УДК 004.85; УДК 004.6

ОСОБЕННОСТИ РЕАЛИЗАЦИИ БАЗЫ ДАННЫХ СИСТЕМЫ УПРАВЛЕНИЯ КОРПОРАТИВНОЙ ПАРКОВКОЙ С ИСПОЛЬЗОВАНИЕМ МАШИННОГО ОБУЧЕНИЯ

А. И. СЫЧ, М. А. ШМУРАДКО, В. А. БУРАЧЁНОК
(Представлено: канд. тех. наук, доц. И. Б. БУРАЧЁНОК)

В статье рассматриваются ключевые аспекты разработки базы данных для системы управления корпоративной парковкой, функционирующей посредством видеонаблюдения и машинного обучения. Демонстрируется подключение. Указаны особенности построения запросов к базе данных для обеспечения эффективного функционирования системы.

Ключевые слова: *умная парковка, база данных, парковочное место, управление данными, машинное обучение, нейронные сети, видеонаблюдение.*

Введение. Одной из проблем современного города является организации парковочных мест для все увеличивающегося транспортного потока автотранспортных средств в городских районах. Недостаток свободных мест, неэффективное использование имеющихся площадей для парковки автомобилей привело к изменению подходов к организации парковочного пространства, в том числе и на корпоративных парковках. Безусловно парковки с использованием видеонаблюдения или так называемые умные парковки значительно упрощают контроль за свободными местами, за счет учета движения транспорта и предотвращения неправильной парковки. Однако организация автоматизированного контроля за въездом, выездом автотранспортных средств, занятыми местами на парковке, за владельцами, имеющими доступ в парковочную зону, требует создания базы данных, которая обеспечит надежное хранение информации о доступных парковочных местах, автомобиле и его владельце, истории посещения парковки.

Цель работы анализ особенностей реализации базы данных системы управления корпоративной парковкой с использованием машинного обучения.

База данных системы управления корпоративной парковкой с использованием машинного обучения имеет схему представленную на рисунке 1.

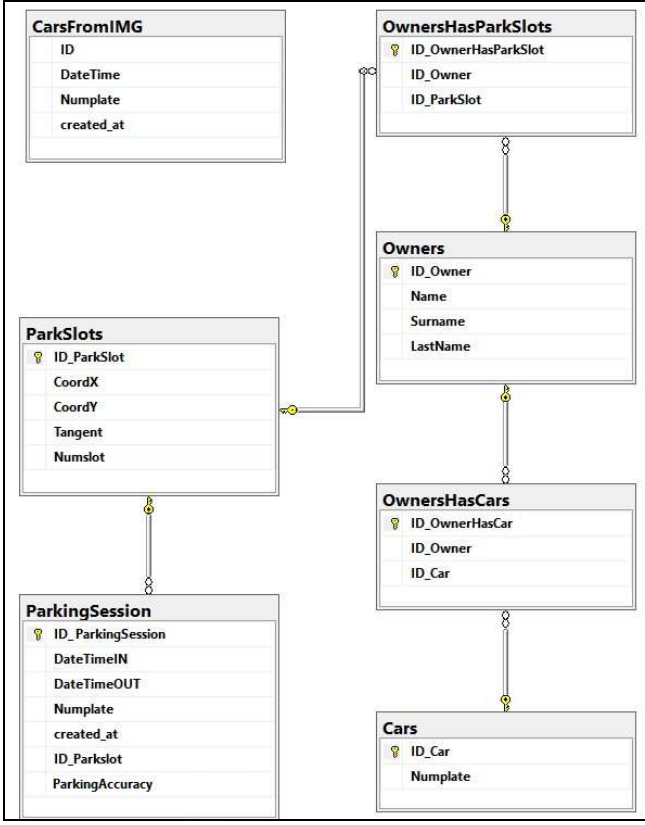

Таблица владельцев (Owners):
Поля: ID_Owner, Name, Surname, LastName.

Таблица автомобилей (Cars):
Поля: ID_Car, Numplate.

Таблица парковочных мест (ParkSlots):
Поля: ID_ParkSlot**,** CoordX, CoordY, Tangent, Numslot.

Таблица парковочных сессий (ParkingSession):
Поля: ID_ParkingSession, DateTimeIN, DateTimeOUT, Numplate, created_at, ID_Parkslot, ParkingAccuracy.

Таблица связей владельцев и автомобилей (OwnersHasCars):
Поля: ID_OwnerHasCar, ID_Owner, ID_Car.

Таблица связей владельцев и парковочных мест (OwnersHasParkSlots):
Поля: ID_OwnerHasParkSlot, ID_Owner, ID_ParkSlot.

Рисунок 1. – Схема реляционной базы данных системы управления корпоративной парковкой

В таблице `Owners` хранятся данные о владельцах. В таблице `Cars` хранятся данные об автомобилях. Таблица `ParkSlots` предназначена для хранения актуальной информации о состоянии парковочных мест (занято/свободно). В таблице `ParkingSession` хранится информации о действиях на парковке, включая время начала и окончания сессий и статус платежа. Таблицы `OwnersHasCars` и `OwnersHasParkslots` связующими соответственно между владельцами и их автомобилями и владельцамии их парковочными местами.

Подключение к базе данных и функции работы с данными владельцев автомобилей реализованы на языке Python с использованием библиотеки **pyodbc** для взаимодействия с базой данных и **tkinter** для создания графического интерфейса пользователя.

Для начала работы с базой данных необходимо установить соединение. В листинге 1 приведен код для подключения к серверу и базе данных. Использование библиотеки **pyodbc** позволяет установить безопасное и стабильное соединение с SQL Server. Важно, чтобы параметры подключения, такие как имя сервера, имя базы данных, учетные данные пользователя, были корректно настроены, что обеспечивает доступ к необходимым данным.

Листинг 1 – Подключение к базе данных

```
server = 'XL23'
database = 'ParkSpotter'
conn = pyodbc.connect('DRIVER={SQLServer};
SERVER='+server+';DATABASE='+database+';Trusted_Connection=yes;')
cursor = conn.cursor()
```

Функция `fetch_owners` отвечает за отображение списка владельцев автомобилей в указанном элементе интерфейса (`owners_listbox`). Она выполняет SQL-запрос для получения всех записей из таблицы `Owners` и отображает их в списке. Функция сначала очищает список `owners_listbox`, затем выполняет запрос для получения всех записей из таблицы `Owners`, и, наконец, добавляет полученные данные в список. В случае возникновения ошибки она выводит сообщение об ошибке. Пример в листинге 2 демонстрирует реализацию этой функции. Важно отметить, что корректная обработка ошибок позволяет избежать сбоев в работе приложения и предоставляет пользователю понятную информацию о возможных проблемах.

Листинг 2 – Реализация функции отображения владельцев

```
def fetch_owners(owners_listbox):
owners_listbox.delete(0, tk.END)
sql_query = 'SELECT * FROM Owners'
cursor.execute(sql_query)
records = cursor.fetchall()
for row in records:
owners_listbox.insert(tk.END, str(row[1]+" "+row[2]+" "+row[3]))
```

Функция `add_owners` позволяет добавлять новых владельцев в базу данных. Она принимает данные из виджета, в котором введено имя владельца, разбивает их на составляющие и вставляет новую запись в таблицу `Owners`. Функция получает данные о владельце, выполняет SQL-запрос для добавления новой записи в таблицу, и затем обновляет список владельцев. В случае ошибки выводится сообщение об ошибке. Реализация представлена в листинге 3. Эта функция является примером важной операции создания данных, которая должна быть безопасной и надежной, чтобы предотвратить потерю или повреждение данных.

Листинг 3 – Реализация функции добавления владельцев

```
def add_owners(owners_listbox, selected_owner_display):
owner_name = selected_owner_display.get()
first_name, surname, last_name = owner_name.split()
sql_query = "INSERT INTO Owners ([Name], Surname, LastName)
VALUES (?, ?, ?)"
cursor.execute(sql_query, (first_name, surname, last_name))
conn.commit()
fetch_owners(owners_listbox)
```

Функция, отвечающая за обновление данных владельцев, включает в себя получение идентификатора владельца, извлечение обновленных данных из соответствующих полей ввода и выполнение SQL-запроса для обновления записи в таблице `Owners`. Обновление данных является критически важной функцией для поддержания актуальности информации в базе данных. Кроме того, интерфейс должен позволять пользователю легко изменять существующие записи, обеспечивая интуитивно понятный и удобный процесс.

Функция `delete_owners` удаления записи о владельце из базы данных также реализована с использованием **pyodbc**. Эта функция получает данные о владельце, выполняет SQL-запрос для удаления соответствующей записи в таблице, и затем обновляет список владельцев. В случае ошибки выводится сообщение об ошибке. Пример в листинге 4 демонстрирует, как можно эффективно управлять удалением данных в базе данных **ParkSpotter**. Удаление данных является не менее важной операцией, так как оно позволяет поддерживать базу данных в актуальном состоянии, избавляясь от ненужных или устаревших записей.

Листинг 4 – Реализация функции удаления владельцев

```
1. def delete_owners(owners_listbox, selected_owner_display):
2. owner_name = selected_owner_display.get()
3. first_name, surname, last_name = owner_name.split()
4. sql_query =
5. "DELETE FROM Owners
6. WHERE [Name] = ? AND Surname = ? AND LastName = ?"
7. cursor.execute(sql_query, (first_name, surname, last_name))
8. conn.commit()
9. fetch_owners(owners_listbox)
```

Перечисленные выше функции обеспечивают основные операции CRUD (создание, чтение, обновление, удаление) для таблицы владельцев в базе данных **ParkSpotter**, позволяя пользователям приложения эффективно управлять данными. Реализация данных функций должна быть оптимизирована для быстрого выполнения запросов и минимизации нагрузки на сервер базы данных.

Для создания новой сессии парковки используется функции приведенная в листинге 5.

Листинг 5 – Создания новой сессии парковки

```
1. def create_parking_session(user_id, spot_id):
2. sql_query = """
3. INSERT INTO parking_sessions (user_id, spot_id, start_time,
4. payment_status)
5. VALUES (?, ?, GETDATE(), 'pending')"""
6. cursor.execute(sql_query, (user_id, spot_id))
7. conn.commit()
```

А для завершения сессии парковки и обновления статуса парковочного места используется функция, представленная в листинге 6.

Листинг 6 - Завершение сессии парковки и обновления статуса парковочного места

```
1. def end_parking_session(session_id, spot_id):
2. update_session_query = """
3. UPDATE parking_sessions
4. SET end_time = GETDATE(), payment_status = 'completed'
5. WHERE session_id = ?
6. cursor.execute(update_session_query, (session_id,))
7. update_spot_query = """
8. UPDATE parking_spots
9. SET status = 'available', last_updated = GETDATE()
10. WHERE spot_id = ?
11. cursor.execute(update_spot_query, (spot_id,))
12. conn.commit()
```

Разработанное программное обеспечение позволяет добавлять новые данные о свободных местах напрямую через алгоритмы видеонаблюдения, которые анализируют потоки автомобилей и автоматически обновляют статус парковочных мест непосредственно в базе данных. При этом важным становится реализация триггеров, которые будут автоматически запускать обновление состояния мест в зависимости

от событий в таблице записей событий входа и выхода автомобилей на парковку. Пример триггера приведен в листинге 7.

Листинг 7 – Обработка событий входа и выхода автомобилей на парковку

```
1.  CREATE TRIGGER CheckPlateInOut
2.  ON ALLTIME
3.  AFTER INSERT
4.  AS
5.  BEGIN
6.  DECLARE @Numplate varchar(15);
7.  DECLARE @Occurrences int;
8.  SELECT @Numplate = Numplate
9.  FROM inserted;
11. SELECT @Occurrences = COUNT(*)
12. FROM ALLTIME
13. WHERE Numplate = @Numplate;
14. IF(@Occurrences % 2 = 0)
15. BEGIN
16. INSERT INTO [OUT] ([DateTime], Numplate)
18. VALUES (GETDATE(), @Numplate);
19. END
20. ELSE
21. BEGIN
22. INSERT INTO [IN] ([DateTime], Numplate)
23. VALUES (GETDATE(), @Numplate); -- Replace 0 with the appropriate
park 24. slot ID
25. END
26. END;
```

Таким образом, интеграция различных технологий и продуманное проектирование обеспечили надежную и эффективную систему, способную справляться с большими объемами данных и предоставлять пользователям необходимые инструменты для управления парковкой. Использование нейросетей в системе требует постоянного обновления и анализа данных, что подразумевает наличие эффективных алгоритмов извлечения и обработки информации. Реализованные запросы к базе данных оптимизированы и позволяют получить быстрый доступ с минимальными задержками при выборе тех или иных данных. База данных протестирована и готова к эксплуатации.

Заключение. Разработка базы данных для системы управления корпоративной парковкой с использованием камер видеонаблюдения выполнена с учетом всех необходимых требований по обеспечению безопасности хранимых данных. Построенные запросы обеспечивают автоматизацию эффективного управления данными и оптимизируют процессы, связанные с контролем за свободными парковочными местами и предотвращением неправильной парковки сотрудниками корпорации. Система исключает несанкционированный заезд на территорию парковки посторонних автомобилей или пропуск автомобилей по чужому электронному пропуску.

ЛИТЕРАТУРА

1. Документация PYODBC. [Электронный ресурс]. – Режим доступа: https://pypi.org/project/pyodbc/ – Дата доступа: 01.05.2024.
2. Официальный сайт Microsoft SQL Server. [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/ru-ru/sql-server/ – Дата доступа: 01.05.2024.
3. Свиридов, А. В. Проектирование баз данных: учебное пособие. / А. В. Свиридов. – Минск: БГТУ, 2020. – 150 с.
4. Кудрявцев, А. А. Системы управления базами данных: подходы и технологии. / А. А. Кудрявцев. – Москва: Инфра-М, 2019. – 180 с.