприятий должна быть внесена следующая информация: название, дата, время, место проведения, продолжительность мероприятия, описание, стоимость билетов.

17. Менеджеры проводят верификацию организаторов, проверяя их подлинность и соответствие критериям сервиса.

18. Менеджеры могут блокировать и разблокировать организаторов и пользователей из системы.

19. Менеджеры имеют доступ к отчётам о деятельности пользователям (как количество попыток входа в систему, количество возвратов билетов) и о деятельности организаторов (как количество созданных мероприятий, количество удаленных мероприятий).

20. Менеджеры на одном уровне с организаторами могут изменять весь каталог мероприятий (включая удаление мероприятий).

21. Одна из обязанностей менеджеров – отвечать пользователям, гостям и организаторам в системе поддержки.

## 1.3. Вывод.

В результате изучения существующих аналогов и формулирования функциональных требований, были определены ключевые аспекты работы системы. В первую очередь, необходимость регистрации и аутентификации пользователей, предоставление им личных кабинетов с возможностью управления данными и историей покупок. Категоризация мероприятий и удобные фильтры позволят пользователям быстро находить интересующие их события.

Важным элементом функционала является возврат билетов, для которого предусмотрены удобные механизмы через личный кабинет.

Создание удобной системы регистрации для организаторов, а также предоставление им инструментов для управления мероприятиями, включая возможность создавать акции, является ключевым функционалом для привлечения организаторов на платформу.

Менеджеры играют важную роль в поддержании работы системы, они проводят верификацию организаторов, контролируют их деятельность, а также обеспечивают обратную связь с пользователями и организаторами через систему поддержки.

Выявленные функциональные требования позволят создать эффективную и удобную базу данных, обеспечивающую управление мероприятиями и удовлетворяющую потребности как пользователей, так и организаторов.

# 2. Разработка архитектуры проекта.

## 2.1. Диаграммы UML, взаимосвязь всех компонентов.

На рисунке 2.1 представлены диаграммы вариантов использования для гостя, пользователя, организатора и менеджера, которые описывают, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. Актер (фигура человека с именем) представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Отношения ассоциации (сплошная линия) специфицируют семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Отношение включения (пунктирная линия со стрелкой и стереотипом include) устанавливается только между двумя вариантами и указывает на тот факт, что некоторое заданное поведение для одного варианта использования включается в качестве составного фрагмента последовательность поведения другого. Отношение расширения (пунктирная линия со стрелкой и стереотипом extend) определяет взаимосвязь базового варианта использования с некоторым другим вариантом использования, функциональное поведение которого задействуется базовым не всегда, а только при выполнении некоторых дополнительных условий.

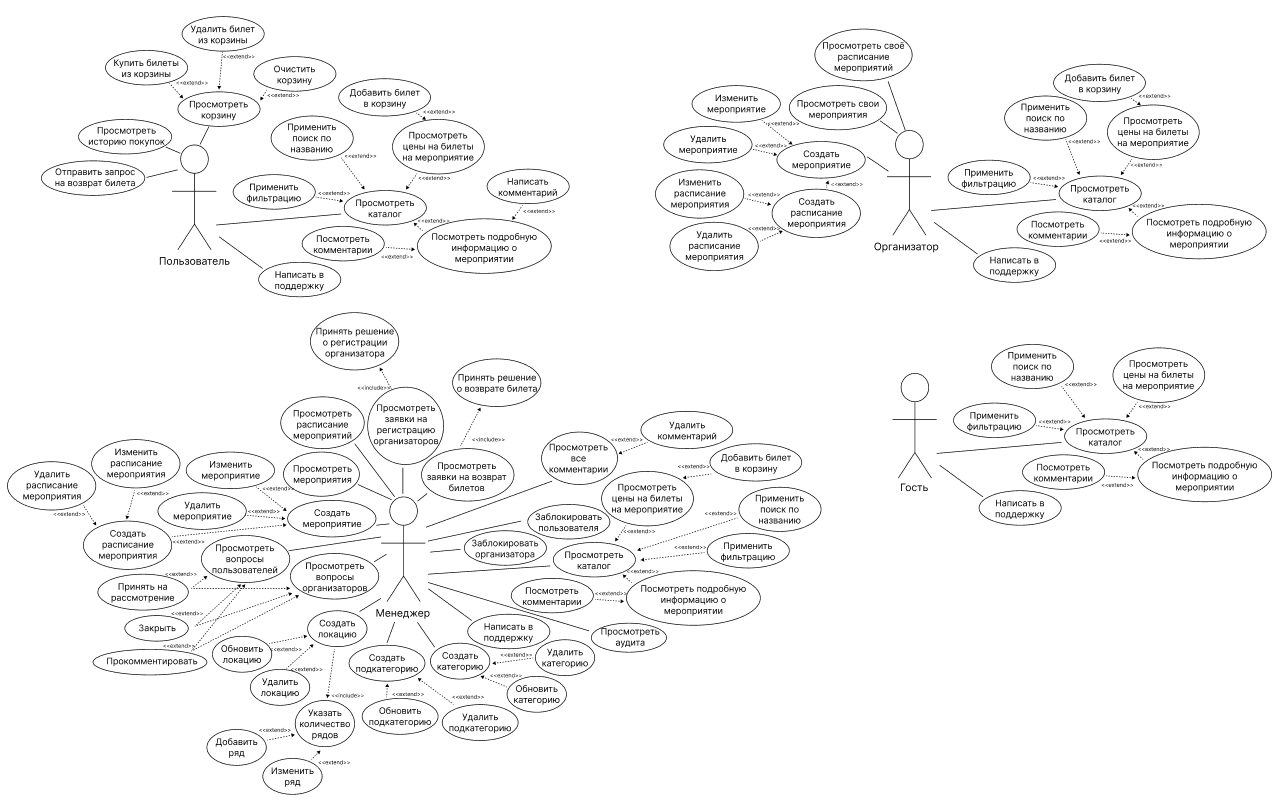


Рисунок 2.1 – Диаграммы вариантов использования для гостя, пользователя, организатора и менеджера

Логическая схема базы данных представлена на рисунке 2.5.



Рисунок 2.5 – Логическая схема базы данных

В логической схеме базы данных продемонстрированы все имеющиеся таблицы и связи один ко многим. В этом типе связей несколько строк из дочерний таблицы зависят от одной строки в родительской таблице. Также используются обозначения PK и FK, что обозначает первичный ключ и внешний ключ соответственно. Первичный ключ уникально идентифицирует строку в таблице. Внешние ключи позволяют установить связи между таблицами. Внешний ключ устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы.

## 2.2. **Описание информационных объектов и ограничений целостности.**

Для построения базы данных сервиса продажи билетов на мероприятия крайне важно определить информационные объекты и гарантировать целостность данных. Ниже представлены основные таблицы, в которых хранится информация о пользователях, событиях, местах проведения и билетах, а также связанные с ними ограничения целостности.

### 2.2.1. Таблица Users.

Таблица Users содержит информацию о зарегистрированных пользователях сервиса, включая их личные данные. Поля таблицы:

1. UserID – уникальный идентификатор пользователя. Тип данных number, первичный ключ.

2. Login – логин пользователя для аутентификации. Тип данных nvarchar2, запрет значений null, уникальное значение.

3. Password – пароль пользователя. Тип данных nvarchar2, запрет значений null.

4. FirstName – имя пользователя. Тип данных nvarchar2, запрет значений null.

5. LastName – фамилия пользователя. Тип данных nvarchar2, запрет значений null.

6. Email – адрес электронной почты пользователя. Тип данных nvarchar2, запрет значений null, уникальное значение.

7. Phone – номер телефона пользователя. Тип данных nvarchar2, запрет значений null, уникальное значение.

8. RoleID – уникальный идентификатор роли. Тип данных number, внешний ключ, запрет значения null.

### 2.2.2. Таблица Events.

Таблица Events содержит информацию о мероприятиях. Поля таблицы:

1. EventID – уникальный идентификатор мероприятия. Тип данных number, первичный ключ.

2. EventName – название мероприятия. Тип данных nvarchar2, запрет значений null, уникальное значение.

3. LocationID – место проведения мероприятия. Тип данных number, запрет значений null, внешний ключ.

4. EventDuration – продолжительность мероприятия. Тип данных interval day to second, запрет значений null.

5. SubcategoryID – подкатегория мероприятия. Тип данных number, запрет значений null, внешний ключ.

6. OrganizerID – организатор мероприятия. Тип данных number, запрет значений null, внешний ключ.

7. Description – описание мероприятия. Тип данных nvarchar2.

### 2.2.3. Таблица Locations.

Таблица Locations содержит информацию о местах проведения мероприятий, доступных для использования организаторами. Поля таблицы:

1. LocationID – уникальный идентификатор места проведения мероприятия. Тип данных number, первичный ключ.

2. LocationName – имя места проведения мероприятия. Тип данных nvarchar2, запрет значений null, уникальное значение.

3. NumberOfSectors – количество рядов в данной локации. Тип данных number, запрет значений null.

### 2.2.4. Таблица SectorRows.

Таблица SectorRows содержит информацию о рядах локации. Поля таблицы:

1. SectorRowID – уникальный идентификатор ряда. Тип данных number, первичный ключ.

2. SectorRow– номер ряда. Тип данных number, запрет значений null.

3. LocationID – уникальный идентификатор локации, в которой расположен ряд. Тип данных number, запрет значений null, внешний ключ.

4. NumberOfSeats – количество сидений в ряду. Тип данных number, запрет значений null.

5. CostFactor – множитель цены данного ряда. Тип данных number, запрет значений null.

### 2.2.5. Таблица Categories.

Таблица Categories содержит список доступных категорий для добавления мероприятий организаторами. Поля таблицы:

1. CategoryID – уникальный идентификатор категории мероприятия. Тип данных number, первичный ключ.

2. CategoryName – название категории мероприятия. Тип данных nvarchar2, запрет значений null, уникальное значение.

### 2.2.6. Таблица Subcategories.

Таблица Subcategories содержит список доступных подкатегорий для добавления мероприятий организаторами. Поля таблицы:

1. SubcategoryID – уникальный идентификатор подкатегории мероприятия. Тип данных number, первичный ключ.

2. SubcategoryName – название подкатегории мероприятия. Тип данных nvarchar2, запрет значений null.

3. CategoryID – уникальный идентификатор категории мероприятия, к которому относится подкатегория. Тип данных number, запрет значений null, внешний ключ.

### 2.2.7. Таблица Tickets.

Таблица Tickets содержит список всех билетов на мероприятия. Поля таблицы:

1. TicketID – уникальный идентификатор билета. Тип данных number, первичный ключ.

2. EventScheduleID – уникальный идентификатор даты проведения мероприятия, на которую покупается билет. Тип данных number, запрет значений null, внешний ключ.

3. Status – статус билета. Тип данных nvarchar2, запрет значений null, проверка значения (в продаже, забронирован, куплен).

4. Price – цена билета. Тип данных number, запрет значений null.

5. SectorRowID – ряд билета. Тип данных number, запрет значений null, внешний ключ.

6. PlaceInRow – место в ряду. Тип данных number, запрет значений null.

### 2.2.8. Таблица Sales.

Таблица Sales содержит список всех покупок билетов на мероприятия. Поля таблицы:

1. SaleID – уникальный идентификатор покупки. Тип данных number, первичный ключ.

2. UserID – уникальный идентификатор пользователя, совершившего покупку. Тип данных number, запрет значений null, внешний ключ.

3. TicketID – уникальный идентификатор купленного билета. Тип данных number, запрет значений null, внешний ключ.

4. SaleDate – дата продажи билета. Тип данных date, запрет значений null.

5. Status – статус продажи. Тип данных nvarchar2, запрет значений null, проверка значения на (действительна, возврат).

### 2.2.9. Таблица Organizers.

Таблица Organizers содержит информацию о всех зарегистрированных организаторов мероприятий. Поля таблицы:

1. OrganizerID – уникальный идентификатор организатора. Тип данных number, первичный ключ.

2. CompanyName – название компании организатора. Тип данных nvarchar2, уникальное значение, запрет значения null.

3. FirstName – имя организатора. Тип данных nvarchar2, запрет значения null.

4. LastName – фамилия организатора. Тип данных nvarchar2, запрет значения null.

5. Email – адрес электронной почты организатора. Тип данных nvarchar2, уникальное значение, запрет значения null.

6. Phone – номер телефона организатора. Тип данных nvarchar2, уникальное значение, запрет значения null.

7. Password – пароль от аккаунта организатора. Тип данных nvarchar2, запрет значения null.

8. RoleID – уникальный идентификатор роли. Тип данных nvarchar2, внешний ключ, запрет значения null.

### 2.2.10. Таблица Comments.

Таблица Comments содержит информацию о всех комментариях к мероприятиям. Поля таблицы:

1. CommentID – уникальный идентификатор комментария. Тип данных number, первичный ключ.

2. UserID – уникальный идентификатор пользователя, оставившего комментарий. Тип данных number, запрет значения null, внешний ключ.

3. EventID – уникальный идентификатор мероприятия, на которое написан комментарий. Тип данных number, запрет значения null, внешний ключ.

4. CommentText – текст комментария. Тип данных nvarchar2, запрет значения null.

5. CommentDate – дата комментария. Тип данных date, запрет значения null.

6. FivePointRating – оценка по пятибалльной шкале. Тип данных number, запрет значения null, проверка, что оценка выставлена в промежутке от 0 до 5.

### 2.2.11. Таблица Managers.

Таблица Managers содержит информацию о всех работающих менеджерах. Поля таблицы:

1. ManagerID – уникальный идентификатор менеджера. Тип данных number, первичный ключ.

2. FirstName – имя менеджера. Тип данных nvarchar2, запрет значения null,

3. LastName – фамилия менеджера. Тип данных nvarchar2, запрет значения null,

4. Email – адрес электронной почты менеджера. Тип данных nvarchar2, запрет значений null, уникальное значение.

5. Phone – номер телефона менеджера. Тип данных nvarchar2, запрет значений null, уникальное значение.

6. Password – пароль от аккаунта менеджера. Тип данных nvarchar2, запрет значений null.

7. DateOfEmployment – дата начала работы менеджера. Тип данных date, запрет значений null.

8. RoleID – уникальный идентификатор роли. Тип данных number, внешний ключ, запрет значения null.

9. Login – логин менеджера. Тип данных nvarchar2, уникальное значение, запрет значений null.

### 2.2.12. Таблица UserQuestions.

Таблица Questions включает информацию о запросах пользователей в службу поддержки. Поля таблицы:

1. QuestionID – уникальный идентификатор запроса. Тип данных number, первичный ключ.

2. UserID – уникальный идентификатор пользователя, который отправил запрос. Тип данных number, запрет значений null, внешний ключ.

3. QuestionText – текст запроса. Тип данных nvarchar2, запрет значений null.

4. QuestionDate – дата отправки запроса. Тип данных date, запрет значений null.

5. AnswerText – текст ответа на запрос. Тип данных nvarchar2.

6. Status – статус запроса. Тип данных nvarchar2, запрет значений null, проверка значения на (не просмотрен, в обработке, закрыт).

7. ManagerID – уникальный идентификатор менеджера, отвечающего за запрос. Тип данных number, запрет значений null, внешний ключ.

### 2.2.13. Таблица OrganizerQuestions.

Таблица Questions включает информацию о всех запросах в службу поддержки. Поля таблицы:

1. QuestionID – уникальный идентификатор запроса. Тип данных number, первичный ключ.

3. OrganizerID – уникальный идентификатор организатора, который отправил запрос. Тип данных number, запрет значений null, внешний ключ.

3. QuestionText – текст запроса. Тип данных nvarchar2, запрет значений null.

4. QuestionDate – дата отправки запроса. Тип данных date, запрет значений null.

5. AnswerText – текст ответа на запрос. Тип данных nvarchar2.

6. Status – статус запроса. Тип данных nvarchar2, запрет значений null, проверка значения на (не просмотрен, в обработке, закрыт).

7. ManagerID – уникальный идентификатор менеджера, отвечающего за запрос. Тип данных number, запрет значений null, внешний ключ.

### 2.2.14. Таблица UserBlocks.

Таблица Blocks включает информацию о блокировках пользователей. Поля таблицы:

1. BlockID – уникальный идентификатор блокировки. Тип данных number, первичный ключ.

2. UserID – уникальный идентификатор заблокированного пользователя. Тип данных number, запрет значений null, внешний ключ.

4. ManagerID – уникальный идентификатор менеджера, создавшего блокировку. Тип данных number, внешний ключ, запрет значений null.

5. Reason – причина блокировки. Тип данных nvarchar2, запрет значений null.

6. EndDate – дата окончания блокировки. Тип данных date.

### 2.2.15. Таблица OrganizerBlocks.

Таблица Blocks включает информацию о блокировках организаторов. Поля таблицы:

1. BlockID – уникальный идентификатор блокировки. Тип данных number, первичный ключ.

3. OrganizerID – уникальный идентификатор заблокированного организатора. Тип данных number, запрет значений null, внешний ключ.

4. ManagerID – уникальный идентификатор менеджера, создавшего блокировку. Тип данных number, внешний ключ, запрет значений null.

5. Reason – причина блокировки. Тип данных nvarchar2, запрет значений null.

6. EndDate – дата окончания блокировки. Тип данных date.

### 2.2.16. Таблица Roles.

Таблица Roles включает информацию о существующих ролях участников. Поля таблицы:

1. RoleID – уникальный идентификатор роли. Тип данных number, первичный ключ.

2. RoleName – название роли. Тип данных number, запрет значений null, проверка значения на (user, manager, organizer).

### 2.2.17. Таблица ShoppingCart.

Таблица ShoppingCart включает информацию о билетах, которые пользователь добавил в корзину для покупки. Поля таблицы:

1. UserID – уникальный идентификатор пользователя, который добавил билеты в корзину. Тип данных number, внешний ключ, запрет значения null.

2. TicketID – уникальный идентификатор билета, добавленного в корзину. Тип данных number, внешний ключ, запрет значения null.

### 2.2.18. Таблица CheckingOrganizers.

Таблица CheckingOrganizers включает информацию об организаторах, которые ещё не прошли проверку модераторов. Поля таблицы:

1. OrganizerID – уникальный идентификатор организатора. Тип данных number, первичный ключ.

2. CompanyName – название компании организатора. Тип данных nvarchar2, уникальное значение, запрет значения null.

3. FirstName – имя организатора. Тип данных nvarchar2, запрет значения null.

4. LastName – фамилия организатора. Тип данных nvarchar2, запрет значения null.

5. Email – адрес электронной почты организатора. Тип данных nvarchar2, уникальное значение, запрет значения null.

6. Phone – номер телефона организатора. Тип данных nvarchar2, уникальное значение, запрет значения null.

7. Password – пароль от аккаунта организатора. Тип данных nvarchar2, запрет значения null.

8. RoleID – уникальный идентификатор роли. Тип данных nvarchar2, внешний ключ, запрет значения null.

### 2.2.19. Таблица EventsSchedule.

Таблица Events содержит информацию о расписании проведении мероприятий. Поля таблицы:

1. EventScheduleID – уникальный идентификатор времени проведения мероприятия. Тип данных number, первичный ключ.

2. EventDate – дата проведения мероприятия. Тип данных timestamp, запрет значений null.

3. EventID – уникальный идентификатор мероприятия. Тип данных number, запрет значений null, внешний ключ.

### 2.2.20. Таблица TicketRefund.

Таблица TicketRefind содержит информацию о билетах, которые пользователи желают вернуть, но они ещё не прошли подтверждение администратора. Поля таблицы:

1. SaleID – уникальный идентификатор покупки билета, требующего возврат. Тип данных number, первичный ключ.

2. UserID – уникальный идентификатор пользователя, купившего билет. Тип данных number, запрет значений null, внешний ключ.

3. Message – текст сообщения с просьбой о возврате. Тип данных nvarchar2, запрет значений null.

4. RefindDate – дата отправки сообщения с просьбой о возврате билета. Тип данных date, запрет значений null.

## 2.3. Вывод.

В данном разделе была представлена детальная схема базы данных для системы по продаже билетов на мероприятия. В процессе разработки были определены и расписаны все необходимые таблицы, их поля и ограничения целостности. Также была представлена логическая схема базы данных для наглядного представления структуры данных.

Проектирование базы данных включало в себя таблицы для хранения информации о пользователях, мероприятиях, билетах, акциях, комментариях и других сущностях, необходимых для функционирования системы. Каждая таблица была описана с указанием её структуры и ограничений, таких как первичные и внешние ключи, а также ограничения на значения полей.

В разделе также была представлена концепция создания пользователей и определены четыре типа ролей с соответствующими привилегиями. Это позволяет управлять доступом к функционалу системы и обеспечить безопасность данных.

В результате была создана основа для дальнейшей реализации системы по продаже билетов на мероприятия, определив структуру базы данных и права доступа пользователей.

# 3. Разработка модели базы данных.

## 3.1. Создание необходимых объектов.

### 3.1.1. Таблицы.

Таблица — совокупность связанных данных, хранящихся в структурированном виде в базе данных. Таблица состоит из набора столбцов, каждый из которых имеет определенный тип данных, определяющий тип информации, которая может быть сохранена в этом столбце. База данных для сервиса по продаже билетов на мероприятия состоит из 20 таблиц, описание которых приведено в главе 2.2.

Пример создания таблицы продемонстрирован в листинге 3.1.

CREATE TABLE Events (

EventID NUMBER PRIMARY KEY,

EventName NVARCHAR2(50) NOT NULL UNIQUE,

LocationID NUMBER NOT NULL,

EventDuration INTERVAL DAY TO SECOND NOT NULL,

SubcategoryID NUMBER NOT NULL,

OrganizerID NUMBER NOT NULL,

Description NVARCHAR2(400),

Cost NUMBER NOT NULL,

CONSTRAINT fk\_LocationForEvents FOREIGN KEY (LocationID) REFERENCES Locations(LocationID),

CONSTRAINT fk\_SubcategoryForEvents FOREIGN KEY (SubcategoryID) REFERENCES Subcategories(SubcategoryID),

CONSTRAINT fk\_OrganizerForEvents FOREIGN KEY (OrganizerID) REFERENCES Organizers(OrganizerID)

);

Листинг 3.1 – Создание таблицы мероприятий

Скрипт создает таблицу «Events» для хранения информации о мероприятиях. Каждое событие имеет уникальный идентификатор (EventID), название (EventName), место проведения (LocationID), продолжительность (EventDuration), подкатегорию (SubcategoryID), организатора (OrganizerID), описание (Description) и стоимость (Cost). Скрипт также устанавливает внешние ключи, связывающие таблицу «Events» с таблицами «Locations», «Subcategories» и «Organizers», обеспечивая целостность данных и гарантируя, что каждое событие связано с существующим местоположением, подкатегорией и организатором.

### 3.1.2. Процедуры.

Процедура – именованный модуль, который выполняет одно или несколько выражений и может принимать или возвращать значения через список параметров.

Процедуры, разработанные в рамках курсового проекта:

* AcceptOranizer. Одобрение менеджером заявки организатора на регистрацию. Соответствующая запись добавляется в таблицу организаторов и создаётся новый пользователь в базе данных.
* AcceptOrganizerQuestion. Принятие менеджером вопроса от организатора. Происходит перевод статуса вопроса в «В процессе».
* AcceptRefend. Подтверждение менеджером возврата купленного пользователем билета. Происходит возврат билета в продажу и перевод статуса покупки пользователя на «Возврат».
* AcceptUserQuestion. Принятие менеджером вопроса от пользователя. Происходит перевод статуса вопроса в «В процессе».
* AddCategory. Добавление новой категории.
* AddLocation. Добавление нового места проведения мероприятий.
* AddRow. Добавление нового ряда.
* AddSubcategory. Добавление подкатегории.
* AddTicketToShoppingCart. Добавление билета в корзину пользователем. Статус билета меняется на «Забронирован».
* AskForRefund. Отправка пользователем запроса на возврат билета.
* AuditEventDelete. Вывод аудита удаления мероприятий организаторами.
* AuditTicketRefend. Вывод аудита отправки пользователями запросов на возврат билетов.
* Authorization. Авторизация, используется для входа в систему для всех ролей. Один из её входных параметров – индекс, указывающий на роль входящего. При успешной авторизации создаётся новое подключение, в строку которого заносятся указанный логин и пароль. При авторизации пользователя и организатора проводится проверка на отсутствие блокировок.
* BlockOrganizer. Блокировка организаторов менеджерами. Присутствует запрет на наложение нескольких блокировок за раз.
* BlockUser. Блокировка пользователей менеджерами. Присутствует запрет на наложение нескольких блокировок за раз.
* BuyTicket. Покупка билетов. Производится покупка всех находящихся в корзине билетов, меняется их статус, записываются покупки на пользователя.
* ClearShoppingCart. Очистка корзины пользователя. Билетам возвращают статус «В продаже».
* CloseOrganizerQuestion. Закрытие менеджером вопроса организатора. После закрытия вопроса, к его ответу нельзя добавлять правки.
* CloseUserQuestion. Закрытие менеджером вопроса пользователя. После закрытия вопроса, к его ответу нельзя добавлять правки.
* CommentOrganizerQuestion. Добавление комментария к вопросу организатора. Комментарии можно добавлять без ограничений, к каждому комментарию приписывается время ответа. Если статус вопроса не был изменён на «В процессе», то это происходит при первом добавлении комментария.
* CommentUserQuestion. Добавление комментария к вопросу пользователя. Комментарии можно добавлять без ограничений, к каждому комментарию приписывается время ответа. Если статус вопроса не был изменён на «В процессе», то это происходит при первом добавлении комментария.
* CreateComment. Добавление комментария к событию. Повторно писать комментарии к одинаковым события не запрещается. Оставлять комментарии могут только те пользователи, что уже посетили мероприятие.
* CreateEvent. Создание события с проверкой переданных данных.
* CreateEventAfterHours. Вывод аудита создания мероприятий организаторами в нерабочее время (22:00-08:00).
* CreateEventSchedule. Создание даты проведения мероприятия. Имеется проверка, что мероприятие должно проводится не позже, чем через год с момента создания, и не раньше, чем следующий день с момента создания. Имеется проверка, что локация, в которой проводится расписание, не должна быть занята в момент проведения (учитываются даты начала других мероприятий и их продолжительность).
* CreateOrganizerQuestion. Создание вопроса от лица организатора.
* CreateUserQuestion. Создание вопроса от лица пользователя.
* DeleteCategory. Удаление категории.
* DeleteComment. Удаление комментария к мероприятию.
* DeleteEvent. Удаление мероприятия организатором.
* DeleteEventByManager. Удаление мероприятия менеджером, не требует подтверждения владения мероприятием.
* DeleteEventSchedule. Удаление даты проведения мероприятия организатором.
* DeleteEventScheduleByManager. Удаление даты проведения мероприятия менеджером, не требует подтверждения владения мероприятием.
* DeleteLocation. Удаление места проведения мероприятий.
* DeleteRow. Удаление ряда. Так как ряды идут в строгом возрастании, то при удалении ряда, номер остальных сдвигается вниз.
* DeleteSubcategory. Удаление подкатегории.
* DeleteTicketFromShoppingCart. Удаление одного билета из корзины. У билета меняется статус на «В продаже».
* DenyOrganizer. Отклонение запроса организатора на регистрацию.
* DenyRefend. Отклонение запроса пользователя на возврат билета.
* FailerLoginAttempts. Вывод аудита неудачных попыток входа в систему.
* FindManagerWithLeastQuestions. Вспомогательная процедура, используемая при создании вопросов от пользователей и организаторов. При создании нового вопроса, система ищет менеджера с наименьшим количеством вопросов под его ответственностью и записывает вопрос на него.
* FindTicketPrice. Нахождение стоимости билета по переданному уникальному идентификатору времени проведения события, номеру ряда и места в ряду.
* GetPlacesInRowByEventNameAndSectorRow. Получение списка доступных для покупки мест в ряду по переданному уникальному идентификатору времени проведения события, номеру ряда.
* GetPurchaseHistory. Получение истории покупок конкретного пользователя.
* GetShoppingCartInfo. Получение информации о количестве билетов в корзине пользователя и их общей стоимости.
* GetSubcategoriesByCategoryName. Получение списка подкатегорий, относящихся к переданной категории.
* GetTicketPricesByEventName. Вывод информации о билетах мероприятия. Пользователь получает список следующего формата: номер ряда, количество доступных к продаже мест в этом ряду и цена (цена распределяется по рядам).
* ManagerRegistration. Регистрация менеджера. Если введённые данные проходят проверку на уникальность и соответствие регулярным выражениям, соответствующая запись добавляется в таблицу менеджеров и создаётся новый пользователь в базе данных.
* OrganizerRegistration. Регистрация менеджера. Если введённые данные проходят проверку на уникальность и соответствие регулярным выражениям, менеджерам отправляется запрос на регистрацию организатора.
* SearchingCatalog. Частичный поиск в каталоге мероприятий по названию;
* SelectFromCategories. Вывод всех категорий.
* SelectFromCheckingOrganizers. Вывод списка всех организаторов, проходящих проверку для одобрения регистрации.
* SelectFromLocations. Вывод всех мест проведения мероприятий.
* SelectFromSectorRows. Вывод всех рядов.
* SelectFromSubcategorits. Вывод всех подкатегорий.
* ShowAllComments. Вывод всех существующих комментариев.
* ShowAllEvents. Вывод всех существующих мероприятий.
* ShowAllEventsSchedule. Вывод всего существующего расписания мероприятий.
* ShowCataloge. Вывод всего каталога расписания мероприятий. Каталог фильтруется по периоду времени проведения, месту проведения, категории, подкатегории, диапазону цены. В зависимости от переданных параметров, каталог сортируется по названию или по дате проведения, а также может сортироваться как по возрастанию, так и по убыванию.
* ShowComments. Вывод комментариев для конкретного мероприятия.
* ShowEventByEventScheduleID. Вывод информации о мероприятии по уникальному идентификатору времени проведения мероприятия.
* ShowEventsSchedule. Вывод полного расписания мероприятий конкретного организатора.
* ShowFullScheduleOfEventByEventScheduleID. Вывод полного расписания мероприятия по одной из дат проведения. Необходимо, чтобы из каталога, где размещаются мероприятия по датам проведения, посмотреть полное расписание мероприятия.
* ShowOrganizerBlock. Вывод списка всех блокировок организаторов.
* ShowOrganizerEvent. Вывод полного списка мероприятий конкретного организатора.
* ShowOrganizerQuestion. Вывод всех вопросов, заданных конкретным организатором.
* ShowOrganizerQuestionForManager. Вывод всех вопросов организаторов, относящихся к конкретному менеджеру.
* ShowRefund. Вывод всех запросов на возврат билетов.
* ShowShoppingCart. Вывод корзины конкретного пользователя.
* ShowUserBlock. Вывод списка всех блокировок пользователей.
* ShowUserQuestion. Вывод всех вопросов, заданных конкретным пользователем.
* ShowUserQuestionForManager. Вывод всех вопросов пользователей, относящихся к конкретному менеджеру.
* UpdateCategoty. Обновление названия категории.
* UpdateEvent. Обновление событий организатором. Обновить можно название, продолжительность, подкатегорию, описание, цену.
* UpdateEventByManager. Обновление событий менеджером, не требует подтверждения владения мероприятием.
* UpdateEventsSchedule. Обновление даты проведения мероприятия организатором. Выполняется проверка, что локация на указанное время не занята другим мероприятием.
* UpdateEventsScheduleByManager. Обновление даты проведения мероприятия менеджером, не требует подтверждения владения мероприятием.
* UpdateLocation. Обновление локации. Обновить можно название локации и количество рядов.
* UpdateRow. Обновление ряда. Обновить можно количество сидений в ряду и множитель цены.
* UpdateSubcategory. Обновление подкатегории.
* UserRegistration. Регистрация пользователя. Если введённые данные проходят проверку на уникальность и соответствие регулярным выражениям, соответствующая запись добавляется в таблицу пользователей и создаётся новый пользователь в базе данных.

Пример создания процедуры продемонстрирован в листинге 3.2.

CREATE OR REPLACE PROCEDURE CommentOrganizerQuestion(

QuestionID\_in IN NUMBER,

ManagerLogin\_in IN NVARCHAR2,

AnswerText IN NVARCHAR2,

Message\_out OUT NVARCHAR2

)

AS

ManagerID\_f NUMBER;

MatchesNumber NUMBER;

CurrentStatus NVARCHAR2(15);

CurrentAnswer NVARCHAR2(500);

Ex\_Invalid\_MatchesNumber EXCEPTION;

Ex\_Invalid\_Closed EXCEPTION;

BEGIN

SELECT ManagerID INTO ManagerID\_f FROM TicketVibe\_Managers WHERE Login = ManagerLogin\_in;

SELECT COUNT (\*) INTO MatchesNumber FROM TicketVibe\_OrganizerQuestions WHERE ManagerID = ManagerID\_f AND QuestionID = QuestionID\_in;

IF MatchesNumber = 0 THEN

RAISE Ex\_Invalid\_MatchesNumber;

END IF;

SELECT Status INTO CurrentStatus FROM TicketVibe\_OrganizerQuestions WHERE ManagerID = ManagerID\_f AND QuestionID = QuestionID\_in;

IF CurrentStatus = 'Not Viewed' THEN

UPDATE TicketVibe\_OrganizerQuestions SET Status = 'In processing' WHERE QuestionID = QuestionID\_in;

END IF;

IF CurrentStatus = 'closed' THEN

RAISE Ex\_Invalid\_Closed;

END IF;

SELECT AnswerText INTO CurrentAnswer FROM TicketVibe\_OrganizerQuestions WHERE ManagerID = ManagerID\_f AND QuestionID = QuestionID\_in;

CurrentAnswer := CurrentAnswer || ' ' || TO\_CHAR(SYSDATE, 'DD.MM.YYYY HH24:MM:SS');

CurrentAnswer := CurrentAnswer || ' 'AnswerText;

UPDATE TicketVibe\_OrganizerQuestions SET AnswerText = CurrentAnswer WHERE QuestionID = QuestionID\_in;

COMMIT;

EXCEPTION

WHEN Ex\_Invalid\_MatchesNumber THEN

Message\_out := 'This manager is not responsible for this question';

WHEN Ex\_Invalid\_Closed THEN

Message\_out := 'This question is closed';

WHEN NO\_DATA\_FOUND THEN

Message\_out := 'The question was not found';

WHEN OTHERS THEN

Message\_out := 'Error during commenting';

END;

Листинг 3.2 – Создание процедуры комментирования вопроса организатора менеджером

Продемонстрированная выше процедура проверяет, имеет ли менеджер право отвечать на данный вопрос, а также проверяет статус вопроса (открыт или закрыт). Если вопрос открыт, и менеджер имеет право на ответ, процедура обновляет статус вопроса на «В обработке» (если он был «Не просмотрен») и добавляет комментарий менеджера вместе с датой и временем к существующему тексту ответа. В случае ошибок, таких как несоответствие менеджера вопросу, закрытый статус вопроса или другие ошибки, процедура возвращает соответствующее сообщение об ошибке.

### 3.1.3. Функции.

Функция — это программная конструкция, которая выполняет определенные действия и возвращает одно значение. Функции используются для выполнения вычислений, обработки данных и выполнения других операций. Функция определяется с помощью ключевого слова CREATE FUNCTION и должна обязательно включать возвращаемый тип данных.

Функции и процедуры имеют схожую структуру, но они предназначены для разных целей и имеют некоторые ключевые различия. Функция обязательно возвращает одно значение. Она часто используется для вычислений и может быть вызвана внутри SQL-запросов. Процедура также является программной конструкцией, но ее основная задача — выполнение определенного набора действий. Процедура не обязана возвращать значение, хотя может это делать через выходные параметры.

Функции, разработанные в рамках курсового проекта:

* export\_events\_to\_xml. Экспорт списка событий в XML документ.
* import\_events\_from\_xml. Импорт списка событий из XML документа. Происходит проверка на уникальность импортируемых событий.

Пример создания функции продемонстрирован на листинге 3.3.

CREATE OR REPLACE FUNCTION import\_events\_from\_xml(p\_filename IN VARCHAR2) RETURN VARCHAR2 IS

l\_file\_handle UTL\_FILE.file\_type;

l\_raw RAW(32767);

l\_clob CLOB;

l\_amount BINARY\_INTEGER := 32767;

l\_offset INTEGER := 1;

l\_xml XMLTYPE;

l\_event\_id NUMBER;

l\_event\_name NVARCHAR2(50);

l\_message VARCHAR2(1000);

BEGIN

l\_file\_handle := UTL\_FILE.fopen('XML\_DIRECTORY', p\_filename, 'rb');

WHILE TRUE LOOP

UTL\_FILE.get\_raw(l\_file\_handle, l\_raw, l\_amount);

l\_clob := l\_clob UTL\_RAW.cast\_to\_varchar2(l\_raw);

EXIT WHEN LENGTH(l\_raw) < l\_amount;

END LOOP;

UTL\_FILE.fclose(l\_file\_handle);

l\_xml := XMLTYPE(l\_clob);

FOR r IN (

SELECT extractvalue(value(t), '/Event/EventID') AS EventID,

extractvalue(value(t), '/Event/EventName') AS EventName,

extractvalue(value(t), '/Event/LocationID') AS LocationID,

extractvalue(value(t), '/Event/EventDuration') AS EventDuration,

extractvalue(value(t), '/Event/SubcategoryID') AS SubcategoryID,

extractvalue(value(t), '/Event/OrganizerID') AS OrganizerID,

extractvalue(value(t), '/Event/Description') AS Description,

extractvalue(value(t), '/Event/Cost') AS Cost

FROM TABLE(XMLSEQUENCE(extract(l\_xml, '/Events/Event'))) t

) LOOP

l\_event\_id := r.EventID;

l\_event\_name := r.EventName;

BEGIN

SELECT 1 INTO l\_event\_id FROM TicketVibe\_Events WHERE EventID = r.EventID;

l\_message := l\_message || 'Event with ID ' || r.EventID || ' already exists. ';

CONTINUE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

NULL;

END;

BEGIN

SELECT 1 INTO l\_event\_name FROM TicketVibe\_Events WHERE EventName = r.EventName;

l\_message := l\_message || 'Event with name ' || r.EventName || ' already exists. ';

CONTINUE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

NULL;

END;

INSERT INTO TicketVibe\_Events (EventID, EventName, LocationID, EventDuration, SubcategoryID, OrganizerID, Description, Cost)

VALUES (r.EventID, r.EventName, r.LocationID, r.EventDuration, r.SubcategoryID, r.OrganizerID, r.Description, r.Cost);

END LOOP;

COMMIT;

RETURN l\_message;

END;

Листинг 3.3 – Создание функции импорта списка событий из XML документа

Функция import\_events\_from\_xml предназначена для импорта информации о событиях из XML-файла в таблицу базы данных Events. Для работы ей нужно передать имя файла в качестве входного параметра p\_filename. Перед вставкой каждого события функция проверяет, существует ли уже событие с таким же уникальным идентификатором или именем После завершения импорта всех событий функция фиксирует транзакцию с помощью COMMIT и возвращает строку l\_message с информацией о ходе импорта. Возврат значения является обязательным условием для функций.

### 3.1.4. Триггеры.

Триггеры — это специальные хранимые процедуры, которые выполняются автоматически при определенных событиях, происходящих в базе данных. Эти события могут быть изменениями данных в таблицах (например, вставка, обновление или удаление записей), либо событиями базы данных (например, включение или выключение базы данных).

Триггеры, разработанные в рамках курсового проекта:

* AFTER\_DELETE\_CATEGORY. Триггер, срабатывающий после удаление категории. Удаляются все соответствующие подкатегории.
* AFTER\_DELETE\_EVENT. Триггер, срабатывающий после удаления события. Удаляются все соответствующие записи времени проведения данного мероприятия.
* AFTER\_DELETE\_EVENTSCHEDULE. Триггер, срабатывающий после удаления времени проведения мероприятия. Удаляются все соответствующие билеты на это мероприятие по указанному времени.
* AFTER\_DELETE\_LOCATION. Триггер, срабатывающий после удаления места проведения мероприятий. Удаляются все мероприятия, прописанные на данную локацию и записи рядов.
* AFTER\_DELETE\_SECTORROWS. Триггер, срабатывающий после удаление ряда. Удаляет все билеты, относящиеся к этому ряду.
* AFTER\_DELETE\_SUBCATEGORY. Триггер, срабатывающий после удаление подкатегории. Удаляет все события, относящиеся к этой подкатегории.
* AFTER\_DELETE\_TICKET. Триггер, срабатывающий после удаление билетов. Удаляются все соответствующие продажи и билеты в корзинах пользователей.
* AFTER\_INSERT\_EVENTSCHEDULE. Триггер, срабатывающий после вставки времени проведения мероприятия. В соответствии с выбранным местом проведения, количеством рядов в нём, количеством мест в каждом ряду и множителем цены ряда, генерируются билеты на данное мероприятия.
* AFTER\_INSERT\_LOCATION. Триггер, срабатывающий после вставки места проведения мероприятия. В соответствии с указанным количеством рядов в локации, создаются записи рядов. По умолчанию в каждом ряду создаётся 30 мест с множителем цены равным единице.
* AFTER\_UPDATE\_EVENT. Триггер, срабатывающий после обновления мероприятия. Производится проверка на изменение цены при обновлении, и в случае, если цена изменилась, происходит пересчёт стоимости билетов.
* AFTER\_UPDATE\_LOCATION. Триггер, срабатывающий после обновления места проведения мероприятия. Производится проверка на изменение количества рядов при обновлении, и в случае, если количество изменилось, происходит их пересчёт.
* AFTER\_UPDATE\_SECTORROWS. Триггер, срабатывающий после изменения ряда. Удаляет лишние билеты, если количество сидений стало меньше, или добавляет новые, если количество сидений стало больше.

Пример создания триггера продемонстрирован в листинге 3.4.

CREATE OR REPLACE TRIGGER AFTER\_INSERT\_EVENTSCHEDULE

AFTER INSERT ON TicketVibe\_EventsSchedule

FOR EACH ROW

DECLARE

LocationID\_in NUMBER;

EventID\_in NUMBER;

EventScheduleID\_in NUMBER;

SectorRowID\_f NUMBER;

Price\_in NUMBER;

Price\_f NUMBER;

NumberOfSeats\_f NUMBER;

max\_TicketID NUMBER;

BEGIN

EventID\_in := :NEW.EventID;

EventScheduleID\_in := :NEW.EventScheduleID;

SELECT LocationID INTO LocationID\_in FROM TicketVibe\_Events WHERE EventID = EventID\_in;

FOR row\_rec IN (SELECT sr.SectorRowID, sr.NumberOfSeats, sr.CostFactor

FROM TicketVibe\_SectorRows sr

WHERE sr.LocationID = LocationID\_in

ORDER BY sr.SectorRowID)

LOOP

SELECT NVL(MAX(TicketID), 1000) INTO max\_TicketID FROM TicketVibe\_Tickets;

SectorRowID\_f := row\_rec.SectorRowID;

NumberOfSeats\_f := row\_rec.NumberOfSeats;

SELECT Cost INTO Price\_in FROM TicketVibe\_Events WHERE EventID = EventID\_in;

Price\_f := Price\_in \* row\_rec.CostFactor;

FOR i IN 1..NumberOfSeats\_f LOOP

INSERT INTO TicketVibe\_Tickets (TicketID, EventScheduleID, Status, Price, SectorRowID, PlaceInRow)

VALUES (max\_TicketID + i, EventScheduleID\_in, 'On sale', Price\_f, SectorRowID\_f, i);

END LOOP;

END LOOP;

END;

Листинг 3.4 – Создание триггера, срабатывающего после вставки расписания мероприятия

Триггер AFTER\_INSERT\_EVENTSCHEDULE срабатывает после добавления новой записи в таблицу EventsSchedule. Триггер предназначен для автоматической генерации билетов на мероприятие. Для каждой строки в таблице SectorRows, соответствующей месту проведения мероприятия, триггер создает нужное количество билетов (NumberOfSeats) с уникальными идентификаторами, статусом «В продаже», ценой, рассчитанной на основе стоимости мероприятия и коэффициента стоимости ряда, а также информацией о секторе, ряде и месте в ряду.

### 3.1.5. Синонимы.

Синонимы — это объекты, которые позволяют ссылаться на другие объекты базы данных под альтернативными именами. Это означает, что можно создать синоним для таблицы, представления, процедуры, функции или другого объекта базы данных и обращаться к ним по этому синониму, вместо оригинального имени объекта.

В данной курсовой работе ко всем процедурам были созданы публичные синонимы для упрощения доступа к объектам другой схемы. Публичные синонимы – это синонимы, которые доступны всем пользователям базы данных, независимо от того, в какой схеме они были созданы. Пример создания синонима продемонстрирован в листинге 3.5.

CREATE PUBLIC SYNONYM TicketVibe\_CreateUserQuestion FOR Programmer\_1.CreateUserQuestion;

Листинг 3.5 – Создание синонима на процедуру

Создание публичного синонима TicketVibe\_CreateUserQuestion для процедуры CreateUserQuestion, которая находится в схеме пользователя Programmer\_1, означает, что любой пользователь базы данных сможет вызывать процедуру CreateUserQuestion используя имя TicketVibe\_CreateUserQuestion, без необходимости указывать имя схемы Programmer\_1. Это упрощает доступ к процедуре и делает код более читаемым.

### 3.1.6. Индексы.

Индексы представляют собой объекты, создаваемые для ускорения выполнения запросов к базе данных. Индексы позволяют быстро находить строки в таблице на основе значений одного или нескольких столбцов. Когда индекс создается на столбце или группе столбцов таблицы, база данных создает отображение между значениями в столбце индекса и соответствующими строками в таблице.

Индексы, разработанные в рамках курсового проекта:

* idx\_EventScheduleID. Индекс таблицы Tickets для поиска билетов по уникальным идентификаторам времени проведения мероприятия.
* idx\_EventID. Индекс таблицы EventsSchedule для поиска времени проведения мероприятий по уникальным идентификаторам мероприятий.
* idx\_UserID. Индекс таблицы Sales для поиска покупок по уникальным идентификаторам пользователей.

Пример создания индекса продемонстрирован в листинге 3.6.

CREATE INDEX idx\_EventScheduleID ON TicketVibe\_Tickets(EventScheduleID);

Листинг 3.6 – Создание синонима на процедуру

Создание индекса idx\_EventScheduleID призвано ускорить поиск билетов по уникальному идентификатору расписания мероприятий.

## 3.2. Описание используемой технологии.

Для создания базы данных сервиса по продаже билетов на мероприятия была выбрана технология аудита. Аудит — это процесс отслеживания и регистрации действий пользователей в базе данных, таких как выполнение SQL-операторов, доступ к объектам базы данных и изменение данных. Аудит используется для обеспечения безопасности данных и отслеживания активности пользователей.

Для включения аудита необходимо изменить значение параметра AUDIT\_TRAIL и AUDIT\_SYS\_OPERATIONS. Значение данных параметров продемонстрировано на рисунке 3.1.

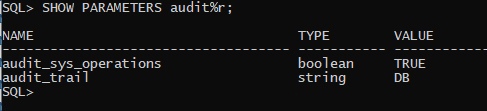


Рисунок 3.1 – Значение параметров, необходимых для аудита

Команда AUDIT в Oracle используется для активации аудита и определения, какие операции должны быть отслежены в базе данных. Установка отслеживаемых операций требует наличие привилегий AUDIT SYSTEM и AUDIT ANY.

Первая команда, для который был применён аудит – это подключения сеанса. Однако наиболее интересующий момент в подключении – неудачные попытки. Чтобы не переполнять журнал аудита ненужной информацией, можно использовать фокусировку, то есть сужение области действия аудита. Столбцы success и failure в представлении dba\_stmt\_audit\_opts (представление содержит информацию о настройках аудита) означают образование записей аудита при удачном и неудачном выполнении команд соответственно. Включение отслеживания только неудачных попыток подключения сеанса продемонстрировано в листинге 3.7.

AUDIT SESSION WHENEVER NOT SUCCESSFUL;

Листинг 3.7 – Включение отслеживания только неудачных попыток подключения сеанса

Далее аудит, как показано в листинге 3.8, был включен для выполнения четырёх процедур, а именно создания и удаления событий и их расписаний, и вставки значений в таблицу запросов возврата билетов.

AUDIT INSERT ON TicketVibe\_TicketRefund;

AUDIT EXECUTE ON TicketVibe\_CreateEvent;

AUDIT EXECUTE ON TicketVibe\_CreateEventSchedule;

AUDIT EXECUTE ON TicketVibe\_DeleteEvent;

AUDIT EXECUTE ON TicketVibe\_DeleteEventSchedule;

Листинг 3.8 – Включение параметров аудита

Так как параметр AUDIT\_TRAIL инициализирован значением DB, то данные аудита хранятся в таблице AUD$. Для доступа к данным удобнее использовать представление DBA\_AUDIT\_TRAIL. В листинге 3.9 продемонстрирована процедура, позволяющая просмотреть список неудачных подключений. В данном случае используется представление DBA\_AUDIT\_SESSION, которое базируется на вышеупомянутом, содержащее записи, касающиеся команд CONNECT и DISCONNECT. Столбец username – это пользователь, аудит действий которого был выполнен, terminal – компьютер, на котором выполнено действие, timestamp – время, returncode – код ошибки, генерируемый действием.

CREATE OR REPLACE PROCEDURE FailerLoginAttempts (

Cursor\_out OUT SYS\_REFCURSOR

)

AS

BEGIN

OPEN Cursor\_out FOR

SELECT username,

terminal,

timestamp

FROM dba\_audit\_session

WHERE returncode <> 0

GROUP BY username, terminal, TO\_CHAR(timestamp, 'dd/mm/yyyy');

END;

Листинг 3.9 – Процедура вывода аудита неудачных подключений

Ещё пример – отслеживание создания новых событий в нерабочее время для организаторов. Создание процедуры продемонстрировано в листинге 3.10.

CREATE OR REPLACE PROCEDURE CreateEventAfterHours (

Cursor\_out OUT SYS\_REFCURSOR

)

IS

BEGIN

OPEN Cursor\_out FOR

SELECT username, terminal, timestamp, action\_name

FROM dba\_audit\_trail a

WHERE action\_name = 'EXECUTE PROCEDURE'

AND ( obj\_name = 'CREATEEVENTSCHEDULE' OR obj\_name = 'CREATEEVENT')

AND TO\_NUMBER(TO\_CHAR(a.timestamp, 'HH24')) NOT BETWEEN 8 AND 22;

END;

Листинг 3.10 – Процедура вывода аудита создания событий в нерабочее время

Для этого отслеживания была создана процедура, получающая значения из представления DBA\_AUDIT\_TRAIL, и проверяющая timestamp на нахождение в определённом временном диапазоне.

## 3.3. Вывод.

В данном разделе курсового проекта по базам данных были рассмотрены ключевые аспекты проектирования и разработки баз данных. Была проведена разработка структуры таблиц, включая определение первичных и внешних ключей, типов данных, ограничения целостности и связей между таблицами. Были созданы процедуры и триггеры, что позволяют автоматизировать выполнение определенных операций. Использование индексов способствовало оптимизации производительности запросов к базе данных, ускоряя поиск и выборку данных. Была описана технология аудита, с помощью которой можно отслеживать подозрительные действия пользователей сервиса.

# 4. Установка, настройка и использование Oracle.

Oracle Database — это реляционная система управления базами данных (СУБД), разработанная и продвигаемая компанией Oracle. Она является одной из самых популярных и мощных баз данных на рынке. Oracle DB предоставляет широкий набор функций для хранения, организации и обработки данных, а также для управления доступом к ним.

## 4.1. Подключение базы данных к приложению WPF.

Подключение базы данных Oracle к приложению WPF осуществляется с помощью ODP.NET — полностью управляемого поставщика данных, обеспечивающего доступ к базе данных Oracle из .NET Framework. Использование ODP.NET предоставляет ряд преимуществ. Во-первых, будучи написанным на управляемом коде, он обеспечивает лучшую интеграцию с .NET Framework и исключает проблемы с развертыванием неуправляемых библиотек. Во-вторых, ODP.NET оптимизирован для высокой производительности и предлагает функции, повышающие скорость и эффективность доступа к данным. В общем ODP.NET отличается простым и интуитивно понятным API для работы с базой данных Oracle.

Процесс подключения начинается с установки ODP.NET. После скачивания последней версии с сайта Oracle, драйвер устанавливается согласно инструкциям мастера установки. Далее в проект WPF, открытый в Visual Studio, добавляется ссылка на ODP.NET. Для этого в обозревателе решений нужно щелкнуть правой кнопкой мыши по папке «Ссылки», выбрать «Добавить ссылку», перейти на вкладку «Обзор» и найти файл «Oracle.ManagedDataAccess.dll» в папке установки ODP.NET.

Следующим шагом является создание строки подключения, содержащей информацию, необходимую для подключения к базе данных Oracle. Она включает имя хоста, номер порта, имя сервиса базы данных, имя пользователя и его пароль. Пример строки подключения продемонстрирован на листинге 4.1.

string connectionString = "DATA SOURCE=localhost:1521/orcl.mshome.net;TNS\_ADMIN=C:\\Users\\oracledatabase\\Oracle\\network\\admin;PERSIST SECURITY INFO=True;USER ID=Programmer\_1;PASSWORD=Qwerty99880";

Листинг 4.1 – Строка подключения

Далее создается объект OracleConnection, представляющий собой подключение к базе данных Oracle. В конструктор объекта передается строка подключения. Для открытия подключения к базе данных используется метод Open() объекта OracleConnection. После установления соединения можно выполнять SQL-запросы к базе данных. Для этого используются объекты OracleCommand. В случае использования процедуры ShowAllComments, код продемонстрирован на листинге 4.2, где тип команды явно указывается как StoredProcedure, а имя процедуры задаётся как «ShowAllComments».

using (connection = new OracleConnection(connectionString))

{

connection.Open();

using (OracleCommand command = connection.CreateCommand())

{

command.CommandType = CommandType.StoredProcedure;

command.CommandText = "TicketVibe\_ShowAllComments";

command.Parameters.Add("result", OracleDbType.RefCursor, ParameterDirection.Output);

OracleDataAdapter adapter = new OracleDataAdapter(command);

DataTable dataTable = new DataTable();

adapter.Fill(dataTable);

CommentsDataGrid.ItemsSource = dataTable.DefaultView;

connection.Close();

}

}

Листинг 4.2 – Пример работы с процедурами в приложении

Каждый параметр хранимой процедуры представляется в коде объектом «OracleParameter». Для создания объекта используются различные конструкторы, позволяющие указать имя параметра, тип данных и направление (входной, выходной или входной/выходной). В конце подключение закрывается.

## 4.2. Создание ролей для разграничения доступа.

Роль представляет собой поименованный набор привилегий, то есть разрешений, которые определяют, какие операции или действия могут быть выполнены. В рамках курсового проекта было разработано четыре роли:

1. Гость (QuestRole). Роль ключает в себя привилегии создания подключения, выполнения процедур ShowCatalog, GetSubcategoriesByCategoryName, SelectFromLocations, SelectFromCategories, GetTicketPricesByEventName, GetPlacesInRowByEventNameAndSectorRow, CreateUserQuestion, FindTicketPrice, ShowEventByEventScheduleID, ShowComments, SearchInCatalog.

2. Пользователь (UserRole). Роль включает в себя привилегии создания подключения, выполнения процедур ShowUserQuestion, ShowCatalog, GetSubcategoriesByCategoryName, SelectFromLocations, SelectFromCategories, GetTicketPricesByEventName, GetPlacesInRowByEventNameAndSectorRow, FindTicketPrice, AddTicketToShoppingCart, ShowShoppingCart, DeleteTicketFromShoppingCart, ClearShoppingCart, BuyTicket, GetPurchaseHistory, AskForRefund, SearchInCatalog, ShowFullScheduleOfEventByEventScheduleID, ShowEventByEventScheduleID, ShowComments, CreateComment, CreateUserQuestion, GetShoppingCartInfo.

3. Организатор (OrganizerRole). Роль включает в себя привилегии создания подключения, выполнения процедур ShowOrganizerEvent, CreateEvent, UpdateEvent, DeleteEvent, ShowOrganizerQuestion, CreateOrganizerQuestion, CreateEventSchedule, ShowEventsSchedule, UpdateEventSchedule, DeleteEventSchedule, ShowCatalog, GetSubcategoriesByCategoryName, SelectFromLocations, SelectFromCategories, GetTicketPricesByEventName, GetPlacesInRowByEventNameAndSectorRow, FindTicketPrice, SearchInCatalog, ShowFullScheduleOfEventByEventScheduleID, ShowEventByEventScheduleID, ShowComments.

4. Менеджер (ManagerRole). Роль включает в себя привилегии создания подключения, обращения к представлениям DBA\_AUDIT\_TRAIL, DBA\_AUDIT\_SESSION, выполнения процедур AddLocation, AddCategory, AddSubcategory, AddAmountOfRowsIntoTempTable, AddRow, UpdateLocation, DeleteLocation, UpdateCategory, DeleteCategory, UpdateSubcategory, DeleteSubcategory, DenyOrganizer, AcceptOrganizer, BlockUser, UpdateRow, DeleteRow, AcceptRefend, DenyRefend, ShowRefund, BlockOrganizer, ShowOrganizerBlock, ShowUserBlock, ShowAllComments, DeleteComment, AcceptOrganizerQuestion, CommentOrganizerQuestion, CloseOrganizerQuestion, ShowOrganizerQuestionForManager, AcceptUserQuestion, CommentUserQuestion, CloseUserQuestion, ShowUserQuestionForManager, DeleteEventScheduleByManager, DeleteEventByManager, UpdateEventScheduleByManager, UpdateEventByManager, ShowAllEvents, ShowAllEventsSchedule, SelectFromLocations, SelectFromCategories, SelectFromSubcategories, SelectFromSectorRows, SelectFromCheckingOrganizers, ShowCatalog, GetSubcategoriesByCategoryName, GetTicketPricesByEventName, GetPlacesInRowByEventNameAndSectorRow, FindTicketPrice, ShowFullScheduleOfEventByEventScheduleID, ShowEventByEventScheduleID, ShowComments, SearchInCatalog, FailerLoginAttempts, AuditTicketRefund, CreateEventAfterHours, AuditEventDelete.

## 4.3. Описание процедур экспорта и импорта.

XML — это язык разметки, используемый для хранения и передачи структурированных данных. Он определяет набор правил для кодирования документов в формате, который является удобочитаемым как для человека, так и для машины.

Код, представленный в листинге 4.3, формирует XML документ из данных таблицы Events. Для этого он использует функцию XMLELEMENT, которая формирует XML элементы для каждого мероприятия и объединяет их в общий список мероприятий. Затем XML-документ преобразуется в CLOB – формат, который используется для хранения больших текстовых данных.

Далее открывается XML-файл с помощью инструмента UTL\_FILE и записывается CLOB в файл по частям.

В итоге код создает XML-файл с информацией о мероприятиях, взятой из базы данных. Это позволяет сохранять данные в удобном для обмена формате.

CREATE OR REPLACE FUNCTION export\_events\_to\_xml(p\_filename IN VARCHAR2) RETURN VARCHAR2 IS

l\_xml XMLTYPE;

l\_file\_handle UTL\_FILE.file\_type;

l\_clob CLOB;

l\_amount BINARY\_INTEGER := 32767;

l\_offset INTEGER := 1;

l\_raw RAW(32767);

BEGIN

SELECT XMLELEMENT("Events",

XMLAGG(

XMLELEMENT("Event",

XMLELEMENT("EventID", EventID),

XMLELEMENT("EventName", EventName),

XMLELEMENT("LocationID", LocationID),

XMLELEMENT("EventDuration", EventDuration),

XMLELEMENT("SubcategoryID", SubcategoryID),

XMLELEMENT("OrganizerID", OrganizerID),

XMLELEMENT("Description", Description),

XMLELEMENT("Cost", Cost))

)

)

INTO l\_xml FROM TicketVibe\_Events;

l\_clob := l\_xml.getClobVal();

l\_file\_handle := UTL\_FILE.fopen('XML\_DIRECTORY', p\_filename, 'wb');

WHILE l\_offset < DBMS\_LOB.getlength(l\_clob) LOOP

l\_raw := UTL\_RAW.cast\_to\_raw(DBMS\_LOB.substr(l\_clob, l\_amount, l\_offset));

UTL\_FILE.put\_raw(l\_file\_handle, l\_raw);

l\_offset := l\_offset + l\_amount;

END LOOP;

UTL\_FILE.fclose(l\_file\_handle);

RETURN 'Events exported to ' || p\_filename || ' successfully.';

END;

Листинг 4.3 – Процедура экспорта в xml файл

Код из листинга 3.3 автоматизирует процесс добавления новых мероприятий в базу данных. Информация о мероприятиях хранится в XML файле.

Для начала, открывается XML файл, используя инструмент UTL\_FILE, затем информация о каждом мероприятии преобразуется в формат, понятный для базы данных, используя XMLTYPE – инструмент, который играет роль переводчика.

Далее код использует extractvalue, чтобы найти конкретную информацию о мероприятии, например, его название, стоимость, место проведения и т.д. Это похоже на поиск нужной информации в текстовом документе.

## 4.4. Тестирование производительности базы данных.

В данном исследовании проводится тестирование базы данных с целью оценки её производительности при работе с большим объемом данных. Для этого будет использована таблица, в которую будет добавлено 100 000 строк данных. После заполнения базы данных будет выполнена проверка времени поиска данных в ней, в том числе с использованием индексов и без них. Тестирование будет проводится на таблице Tickets, так как это самая объёмная по количеству данных таблица в базе данных. На момент теста в таблице находится 102975 записей, созданных при помощи триггера AFTER\_INSERT\_EVENTSCHEDULE. Тестирование производится на основе процедуры по поиску стоимости билета FindTicketPrice, где по уникальному идентификатору времени проведения мероприятия, ряду и месту в ряду ищется стоимость билета. Время выполнения процедуры до применения индекса равно 103 мс (рисунок 4.1).

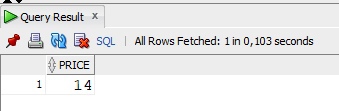


Рисунок 4.1 – Время выполнения процедуры до добавления индекса

На рисунке 4.2 представлен результат после добавления индекса по столбцу EventScheduleID. Время выполнения сократилось до 17 мс.

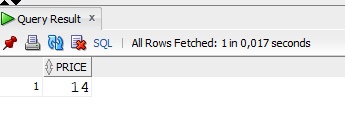


Рисунок 4.2 – Время выполнения процедуры до добавления индекса

## 4.5. Вывод.

В данном разделе курсовой работы исследуется использование реляционной СУБД Oracle для управления данными о мероприятиях. В системе реализована ролевая модель доступа с четырьмя ролями: гость, пользователь, организатор и менеджер. Каждая роль наделена специфическим набором привилегий, определяющим доступные операции, что обеспечивает безопасность данных.

Для обмена данными реализованы процедуры экспорта и импорта данных в формате XML. Процедура экспорта формирует XML файл из данных таблицы, используя функции XMLELEMENT и UTL\_FILE. Процедура импорта считывает данные из XML файла и добавляет их в базу, используя UTL\_FILE, XMLTYPE и extractvalue.

Проведено тестирование производительности базы данных путем добавления 100 000 записей в таблицу Tickets и измерения времени выполнения процедуры поиска FindTicketPrice. Результаты показали, что использование индекса по столбцу EventScheduleID сократило время выполнения процедуры с 103 мс до 17 мс, демонстрируя значительное улучшение производительности.

Данный раздел демонстрирует практическое применение Oracle для управления данными о мероприятиях с акцентом на безопасность, обмен данными и производительность.

# 5. Тестирование, проверка работоспособности и анализ полученных результатов.

## 5.1. Тестирование гостевой части.

Чтобы начать работу в системе, гость не должен проходить этапы регистрации и авторизации. В приложении имеется отдельная кнопка для неавторизованного входа, продемонстрированная на рисунке 5.1.

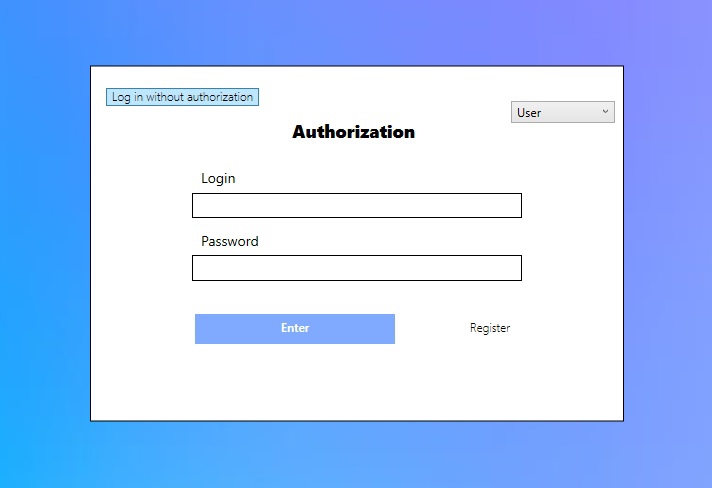


Рисунок 5.1 – Неавторизованный вход

Каждый пользователь системы после авторизации попадает на страницу каталога мероприятий, в том числе и гость. В каталоге мероприятия отображаются со следующей информацией: название мероприятия, место проведения, продолжительность, категория, подкатегория, минимальная стоимость билетов, дата и время проведения. На странице каталога можно выполнить фильтрацию мероприятий по дате, локации, категории и подкатегории, стоимости. Выпадающий список подкатегорий генерируется в случае, если выбрана категория, за что отвечает процедура GetSubcategoriesByCategoryName. Фильтрация запускается при нажатии соответствующей кнопки. Кроме того, можно выбрать способ сортировки каталога: по дате или по названию, по возрастанию или убыванию. На рисунке 5.2 продемонстрировано состояние каталога до применения фильтрации.

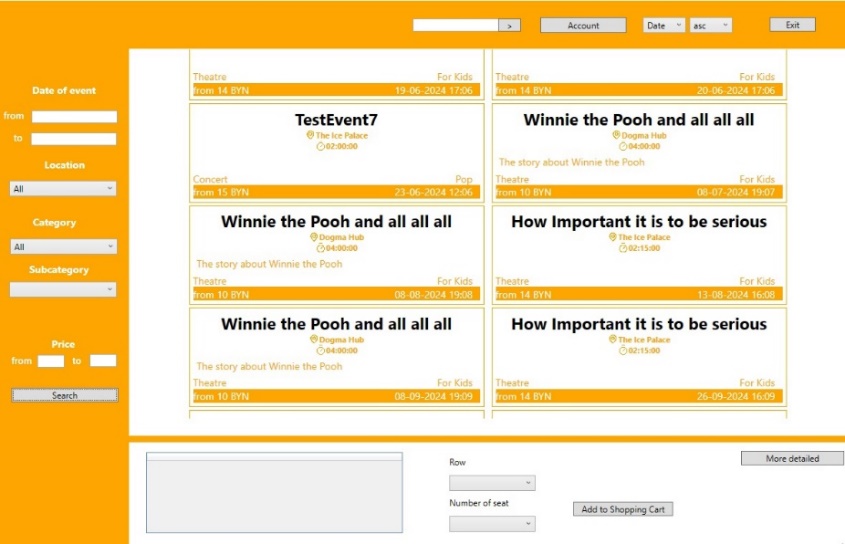


Рисунок 5.2 – Каталог до применения фильтрации

Для фильтрации используется процедура ShowCatalog. На рисунке 5.3 показан пример фильтрации с указанием категории, подкатегории, даты проведения и стоимости билетов.

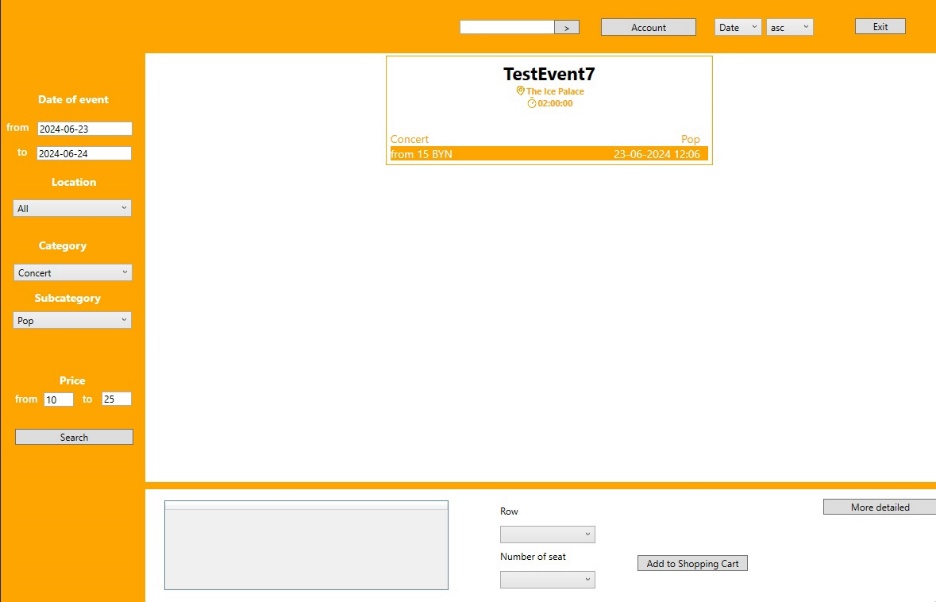


Рисунок 5.3 – Первый вариант применения фильтрации

На рисунке 5.4 продемонстрирована сортировка по имени в убывающем порядке.

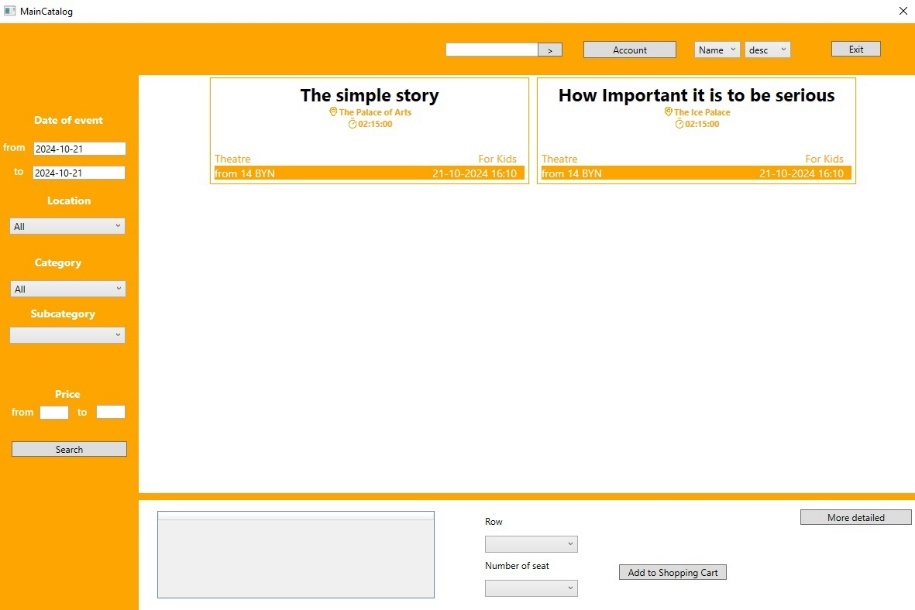


Рисунок 5.4 – Второй вариант применения фильтрации

Имеется возможность частичного поиска по каталогу, реализованная с помощью процедуры SearchInCatalog, которая представлена на рисунке 5.5.

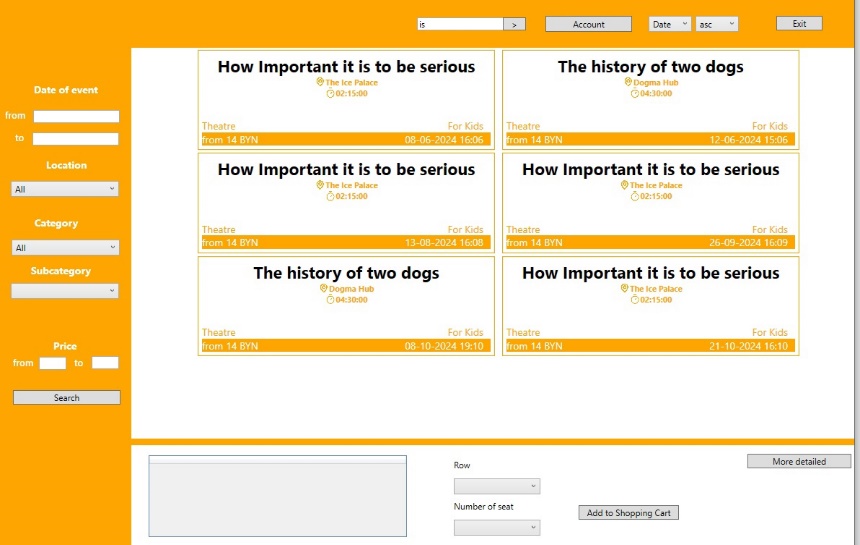


Рисунок 5.5 – Поиск по каталогу

При клике на мероприятие в каталоге запускается процедура GetTicketPricesByEventName, в результате которой в секцию выбора билета выводится информация о билетах выбранного мероприятия в виде списка следующего формата: номер ряда, количество доступных к продаже мест в этом ряду и цена, которая высчитывается как цена мероприятия, умноженная на множитель цены ряда, как на рисунке 5.6. Из этой же процедуры извлекается значение результирующего столбца SectorRow для внесения его в выпадающий список Row, с которого пользователь начинает выбор билета. При выборе номера ряда билета в списке Row, запускается следующая процедура – GetPlacesInRowByEventNameAndSectorRow, которая по выбранному ряду выводит список всех доступных мест в нём, и они заносятся в следующий выпадающий список. После того, как будет выбрано место, запускается процедура FindTicketPrice, что выведет стоимость билета.

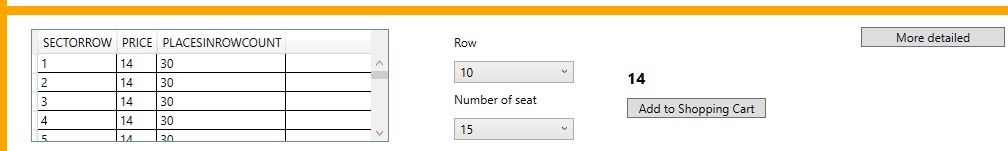


Рисунок 5.6 – Раздел выбора билета

Кнопка добавления билета в корзину недоступна, но можно посмотреть более подробную информацию о мероприятии, нажав на кнопку «More detailed». На странице мероприятия, представленной на рисунке 5.7, можно увидеть информацию о мероприятии (процедура ShowEventByEventScheduleID), полное расписание проведения данного мероприятия (процедура ProcedureShowFullScheduleOfEvent), комментарии к данному мероприятию (процедура ShowComments).

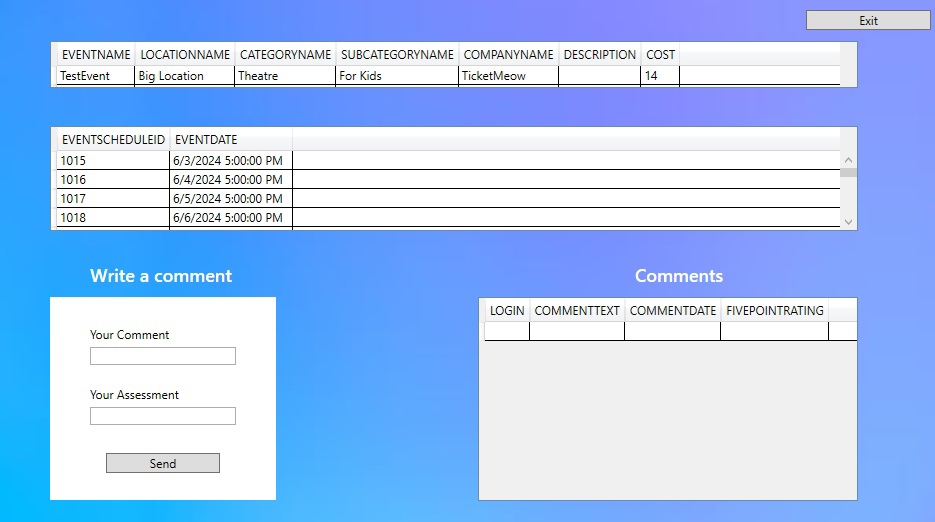


Рисунок 5.7 – Страница мероприятия

Есть блок для написания комментариев, но он недоступен для гостя и, как показано на рисунке 5.8, выдаёт ошибку.

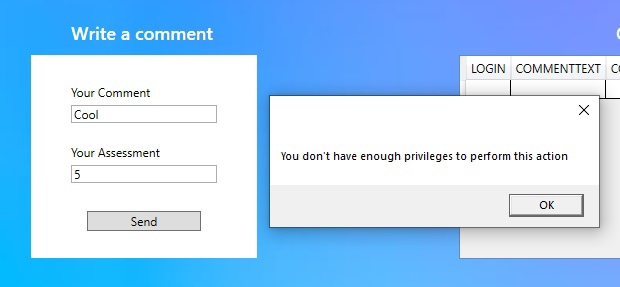


Рисунок 5.8 – Ошибка, предупреждающая о нехватке привилегий

Единственное, чем отличается каталог для гостя – наличие поля, представленного на рисунке 5.9, для написания вопроса менеджеру.

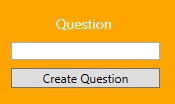


Рисунок 5.9 – Поле для написания вопроса гостем

Вопросы будут попадать в таблицу менеджера с пользовательскими вопросами.

## 5.2. Тестирование пользовательской части.

Для создания пользователя имеется отдельное окно регистрации, продемонстрированное на рисунке 5.10.

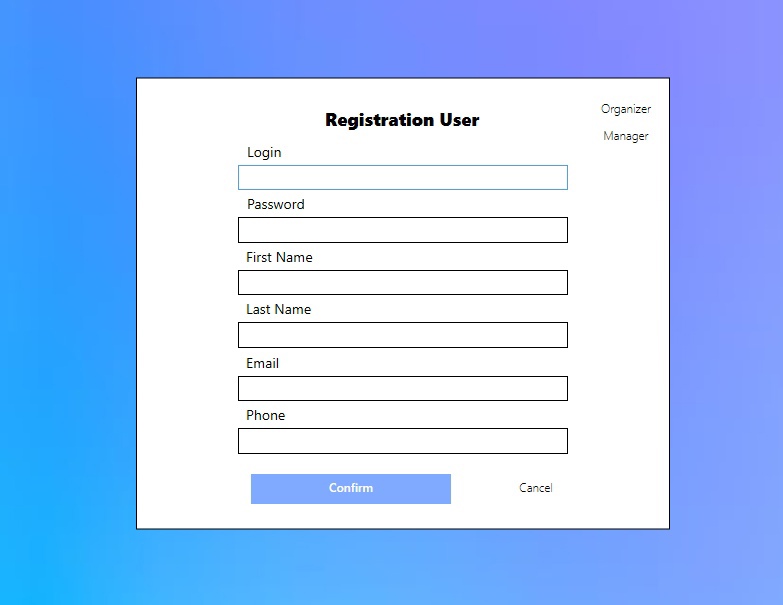


Рисунок 5.10 – Окно регистрации пользователя

Регистрация осуществляется при помощи процедуры UserRegistration. Значения, введённые в каждое поле, проходят валидацию внутри процедуры. Значения должны подходить по следующим параметрам: логин должен быть уникальным с длиной от 4 до 40 символов, пароль с длинной от 5 до 30 символов, имя с длиной от 2 до 20 символов, фамилия с длиной от 2 до 25 символов, электронная почта должна быть уникальной с длиной от 5 до 50 символов и подходящей под регулярное выражение, телефон должен быть уникальным с длиной от 8 до 30 символов и подходящим под регулярное выражение, представленное на листинге 5.1. Пример вывода ошибки продемонстрирован на рисунке 5.11.

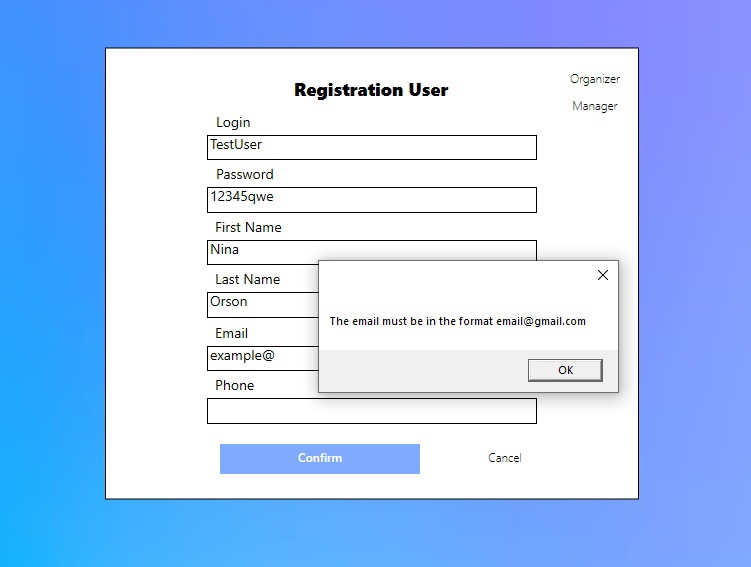


Рисунок 5.11 – Ошибка при неверном вводе данных в окно регистрации

IF NOT REGEXP\_LIKE(Email\_in, '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$') THEN

RAISE Ex\_Invalid\_Email\_Reg;

END IF;

IF NOT REGEXP\_LIKE(Phone\_in, '^\+375\s(?:33|25|17|44|29)\s\d{3}\s\d{2}\s\d{2}$') THEN

RAISE Ex\_Invalid\_Phone\_Reg;

END IF;

Листинг 5.1 – Регулярные выражения, используемые в регистрации

После прохождения всех проверок на корректность введённых значений, соответствующая запись добавляется в таблицу Users, как показано на рисунке 5.12.



Рисунок 5.12 – Добавления нового пользователя в таблицу

Также создаётся новый пользователь в базе данных. Наличие пользователя продемонстрировано на рисунке 5.13.

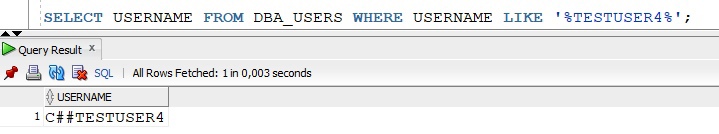


Рисунок 5.13 – Создание нового пользователя в системе

Далее работа продолжается с окном авторизации, как показано на рисунке 5.14.

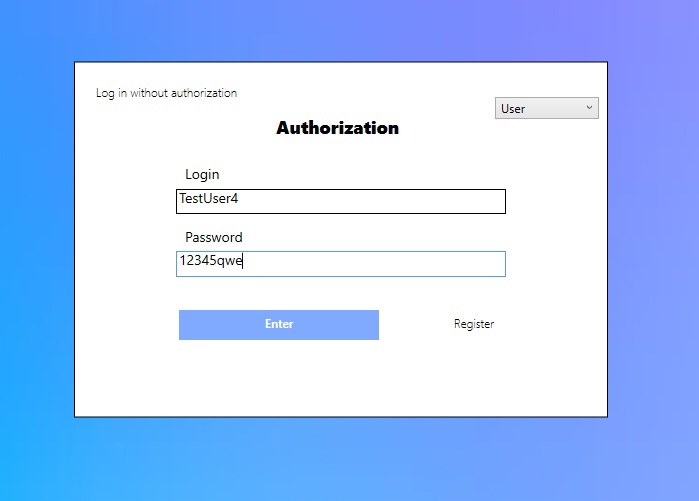


Рисунок 5.14 – Окно авторизации

Пользователь вводит логин и пароль, процедура Authorization ищет в таблице пользователя с переданным логином и сверяет его пароль с полученным. Если правильный пароль не совпадает с полученным, пользователь получает ошибку, продемонстрированную на рисунке 5.15.

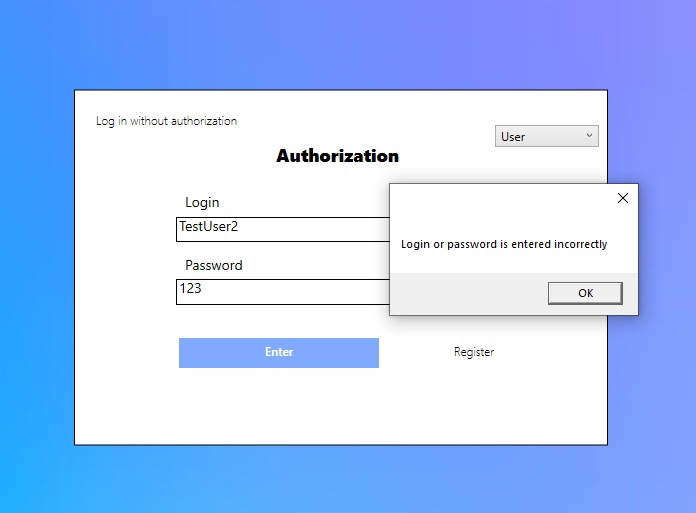


Рисунок 5.15 – Ошибка при вводе неверного пароля

В случае, если пользователь с логином, который был введён при авторизации, вообще не существует в базе данных, то выводится соответствующая ошибка, как на рисунке 5.16.



Рисунок 5.16 – Ошибка при вводе неверного логина

Далее создаётся новая строка подключения, как показано на листинге 5.2, через которую устанавливается связь с базой данных. Пользователь попадает на главный каталог.

OracleConnection connection;

string connectionString;

private string login;

private string password;

private int role;

public MainCatalog(string login, string password, int role)

{

InitializeComponent();

connectionString = $"DATA SOURCE=localhost:1521/orcl.mshome.net;TNS\_ADMIN=C:\\Users\\oracledatabase\\Oracle\\network\\admin;PERSIST SECURITY INFO=True;USER ID=C##{login};PASSWORD={password}";

this.login = login;

this.password = password;

this.role = role;

}

Листинг 5.2 – Создание новой строки подключения

Работа с каталогом у пользователя соответствует работе гостя. Однако теперь пользователь может добавлять билеты в корзину. После добавления, билеты оказываются в личном кабинете в корзине пользователя, продемонстрированной на рисунке 5.16.

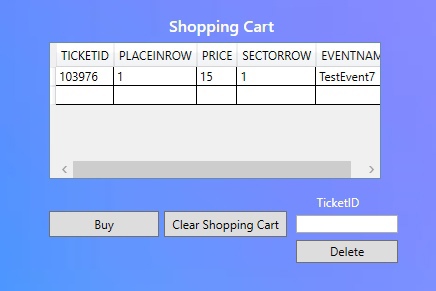


Рисунок 5.16 – Корзина пользователя

С корзиной доступны следующий действия: удаление одного билета по его уникальному идентификатору (процедура DeleteTicketFromShoppingCart), полная очистка корзины (процедура ClearShoppingCart), покупка, при которой оплачиваются все билеты в корзине (процедура BuyTicket).

Можем вернуться на страницу каталога и увидеть, что билет пропал из продажи (из продажи он исчезает ещё на этапе добавления билета в корзину, он становится забронированным). Отсутствие билета в продаже продемонстрировано на рисунках 5.17 и 5.18. Видно, что количество билетов в ряду стало на единицу меньше.

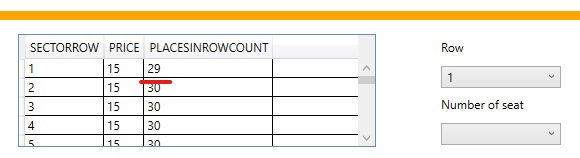


Рисунок 5.17 – Пропажа билета из продажи

Также при рассмотрении списка мест в выбранном ряду можно заметить пропажу билета.

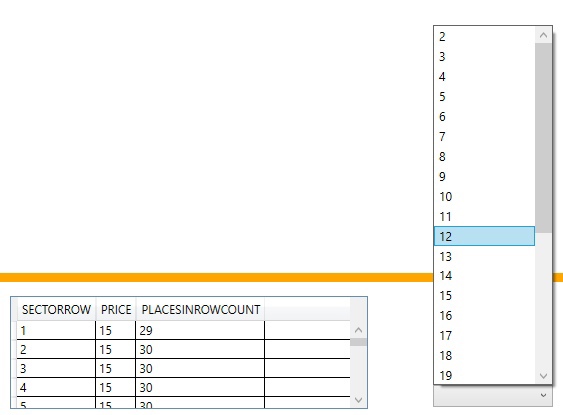


Рисунок 5.18 – Отсутствие купленного места в списке

После покупки, мы можем просмотреть историю покупок, продемонстрированную на рисунке 5.19, в этом же личном кабинете.

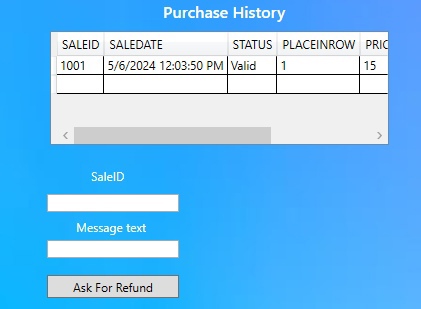


Рисунок 5.19 – История покупок пользователя

С историей покупок доступна ещё одна функция – отправка запроса на возврат билета, которая осуществляется с помощью процедуры AskForRefund. Этот запрос приходит в личный кабинет менеджера, который может его отклонить или принять. В случае принятия запроса, статус покупки меняется на «Возврат», как на рисунке 5.20.



Рисунок 5.20 – Изменение статуса покупки на «Возврат»

Если вернуться к каталогу и посмотреть билеты, доступные к продаже, то, как продемонстрировано на рисунке 5.21, видно, что билет возвращается к продаже.

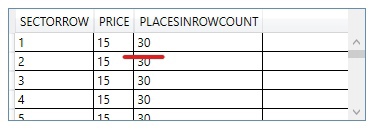


Рисунок 5.21 – Возврат билета в продажу

В системе имеется возможность написания вопроса в систему поддержки, для этого имеется форма в личном кабинете, продемонстрированная на рисунке 5.22. Пользователь может написать текст вопроса и отправить его с помощью процедуры CreateUserQuestuion.

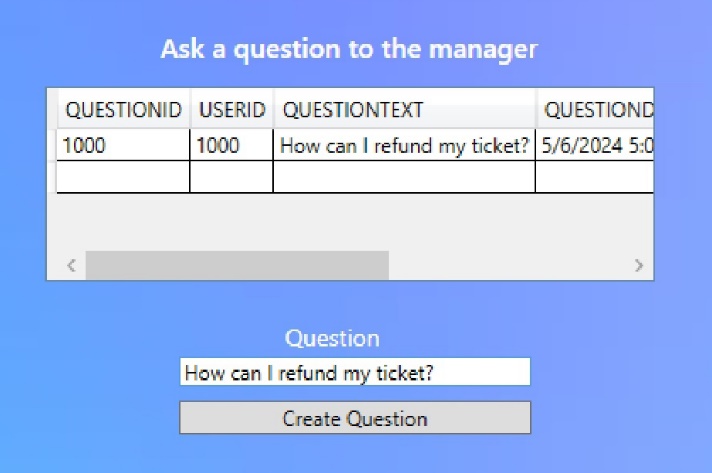


Рисунок 5.22 – Форма вопросов пользователя

Важные критерии в данной форме для пользователя – статус вопроса и ответ. На момент создания, статус будет «Не просмотрено», а поле с ответом – пустое, как на рисунке 5.23.

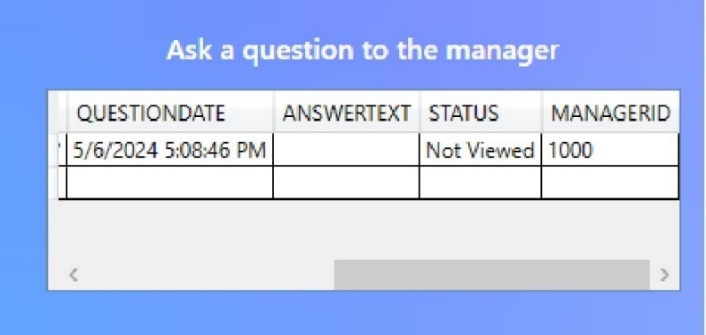


Рисунок 5.23 – Состояние вопроса сразу после создания

Через время пользователь может получить ответ, причём ответы от менеджера могут поступать несколько раз, каждый из которых будет отмечен временем написания. В момент работы менеджера над вопросом статус меняется на «В процессе», а в дальнейшем поменяется на «Закрыто». Пример, как может выглядеть форма с вопросами во время работы менеджера над ними, предоставлен на рисунке 5.24.

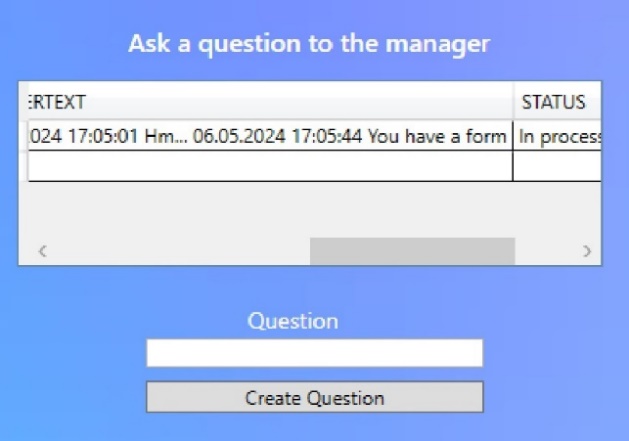


Рисунок 5.24 – Состояние вопроса при работе менеджера

Пользователь также может быть заблокирован менеджером. В случае блокировки, пользователь будет получать отказ при попытке входа в систему, как на рисунке 5.25.

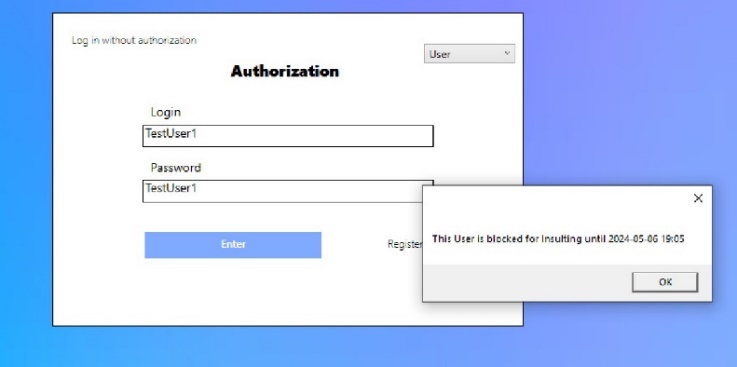


Рисунок 5.25 – Предупреждение о блокировке

При нахождении пользователя в списке заблокированных, прежде чем отправить сообщение о блокировке пользователю, дата окончания блокировки сверяется с текущей. Если время блокировки истекло, пользователь удаляется из таблицы заблокированных и может перейти к просмотру каталога.

## 5.3. Тестирование области работы организатора.

Процесс регистрации и авторизации у организатора аналогичен регистрации и авторизации пользователя. Организаторы также могут просматривать каталог, фильтровать его, просматривать информацию об отдельных мероприятиях и их комментарии. В их личном кабинете имеется форма отправки вопросов менеджерам, аналогичная форме пользователей.

В личном аккаунте организатора имеется форма для создания мероприятия, продемонстрированная на рисунке 5.26. Мероприятия создаются при помощи процедуры CreateEvent. Длина названия мероприятия должна быть от 4 до 50 символов, описание может быть пустым, а для корректности ввода продолжительности мероприятия есть регулярное выражение, продемонстрированное в листинге 5.3.

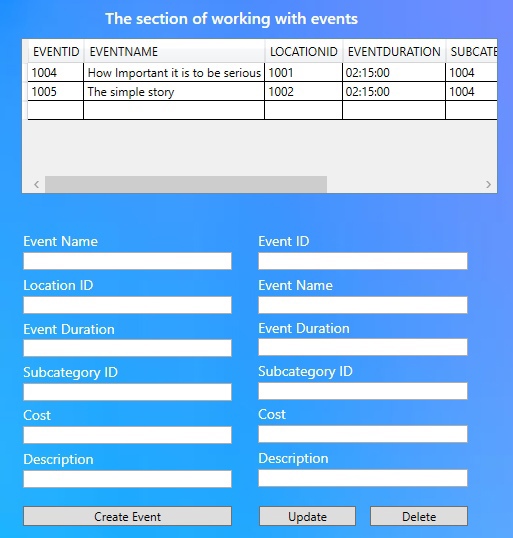


Рисунок 5.26 – Форма создания мероприятий организатором

IF REGEXP\_LIKE(EventDuration\_in, '^\d+ \d{2}:\d{2}:\d{2}$') THEN

interval\_EventDuration := TO\_DSINTERVAL(EventDuration\_in);

ELSE

RAISE Ex\_Invalid\_Interval;

END IF;

Листинг 5.3 – Регулярное выражение, используемое в создании событий

После создания мероприятия, организатор может его изменять (процедура UpdateEvent) или удалять (процедура DeleteEvent). Также в личном аккаунте есть отдельная форма с созданием расписания мероприятия. Создавать расписание (процедура CreateEventsSchedule), изменять его (процедура UpdateEventsSchedule) и удалять (процедура DeleteEventsSchedule), организатор может только в отношении своего собственного мероприятия. Форма представлена на рисунке 5.27.

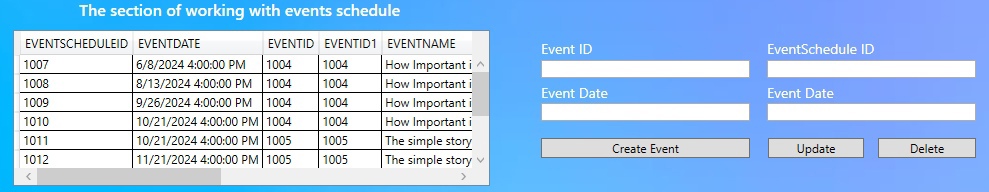


Рисунок 5.27 – Форма создания расписания мероприятий организатором

При создании расписания мероприятия, важно учитывать, чтобы выбранная локация не была занята в указанное время, иначе система предупредит об ошибке. Пример подобного предупреждения представлен на рисунке 5.28.

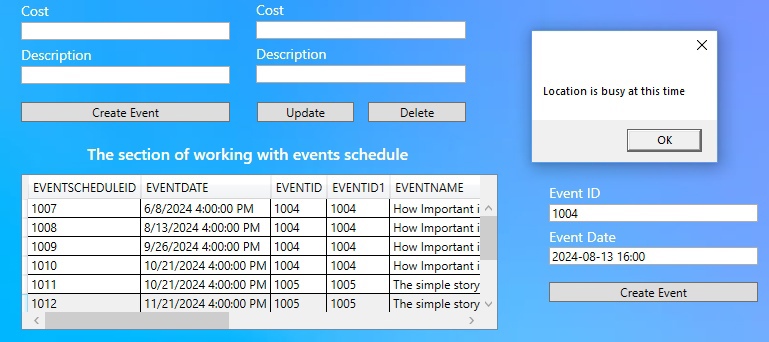


Рисунок 5.28 – Предупреждение о недоступности локации в указанное время

После создания расписания мероприятия, при помощи триггера AFTER\_INSERT\_EVENTSCHEDULE в системе автоматически появляются доступные к продаже билеты на все места указанной локации.

## 5.4. Тестирование области работы менеджера.

Процесс регистрации и авторизации у менеджера аналогичен регистрации и авторизации пользователя и организатора. Менеджеры также могут просматривать каталог, фильтровать его, просматривать информацию об отдельных мероприятиях и их комментарии.

В личном кабинете менеджеров имеется секция работы с локациями, продемонстрированная на рисунке 5.29. Менеджер вводит название локации и количество рядов в ней. По умолчанию в каждом ряду создаётся по 30 мест и присваивается множитель цены, равный единице (процедура AddLocation). Имя локации и количество рядов в ней можно изменять при помощи процедуры UpdateLocation. При изменении количества рядов в локации, корректируются таблицы доступных в системе рядов и билетов. Локацию можно удалять, за что отвечает процедура DeleteLocation.

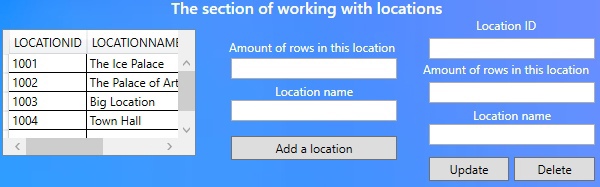


Рисунок 5.29 – Секция работы с локациями

Существует секция работы с категориями и подкатегориями, представленные на рисунке 5.30. При создании категории, за что отвечает процедура AddCategory, менеджер указывает название, которое в последствии можно сменить. В соответствии с процедурой AddSubcategory, при создании подкатегории также необходимо указать, к какой категории она будет относиться. У одной категории не может быть несколько подкатегорий с одинаковым названием, но названия подкатегорий, относящихся к разным категориям, могут быть одинаковыми. Соответственно категории и подкатегории можно изменять (процедуры UpdateCategory и UpdateSubcategory) и удалять (процедуры DeleteCategory и DeleteSubcategory).

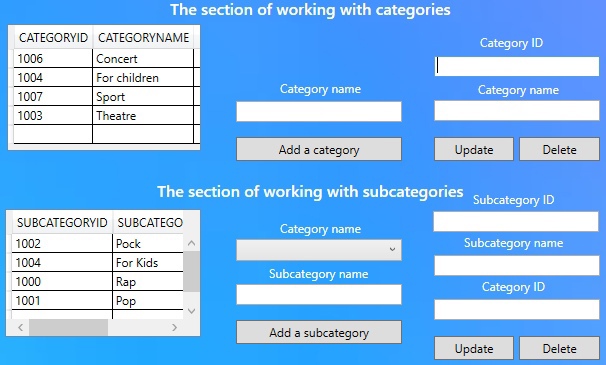


Рисунок 5.30 – Секция работы с категориями и подкатегориями

На рисунке 5.31 изображена секция работы с рядами. Можно добавить ряд к существующей локации (процедура AddRow), указав количество сидений и множитель цены, а также можно изменять ряды или их удалять при помощи процедур UpdateRow и DeleteRow. В соответствии с изменением рядов, корректируются имеющиеся в продаже билеты.

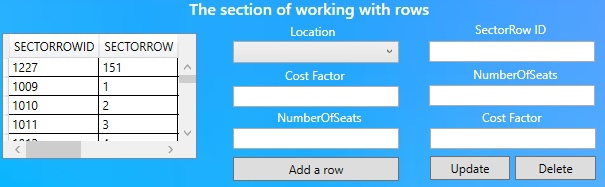


Рисунок 5.31 – Секция работы с рядами

В личном кабинете менеджера имеется секция работы с запросами на возврат билетов, представленный на рисунке 5.32. Менеджер может отклонять или принимать запросы в соответствии с процедурой AcceptRefund.

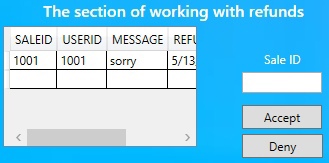


Рисунок 5.32 – Секция работы с запросами на возврат билетов

Также менеджер имеет доступ к секции одобрения заявок на регистрацию от организаторов с подобным функционалом, продемонстрированной на рисунке 5.33, прописанным в процедуре AcceptOrganizer.

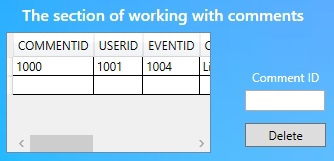


Рисунок 5.33 – Секция работы с запросами на регистрацию от организаторов

У менеджеров имеется секция работы с блокировками, представленная на рисунке 5.34. Для блокировки пользователей и организаторов имеются разные формы. Менеджеры вводят информацию о причине блокировки и её дате завершения. За блокировку пользователей отвечает процедура BlockUser, а за блокировку организаторов – BlockOrganizer.

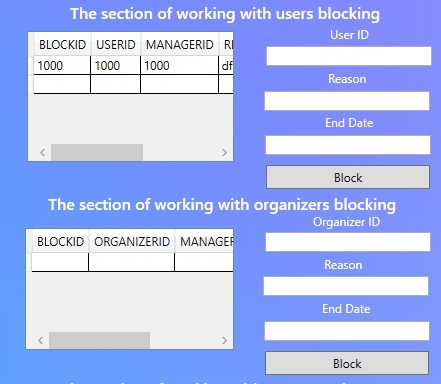


Рисунок 5.34 – Секция работы с блокировками

В кабинете присутствует секция работа с вопросами. Вопросы от пользователей и менеджеров разделены на разные формы. Когда пользователь или организатор отправляет вопрос, процедура FindManagerWithLeastQuestions находит менеджера с наименьшим количеством запросов и в ячейке ответственного менеджера за вопрос ставит его. Менеджер, в соответствии с процедурами AcceptOrganzerQuestion и AcceptUserQuestion, может принять вопрос, в таком случае статус вопроса меняется на «В процессе» с «Не просмотрено», ответить на вопрос, в соответствии с процедурами CommentOrganzerQuestion и CommentUserQuestion, причём количество ответов не ограниченно и их можно писать до тех пор, пока вопрос не будет закрыт. К каждому ответу приурочивается дата его написания, все ответы складываются в одну строку и отображаются в столбце «Ответ» таблицы вопросов. Если на момент написания первого ответа, статус вопроса ещё не был изменён с «Не просмотрено», то это делается автоматически. И, соответственно, менеджер может закрыть вопрос, используя процедуры CloseOrganzerQuestion и CloseUserQuestion. Секция представлена на рисунке 5.35.

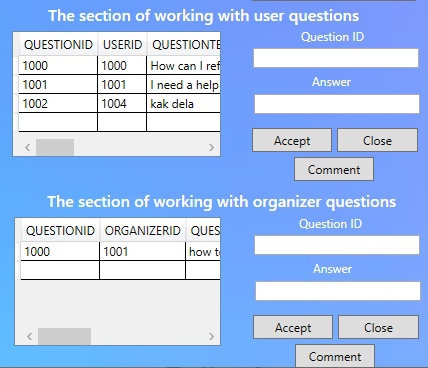


Рисунок 5.35 – Секция работы с вопросами

Менеджер может изменять и удалять все мероприятия и расписания в системе. Разница с соответствующей функцией у организаторов в том, что организаторы при изменении или удалении проходят проверку на владение мероприятием, а менеджеры эти действия могут выполнять с любым мероприятием.

Менеджер имеет доступ к странице с аудитом, продемонстрированной на рисунке 5.36, где представлено четыре таблицы.

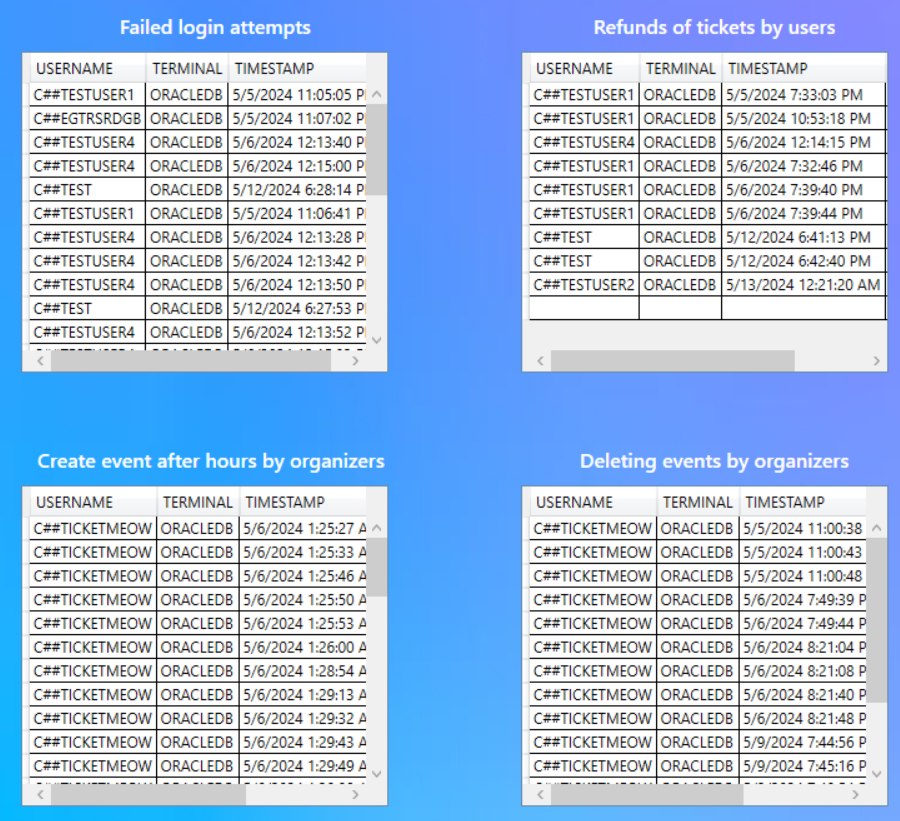


Рисунок 5.36 – Страница просмотра результатов аудита

Имеется информация о неудачных попытках входа (процедура FailerLoginAttempts), об отправлении заявок на возврат билета (AuditTicketRefund), о создании мероприятий организаторами в нерабочее время (CreateEventAfterHours), об удалении мероприятий организаторами (AuditEventDelete).

## 5.3. Вывод.

Раздел посвящен тестированию всех функциональных возможностей разработанной системы для различных типов пользователей – гостей, зарегистрированных пользователей, организаторов и менеджеров. Тестирование подтвердило полную работоспособность системы. Все функции, включая фильтрацию каталога, покупку билетов, создание мероприятий, обработку запросов на возврат и взаимодействие с менеджерами через вопросы, работают корректно и соответствуют заявленным требованиям. Валидация данных на этапе регистрации и создания мероприятий предотвращает возникновение ошибок, а система аудита позволяет контролировать действия пользователей и отслеживать важные события.

# Заключение

В данной курсовой работе разработана система управления мероприятиями, основанная на реляционной СУБД Oracle.

В процессе работы были проанализированы аналогичные системы, сформулированы функциональные требования, спроектирована детальная схема базы данных с определением таблиц, полей, ограничений целостности и связей. Разработанная ролевая модель обеспечивает гибкое управление доступом, разграничивая права гостей, пользователей, организаторов и менеджеров.

Для реализации бизнес-логики разработаны процедуры и триггеры, автоматизирующие процессы, такие как регистрация пользователей, добавление мероприятий, покупка билетов и обработка запросов на возврат. Использование индексов и оптимизация запросов обеспечивают высокую производительность системы. Механизмы экспорта и импорта данных в формате XML расширяют возможности взаимодействия с внешними системами.

Тестирование системы подтвердило ее полную работоспособность и соответствие заявленным требованиям. В результате была создана надежная, функциональная и производительная система, отвечающая изначально поставленным требованиям к управлению мероприятиями.

# Список использованных литературных источников

1. TicketPro [Электронный ресурс] – Режим доступа: https://www.ticketpro.by/. – Дата доступа: 10.03.2024.

2. Kvitki.by [Электронный ресурс] – Режим доступа: https://www.kvitki.by/. – Дата доступа: 10.03.2024.

3. Bezkassira [Электронный ресурс] – Режим доступа: https://bezkassira.by/. – Дата доступа: 10.03.2024.

4. Леоненков А. В. Самоучитель UML / Леоненков Александр Васильевич – 2-е изд. – СПб.: БХВ-Петербург, 2004. – 432 с.

5. Кайт Том Oracle для профессионалов / Том Кайт – СПб.: ООО «ДиаСофтЮП», 2003. – 672 с.

# Приложение А. Листинг таблиц

CREATE TABLE Organizers (

OrganizerID NUMBER PRIMARY KEY,

CompanyName NVARCHAR2(40) NOT NULL UNIQUE,

Password NVARCHAR2(30) NOT NULL,

FirstName NVARCHAR2(20) NOT NULL,

LastName NVARCHAR2(25) NOT NULL,

Email NVARCHAR2(50) NOT NULL UNIQUE,

Phone NVARCHAR2(30) NOT NULL UNIQUE,

RoleID NUMBER NOT NULL,

CONSTRAINT fk\_RoleOrganizer FOREIGN KEY (RoleID) REFERENCES Roles(RoleID)

);

Листинг А.1 – Таблица организаторов

CREATE TABLE Sales (

SaleID NUMBER PRIMARY KEY,

UserID NUMBER NOT NULL,

TicketID NUMBER NOT NULL,

SaleDate DATE NOT NULL,

Status NVARCHAR2(50) NOT NULL CHECK (Status IN ('Valid', 'Refund')),

CONSTRAINT fk\_UserOfSale FOREIGN KEY (UserID) REFERENCES Users(UserID),

CONSTRAINT fk\_TicketOfSale FOREIGN KEY (TicketID) REFERENCES Tickets(TicketID),

);

Листинг А.2 – Таблица продаж

CREATE TABLE SectorRows (

SectorRowID NUMBER PRIMARY KEY,

SectorRow NUMBER NOT NULL,

NumberOfSeats NUMBER NOT NULL,

LocationID NUMBER NOT NULL,

CostFactor NUMBER NOT NULL,

CONSTRAINT fk\_Location FOREIGN KEY (LocationID) REFERENCES Locations(LocationID);

);

Листинг А.3 – Таблица рядов

CREATE TABLE Tickets (

TicketID NUMBER PRIMARY KEY,

EventScheduleID NUMBER NOT NULL,

Status NVARCHAR2(50) NOT NULL CHECK (Status IN ('On sale', 'Booked', 'Purchased')),

Price NUMBER NOT NULL,

SectorRowID NUMBER NOT NULL,

PlaceInRow NUMBER NOT NULL,

CONSTRAINT fk\_TicketsEventsScheduleID FOREIGN KEY (EventScheduleID) REFERENCES EventsSchedule(EventScheduleID),

CONSTRAINT fk\_SectorRowOfTicket FOREIGN KEY (SectorRowID) REFERENCES SectorRows(SectorRowID)

);

Листинг А.4 – Таблица билетов

CREATE TABLE UserBlocks (

BlockID NUMBER PRIMARY KEY,

UserID NUMBER NOT NULL,

ManagerID NUMBER NOT NULL,

Reason NVARCHAR2(200) NOT NULL,

EndDate DATE,

CONSTRAINT fk\_UserBlocks FOREIGN KEY (UserID) REFERENCES Users(UserID),

CONSTRAINT fk\_ManagerBlocks\_User FOREIGN KEY (ManagerID) REFERENCES Managers(ManagerID)

);

Листинг А.5 – Таблица блокировок пользователей

CREATE TABLE UserQuestions (

QuestionID NUMBER PRIMARY KEY,

UserID NUMBER NOT NULL,

QuestionText NVARCHAR2(500) NOT NULL,

QuestionDate DATE NOT NULL,

AnswerText NVARCHAR2(500),

Status NVARCHAR2(30) NOT NULL CHECK (Status IN ('Not Viewed', 'In processing', 'closed')),

ManagerID NUMBER NOT NULL,

CONSTRAINT fk\_UserQuestion FOREIGN KEY (UserID) REFERENCES Users(UserID),

CONSTRAINT fk\_ManagerQuestion\_User FOREIGN KEY (ManagerID) REFERENCES Managers(ManagerID)

);

Листинг А.6 – Таблица вопросов пользователей

# Приложение Б. Листинг процедур

PROCEDURE AddTicketToShoppingCart(

SectorRow\_in IN NUMBER,

NumberOfSeat\_in IN NUMBER,

EventScheduleID\_in IN NUMBER,

Login\_in IN NVARCHAR2,

Message\_out OUT NVARCHAR2

)

AS

UserID\_f NUMBER;

TicketID\_f NUMBER;

Ex\_Invalid\_Ticket\_Exist EXCEPTION;

BEGIN

SELECT UserID INTO UserID\_f FROM TicketVibe\_Users WHERE Login = Login\_in;

SELECT t.TicketID INTO TicketID\_f

FROM TicketVibe\_Tickets t

JOIN TicketVibe\_SectorRows sr ON t.SectorRowID = sr.SectorRowID

JOIN TicketVibe\_EventsSchedule es ON t.EventScheduleID = es.EventScheduleID

WHERE sr.SectorRow = SectorRow\_in

AND t.PlaceInRow = NumberOfSeat\_in

AND es.EventScheduleID = EventScheduleID\_in

AND t.Status = 'On sale';

IF TicketID\_f IS NULL THEN

RAISE Ex\_Invalid\_Ticket\_Exist;

END IF;

INSERT INTO ShoppingCart (UserID, TicketID)

VALUES (UserID\_f, TicketID\_f);

UPDATE TicketVibe\_Tickets SET Status = 'Booked' WHERE TicketID = TicketID\_f;

COMMIT;

Message\_out := 'The ticket has been successfully added to the shopping cart';

EXCEPTION

WHEN Ex\_Invalid\_Ticket\_Exist THEN

Message\_out := 'The ticket was not found';

WHEN NO\_DATA\_FOUND THEN

Message\_out := 'The ticket was not found';

WHEN OTHERS THEN

Message\_out := 'Error during ticket selection';

END;

Листинг Б.1 – Процедура добавления билетов в корзину

PROCEDURE BuyTicket (

UserLogin\_in IN NVARCHAR2,

Message\_out OUT NVARCHAR2

)

IS

max\_SaleID NUMBER;

UserID\_f NUMBER;

TicketsCount\_f NUMBER;

Ex\_Invalid\_Shopping\_Cart EXCEPTION;

Cursor cart\_cursor IS

SELECT TicketID

FROM TicketVibe\_ShoppingCart

WHERE UserID = UserID\_f;

BEGIN

SELECT UserID INTO UserID\_f FROM TicketVibe\_Users WHERE Login = UserLogin\_in;

SELECT COUNT (\*) INTO TicketsCount\_f FROM TicketVibe\_ShoppingCart WHERE UserID = UserID\_f;

IF TicketsCount\_f < 1 THEN

RAISE Ex\_Invalid\_Shopping\_Cart;

END IF;

FOR cart\_rec IN cart\_cursor LOOP

SELECT NVL(MAX(SaleID) + 1, 1000) INTO max\_SaleID FROM TicketVibe\_Sales;

INSERT INTO TicketVibe\_Sales (SaleID, UserID, TicketID, SaleDate, Status)

VALUES (max\_SaleID, UserID\_f, cart\_rec.TicketID, SYSDATE, 'Valid');

UPDATE TicketVibe\_Tickets t

SET Status = 'Purchased'

WHERE TicketID = cart\_rec.TicketID;

END LOOP;

DELETE FROM TicketVibe\_ShoppingCart WHERE UserID = UserID\_f;

COMMIT;

Message\_out := 'Success';

COMMIT;

EXCEPTION

WHEN Ex\_Invalid\_Shopping\_Cart THEN

Message\_out := 'Shopping Cart is empty';

WHEN NO\_DATA\_FOUND THEN

Message\_out := 'No data found';

WHEN OTHERS THEN

Message\_out := 'Error during buying tickets';

END;

Листинг Б.2 – Покупка билетов

PROCEDURE CreateEvent(

EventName\_in IN NVARCHAR2,

Number\_LocationID IN NUMBER,

EventDuration\_in IN NVARCHAR2,

Number\_SubcategoryID IN NUMBER,

OrganizerCompany\_in IN NVARCHAR2,

Description\_in IN NVARCHAR2,

Number\_Cost IN NUMBER,

Message\_Out OUT NVARCHAR2

)

IS

interval\_EventDuration INTERVAL DAY TO SECOND;

OrganizerID\_f NUMBER;

max\_EventID NUMBER;

EventName\_exists NUMBER;

LocationID\_exists NUMBER;

SubcategoryID\_exists NUMBER;

OrganizerID\_exists NUMBER;

Ex\_Invalid\_EventName\_Max EXCEPTION;

Ex\_Invalid\_EventName\_Min EXCEPTION;

Ex\_Invalid\_EventName\_Exist EXCEPTION;

Ex\_Invalid\_EventDuration\_Max EXCEPTION;

Ex\_Invalid\_EventDuration\_Min EXCEPTION;

Ex\_Invalid\_LocationID\_Exist EXCEPTION;

Ex\_Invalid\_OrganizerID\_Exist EXCEPTION;

Ex\_Invalid\_Subcategory\_Exist EXCEPTION;

Ex\_Invalid\_SubcategoryID\_Exist EXCEPTION;

Ex\_Invalid\_TimeStamp EXCEPTION;

Ex\_Invalid\_Interval EXCEPTION;

BEGIN

IF REGEXP\_LIKE(EventDuration\_in, '^\d+ \d{2}:\d{2}:\d{2}$') THEN

interval\_EventDuration := TO\_DSINTERVAL(EventDuration\_in);

ELSE

RAISE Ex\_Invalid\_Interval;

END IF;

SELECT OrganizerID INTO OrganizerID\_f FROM TicketVibe\_Organizers WHERE CompanyName = OrganizerCompany\_in;

SELECT NVL(MAX(EventID) + 1, 1000) INTO max\_EventID FROM TicketVibe\_Events;

SELECT COUNT(\*) INTO EventName\_exists FROM TicketVibe\_Events WHERE EventName = EventName\_in;

SELECT COUNT(\*) INTO LocationID\_exists FROM TicketVibe\_Locations WHERE LocationID = Number\_LocationID;

SELECT COUNT(\*) INTO SubcategoryID\_exists FROM TicketVibe\_Subcategories WHERE SubcategoryID = Number\_SubcategoryID;

SELECT COUNT(\*) INTO OrganizerID\_exists FROM TicketVibe\_Organizers WHERE OrganizerID = OrganizerID\_f;

IF LENGTH(EventName\_in) > 50 THEN

RAISE Ex\_Invalid\_EventName\_Max;

END IF;

IF EventName\_in IS NULL OR LENGTH(EventName\_in) < 4 THEN

RAISE Ex\_Invalid\_EventName\_Min;

END IF;

IF EventName\_exists > 0 THEN

RAISE Ex\_Invalid\_EventName\_Exist;

END IF;

IF LocationID\_exists < 1 THEN

RAISE Ex\_Invalid\_LocationID\_Exist;

END IF;

IF interval\_EventDuration < INTERVAL '0 00:15:00' DAY TO SECOND THEN

RAISE Ex\_Invalid\_EventDuration\_Min;

END IF;

IF interval\_EventDuration > INTERVAL '1 00:00:00' DAY TO SECOND THEN

RAISE Ex\_Invalid\_EventDuration\_Max;

END IF;

IF SubcategoryID\_exists < 1 THEN

RAISE Ex\_Invalid\_SubcategoryID\_Exist;

END IF;

IF OrganizerID\_exists < 1 THEN

RAISE Ex\_Invalid\_OrganizerID\_Exist;

END IF;

IF (Description\_in IS NULL OR LENGTH(Description\_in) < 3) THEN

INSERT INTO TicketVibe\_Events (EventID, EventName, LocationID, EventDuration, SubcategoryID, OrganizerID, Cost)

VALUES (max\_EventID, EventName\_in, Number\_LocationID, interval\_EventDuration, Number\_SubcategoryID, OrganizerID\_f, Number\_Cost);

COMMIT;

END IF;

IF (LENGTH(Description\_in) > 2) THEN

INSERT INTO TicketVibe\_Events (EventID, EventName, LocationID, EventDuration, SubcategoryID, OrganizerID, Cost, Description)

VALUES (max\_EventID, EventName\_in, Number\_LocationID, interval\_EventDuration, Number\_SubcategoryID, OrganizerID\_f, Number\_Cost, Description\_in);

COMMIT;

END IF;

Message\_out := 'The event is created';

EXCEPTION

WHEN Ex\_Invalid\_EventName\_Max THEN

Message\_out := 'Event Name exceeds maximum length 50';

RETURN;

WHEN Ex\_Invalid\_EventName\_Min THEN

Message\_out := 'Event Name less than minimum length 4';

RETURN;

WHEN Ex\_Invalid\_EventName\_Exist THEN

Message\_out := 'The event with the specified Event Name already exist';

WHEN Ex\_Invalid\_LocationID\_Exist THEN

Message\_out := 'There is no Location with this ID';

WHEN Ex\_Invalid\_EventDuration\_Min THEN

Message\_out := 'Event Duration less than minimum: 15 minutes';

WHEN Ex\_Invalid\_EventDuration\_Max THEN

Message\_out := 'Event Duration exceeds maximum: one day';

WHEN Ex\_Invalid\_SubcategoryID\_Exist THEN

Message\_out := 'There is no Subcategory with this ID';

WHEN Ex\_Invalid\_OrganizerID\_Exist THEN

Message\_out := 'There is no Organizer with this ID';

WHEN Ex\_Invalid\_TimeStamp THEN

Message\_out := 'The date is entered in the format YYYY-MM-DD HH24:MI';

WHEN Ex\_Invalid\_Interval THEN

Message\_out := 'The interval is entered in the format 3 10:25:30';

WHEN NO\_DATA\_FOUND THEN

Message\_out := 'Date was not found during event creation';

WHEN OTHERS THEN

Message\_out := 'Error during event creation ';

END;

Листинг Б.3 – Создание мероприятий

PROCEDURE CreateEventSchedule (

EventID\_in IN NUMBER,

EventDate\_in IN TIMESTAMP,

OrganizerCompany\_in IN NVARCHAR2,

Message\_out OUT NVARCHAR2

)

IS

EventScheduleID\_max NUMBER;

OrganizerID\_f NUMBER;

OrganizerID\_exists NUMBER;

v\_overlap\_count NUMBER;

tomorrow TIMESTAMP;

next\_year TIMESTAMP;

v\_event\_start TIMESTAMP;

v\_event\_end TIMESTAMP;

v\_eventDuration INTERVAL DAY TO SECOND;

Ex\_Invalid\_EventDate\_Interval EXCEPTION;

Ex\_Invalid\_EventDate\_Max EXCEPTION;

Ex\_Invalid\_OrganizerID\_Exist EXCEPTION;

Ex\_Invalid\_EventDate\_Min EXCEPTION;

BEGIN

SELECT EventDuration INTO v\_eventDuration FROM TicketVibe\_Events WHERE EventID = EventID\_in;

SELECT OrganizerID INTO OrganizerID\_f FROM TicketVibe\_Organizers WHERE CompanyName = OrganizerCompany\_in;

SELECT COUNT(\*) INTO OrganizerID\_exists FROM TicketVibe\_Events WHERE EventID = EventID\_in AND OrganizerID = OrganizerID\_f;

tomorrow := TRUNC(SYSDATE) + 1;

next\_year := ADD\_MONTHS(TRUNC(SYSDATE), 12);

SELECT NVL(MAX(EventScheduleID) + 1, 1000) INTO EventScheduleID\_max FROM TicketVibe\_EventsSchedule;

IF EventDate\_in < tomorrow THEN

RAISE Ex\_Invalid\_EventDate\_Min;

END IF;

IF EventDate\_in > next\_year THEN

RAISE Ex\_Invalid\_EventDate\_Max;

END IF;

v\_event\_start := EventDate\_in;

v\_event\_end := EventDate\_in + v\_eventDuration;

SELECT COUNT(\*)

INTO v\_overlap\_count

FROM TicketVibe\_EventsSchedule es

JOIN TicketVibe\_Events e ON es.EventID = e.EventID

WHERE e.LocationID = (SELECT LocationID FROM TicketVibe\_Events WHERE EventID = EventID\_in)

AND ((v\_event\_start <= es.EventDate AND v\_event\_end > es.EventDate)

OR (v\_event\_start < es.EventDate + e.EventDuration AND v\_event\_end >= es.EventDate + e.EventDuration)

OR (v\_event\_start < es.EventDate AND v\_event\_end >= es.EventDate + e.EventDuration));

IF v\_overlap\_count > 0 THEN

RAISE Ex\_Invalid\_EventDate\_Interval;

END IF;

IF OrganizerID\_exists < 1 THEN

RAISE Ex\_Invalid\_OrganizerID\_Exist;

END IF;

INSERT INTO TicketVibe\_EventsSchedule (EventScheduleID, EventDate, EventID) VALUES (EventScheduleID\_max, EventDate\_in, EventID\_in);

COMMIT;

EXCEPTION

WHEN Ex\_Invalid\_EventDate\_Max THEN

Message\_out := 'Event Date exceeds maximum date: year later';

WHEN Ex\_Invalid\_EventDate\_Min THEN

Message\_out := 'Event Date less than minimum date: tomorrow';

WHEN Ex\_Invalid\_EventDate\_Interval THEN

Message\_out := 'Location is busy at this time';

WHEN Ex\_Invalid\_OrganizerID\_Exist THEN

Message\_out := 'The event does not belong to this organizer';

WHEN NO\_DATA\_FOUND THEN

Message\_out := 'Data was not found during event schedule creation';

WHEN OTHERS THEN

Message\_out := 'Error during event schedule creation';

END;

Листинг Б.4 – Создание расписания на мероприятия

PROCEDURE ShowCatalog(

timestamp\_start\_date IN TIMESTAMP DEFAULT NULL,

timestamp\_end\_date IN TIMESTAMP DEFAULT NULL,

min\_price\_in IN NUMBER DEFAULT NULL,

max\_price\_in IN NUMBER DEFAULT NULL,

sorttype\_in IN NUMBER DEFAULT 0,

sortorder\_in IN NUMBER DEFAULT 0,

cursor\_out OUT SYS\_REFCURSOR,

Message\_out OUT NVARCHAR2,

location\_in IN NVARCHAR2 DEFAULT NULL,

category\_in IN NVARCHAR2 DEFAULT NULL,

subcategory\_in IN NVARCHAR2 DEFAULT NULL

)

IS

LocationID\_f NUMBER;

CategoryID\_f NUMBER;

SubcategoryID\_f NUMBER;

BEGIN

IF location\_in IS NOT NULL THEN

SELECT LocationID INTO LocationID\_f FROM TicketVibe\_Locations WHERE LocationName = location\_in;

END IF;

IF category\_in IS NOT NULL THEN

SELECT CategoryID INTO CategoryID\_f FROM TicketVibe\_Categories WHERE CategoryName = category\_in;

END IF;

IF subcategory\_in IS NOT NULL THEN

SELECT SubcategoryID INTO SubcategoryID\_f FROM TicketVibe\_Subcategories WHERE SubcategoryName = subcategory\_in;

END IF;

IF (sorttype\_in = 0) THEN

OPEN cursor\_out FOR

SELECT \* FROM TicketVibe\_EventsSchedule es

LEFT JOIN TicketVibe\_Events e ON es.EventID = e.EventID

LEFT JOIN TicketVibe\_Locations l ON e.LocationID = l.LocationID

LEFT JOIN TicketVibe\_Subcategories s ON e.SubcategoryID = s.SubcategoryID

LEFT JOIN TicketVibe\_Categories c ON s.CategoryID = c.CategoryID

WHERE (timestamp\_start\_date IS NULL OR TO\_DATE(TO\_CHAR(es.EventDate, 'YYYY-MM-DD'), 'YYYY-MM-DD') >= timestamp\_start\_date)

AND (timestamp\_end\_date IS NULL OR TO\_DATE(TO\_CHAR(es.EventDate, 'YYYY-MM-DD'), 'YYYY-MM-DD') <= timestamp\_end\_date)

AND (location\_in IS NULL OR e.LocationID = LocationID\_f)

AND (category\_in IS NULL OR e.SubcategoryID IN (

SELECT e.SubcategoryID FROM TicketVibe\_Events e

INNER JOIN TicketVibe\_Subcategories sc ON e.SubcategoryID = sc.SubcategoryID

WHERE sc.CategoryID = CategoryID\_f))

AND (subcategory\_in IS NULL OR e.SubcategoryID = SubcategoryID\_f)

AND (min\_price\_in IS NULL OR e.Cost >= min\_price\_in)

AND (max\_price\_in IS NULL OR e.Cost <= max\_price\_in)

ORDER BY

CASE

WHEN sortorder\_in = 0 THEN EventDate END ASC,

CASE

WHEN sortorder\_in = 1 THEN EventDate END DESC;

END IF;

IF (sorttype\_in = 1) THEN

OPEN cursor\_out FOR

SELECT \* FROM TicketVibe\_EventsSchedule es

LEFT JOIN TicketVibe\_Events e ON es.EventID = e.EventID

LEFT JOIN TicketVibe\_Locations l ON e.LocationID = l.LocationID

LEFT JOIN TicketVibe\_Subcategories s ON e.SubcategoryID = s.SubcategoryID

LEFT JOIN TicketVibe\_Categories c ON s.CategoryID = c.CategoryID

WHERE (timestamp\_start\_date IS NULL OR TO\_DATE(TO\_CHAR(es.EventDate, 'YYYY-MM-DD'), 'YYYY-MM-DD') >= timestamp\_start\_date)

AND (timestamp\_end\_date IS NULL OR TO\_DATE(TO\_CHAR(es.EventDate, 'YYYY-MM-DD'), 'YYYY-MM-DD') <= timestamp\_end\_date)

AND (location\_in IS NULL OR e.LocationID = LocationID\_f)

AND (category\_in IS NULL OR e.SubcategoryID IN (

SELECT e.SubcategoryID FROM TicketVibe\_Events e

INNER JOIN TicketVibe\_Subcategories sc ON e.SubcategoryID = sc.SubcategoryID

WHERE sc.CategoryID = CategoryID\_f))

AND (subcategory\_in IS NULL OR e.SubcategoryID = SubcategoryID\_f)

AND (min\_price\_in IS NULL OR e.Cost >= min\_price\_in)

AND (max\_price\_in IS NULL OR e.Cost <= max\_price\_in)

ORDER BY

CASE

WHEN sortorder\_in = 0 THEN EventName END ASC,

CASE

WHEN sortorder\_in = 1 THEN EventName END DESC;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

Message\_out := 'Data was not found during catalog showing';

WHEN OTHERS THEN

Message\_out := 'Error during catalog showing';

END;

Листинг Б.5 – Вывод каталога мероприятий

# Приложение В. Листинг триггеров

TRIGGER AFTER\_UPDATE\_SECTORROWS

AFTER UPDATE OF NumberOfSeats ON TicketVibe\_SectorRows

FOR EACH ROW

DECLARE

EventScheduleID\_in NUMBER;

SectorRowID\_in NUMBER := :NEW.SectorRowID;

CurrentNumberOfSeats NUMBER := :NEW.NumberOfSeats;

CurrentTicketsCount NUMBER;

Price\_f NUMBER;

Cost\_in NUMBER;

max\_TicketID NUMBER;

BEGIN

FOR event\_rec IN (SELECT DISTINCT EventScheduleID

FROM TicketVibe\_Tickets

WHERE SectorRowID = SectorRowID\_in)

LOOP

EventScheduleID\_in := event\_rec.EventScheduleID;

SELECT COUNT(\*) INTO CurrentTicketsCount

FROM TicketVibe\_Tickets

WHERE EventScheduleID = EventScheduleID\_in

AND SectorRowID = SectorRowID\_in;

IF CurrentTicketsCount > CurrentNumberOfSeats THEN

DELETE FROM TicketVibe\_Tickets

WHERE EventScheduleID = EventScheduleID\_in

AND SectorRowID = SectorRowID\_in

AND PlaceInRow < CurrentNumberOfSeats;

ELSIF CurrentTicketsCount < CurrentNumberOfSeats THEN

FOR i IN CurrentTicketsCount + 1..CurrentNumberOfSeats LOOP

SELECT Cost INTO Cost\_in FROM TicketVibe\_Events e

INNER JOIN TicketVibe\_EventsSchedule es ON es.EventID = e.EventID

WHERE es.EventScheduleID = EventScheduleID\_in;

Price\_f := :NEW.CostFactor \* Cost\_in;

SELECT NVL(MAX(TicketID), 1000) INTO max\_TicketID FROM TicketVibe\_Tickets;

INSERT INTO TicketVibe\_Tickets (TicketID, EventScheduleID, Status, Price, SectorRowID, PlaceInRow)

VALUES (max\_TicketID, EventScheduleID\_in, 'On sale', Price\_f, SectorRowID\_in, i);

END LOOP;

END IF;

END LOOP;

END;

Листинг В.1 – Триггер после обновления количества рядов

TRIGGER AFTER\_INSERT\_LOCATION

AFTER INSERT ON TicketVibe\_Locations

FOR EACH ROW

DECLARE

n NUMBER;

max\_SectorRowID NUMBER;

BEGIN

SELECT NVL(MAX(SectorRowID) + 1, 999) INTO max\_SectorRowID FROM TicketVibe\_SectorRows;

SELECT :NEW.NumberOfSectors INTO n FROM dual;

FOR i IN 1..n LOOP

INSERT INTO TicketVibe\_SectorRows (SectorRowID, SectorRow, NumberOfSeats, LocationID, CostFactor)

VALUES (max\_SectorRowID + i, i, 30, :NEW.LocationID, 1);

END LOOP;

END;

Листинг В.2 – Триггер после вставки локации

TRIGGER AFTER\_UPDATE\_EVENT

AFTER UPDATE ON TicketVibe\_Events

FOR EACH ROW

DECLARE

EventID\_f NUMBER;

ScheduleIDs SYS.ODCINUMBERLIST;

BEGIN

IF :OLD.Cost <> :NEW.Cost THEN

EventID\_f := :NEW.EventID;

SELECT EventScheduleID BULK COLLECT INTO ScheduleIDs

FROM EventsSchedule

WHERE EventID = EventID\_f;

FOR i IN 1..ScheduleIDs.COUNT LOOP

UPDATE TicketVibe\_Tickets

SET Price = :NEW.Cost \* (

SELECT sr.CostFactor

FROM SectorRows sr

WHERE sr.SectorRowID = TicketVibe\_Tickets.SectorRowID

)

WHERE EventScheduleID = ScheduleIDs(i)

AND Status = 'On sale';

END LOOP;

END IF;

END;

Листинг В.3 – Триггер после обновления события

TRIGGER AFTER\_UPDATE\_LOCATION

AFTER UPDATE ON TicketVibe\_Locations

FOR EACH ROW

DECLARE

max\_SectorRow NUMBER;

max\_SectorRowID NUMBER;

BEGIN

SELECT NVL(MAX(SectorRowID) + 1, 1000) INTO max\_SectorRowID FROM TicketVibe\_SectorRows;

IF :OLD.NumberOfSectors <> :NEW.NumberOfSectors THEN

IF :OLD.NumberOfSectors > :NEW.NumberOfSectors THEN

DELETE FROM TicketVibe\_SectorRows WHERE LocationID = :NEW.LocationID

AND SectorRow > :NEW.NumberOfSectors;

ELSE

DECLARE

v\_maxRowNumber NUMBER;

BEGIN

SELECT MAX(SectorRow) INTO max\_SectorRow FROM TicketVibe\_SectorRows WHERE LocationID = :NEW.LocationID;

FOR i IN (max\_SectorRow + 1)..:NEW.NumberOfSectors LOOP

INSERT INTO TicketVibe\_SectorRows (SectorRowID, SectorRow, NumberOfSeats, LocationID, CostFactor)

VALUES (max\_SectorRowID, i, 50, :NEW.LocationID, 1);

max\_SectorRowID := max\_SectorRowID + 1;

END LOOP;

END;

END IF;

END IF;

END;

Листинг В.4 – Триггер после обновления локации