МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных для интернет-магазина мебели с использованием технологии Full-Text Search»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лях Д.А.

подпись, дата

**Руководитель**

Преподаватель-стажер \_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ (подпись, дата)

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

подпись дата инициалы и фамилия

Минск 2024

Содержание

Оглавление

[Введение 3](#_Toc166827019)

[1 Постановка задачи 4](#_Toc166827020)

[2 Проектирование базы данных 4](#_Toc166827021)

[2.1 Определение вариантов использования 4](#_Toc166827022)

[2.2 Диаграммы UML, взаимодействие всех компонентов 5](#_Toc166827023)

[2.3 Вывод 6](#_Toc166827024)

[3 Разработка объектов базы данных 6](#_Toc166827025)

[3.1 Разработка таблиц базы данных 6](#_Toc166827026)

[3.2 Разработка индексов базы данных 9](#_Toc166827027)

[3.3 Разработка процедур базы данных 10](#_Toc166827028)

[3.4 Разработка триггеров базы данных 10](#_Toc166827029)

[3.5 Разработка пользователей 11](#_Toc166827030)

[3.6 Вывод 12](#_Toc166827031)

[4 Описание процедур импорта и экспорта 13](#_Toc166827032)

[5 Тестирование производительности 14](#_Toc166827033)

[6 Описание технологии и ее применение в базе данных 15](#_Toc166827034)

[7 Руководство пользователя 17](#_Toc166827035)

[Заключение 18](#_Toc166827036)

[Список используемых источников 19](#_Toc166827037)

[Приложение А 20](#_Toc166827038)

[Приложение Б 21](#_Toc166827039)

[Приложение В 22](#_Toc166827040)

# Введение

Цель данной работы заключается в создании реляционной базы данных для интернет-магазина, которая обеспечивает пользователя доступом к имеющимся на ней товарам.

База данных - это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД SSMS, поскольку эта система обладает высокой надежностью и производительностью, что позволяет обеспечить эффективное хранение, обработку и управление данными.

Для обеспечения функциональности приложения используется Полнотекстовый поиск по базе данных.

Основные требования к приложению:

* Реализация ролей администратора и пользователя;
* поиск товаров;
* добавление товаров в каталог;
* взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

# 1 Постановка задачи

Важным этапом разработки любого нового продукта является анализ существующих на рынке аналогов. Он позволяет определить, какие решения уже представлены потребителям, выявить их сильные и слабые стороны, а также определиться с направлениями для усовершенствований.

Интернет-магазин – это форма торговли, при которой продавец предлагает на продажу товар или услугу, а покупатели оформляют заказ онлайн широко используются в различных отраслях и имеют значительную актуальность. В бизнесе интернет-магазины используются для продажи различных товаров (Программное обеспечение, техника, продукты, и др.).

На текущий момент рынок интернет-магазинов представляет собой множество платформ с различным уровнем функционала и безопасности. Однако, многие из них не могут предложить полноценного комплекса инструментов, сочетающего в себе удобство, гибкость и высокие стандарты безопасности. Это оставляет простор для новых разработок в этой сфере.

Задача проекта: разработать архитектуру приложения, взаимодействие с которой будет понятно любому пользователю. Построить базу данных и выполнить тестирование готового продукта.

Должны быть выполнены следующие требования:

˗ база данных должна быть реализована в СУБД SSMS.

˗ доступ к данным должен осуществляться только через соответствующие процедуры;

˗ должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON;

˗ необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице;

˗ применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.

# 2 Проектирование базы данных

## 2.1 Определение вариантов использования

Главное начало функциональных требований базы данных заключается в определении способа обработки данных и обеспечении необходимой функциональности пользовательскому интерфейсу. Важно описать, как данные должны быть хранены и организованы, как осуществляется поиск и выборка данных, как обновляются данные и как обеспечивается безопасность информации. Кроме того, следует учесть взаимодействие базы данных с другими системами и программами. Например, для интернет-магазина функциональные требования могут включать в себя функции для хранения информации аукционах и их лотах.

Кроме функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде Use Case диаграмм, которые описывают, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. На рисунке 2.1 представлена UML диаграмма вариантов использования.



Рисунок 2.1– UML диаграмма вариантов использования

Покупатель(Client) имеет возможность просматривать список имеющихся товаров и оформлять заказ на них, а также оформлять возврат при отказе от товара

## 2.2 Диаграммы UML, взаимодействие всех компонентов

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.2.

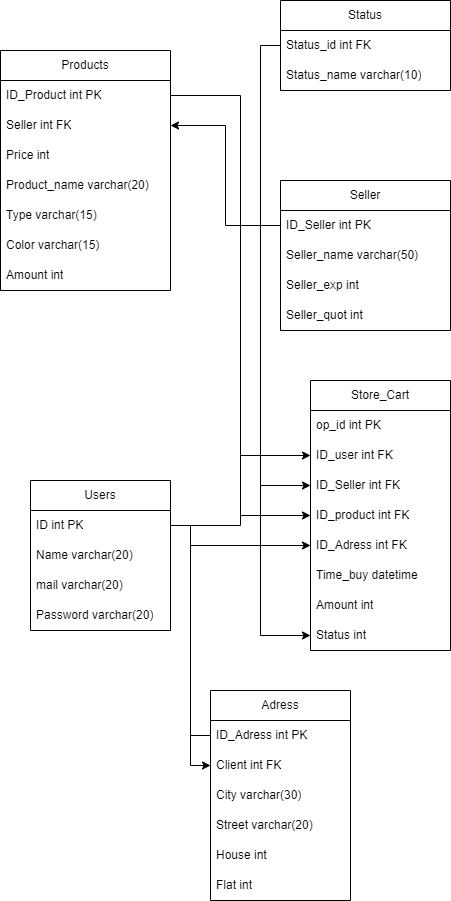


Рисунок 2.2 – Диаграмма базы данных

Видно, что диаграмма показывает связи между таблицами и полями, а также отношения между ними. Одна из основных связей – это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице.

## 2.3 Вывод

Такая архитектура базы данных предоставляет гибкость в управлении данными, высокую производительность и безопасность. Структурированное разделение на авторизацию и доменную логику позволяет оптимально использовать ресурсы системы и обеспечивать эффективное взаимодействие с данными.

# 3 Разработка объектов базы данных

## 3.1 Разработка таблиц базы данных

При разработке приложения для курсового проекта была использована база данных SSMS.

Для реализации базы данных было разработано 6 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Пользователи, Продавцы, Товары, Корзина покупок, Статус товара, Адрес

Таблица Пользователи содержит информацию о пользователях. Таблица представлена в примере 3.1

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID | Индетификатор пользователя | int |
| Name | Имя пользователя | varchar(50) |
| Email | Электронная почта, Уникальный идентификатор пользователя, первичный ключ | varchar(30) |
| password | Пароль пользователя | varchar(20) |

Таблица 3.1-Пользователи

Таблица продавцов содержит информацию о продавцах: их имя,опыт работы и рабочую квоту. Таблица представлена в примере 3.2

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Seller | Индетификатор продавца | int |
| Seller\_Name | Имя продавца | varchar(50) |
| Seller\_exp | Опыт продавца | int |
| Seller\_quot | Квота продавца | int |

Таблица 3.2-Продавцы

Таблица товаров содержит информацию о товарах. Таблица представлена в примере 3.3

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Product | Индетификатор продавца | int |
| Seller | ID продавца | int |
| Price | Цена | int |
| Amount | Количество | int |
| Product\_name | Название | Varchar(20) |
| Type | Тип продукта | Varchar(15) |
| Color | Цвет продукта | Varchar(20) |

Таблица 3.3-Товары

Таблица корзина покупок содержит информацию о покупках. Таблица представлена в примере 3.4

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Product | Индетификатор продукта | int |
| ID\_Seller | ID продавца | int |
| ID\_Adress | адрес | int |
| Amount | Количество | int |
| ID\_user | Пользователь | int |
| Time\_buy | Время покупки | datetime |
| Stat | Статус | int |

Таблица 3.3-Корзина

Таблица статус товара содержит информацию о статусе товара. Таблица представлена в примере 3.5

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Status | Индетификатор статуса | int |
| Status | Наименование статуса | Varchar(15) |

Таблица 3.5-Статус

Таблица Адреса содержит информацию о адресе пользователя. Таблица представлена в примере 3.6

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| ID\_Adress | Индетификатор адреса | int |
| Client | ID клиента | int |
| City | Город | Varchar(20) |
| Street | Улица | Varchar(20) |
| House | Дом | int |
| Flat | Квартира | Int |

Таблица 3.6-Адрес

Перед созданием таблиц, для их структуризации и изоляции было принято решение использовать файловые группы.

Файловая группа – это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания файловых групп будет представлен в листинге 3.1.

USE master;

GO

ON PRIMARY

( NAME='Mebel\_shop\_Primary',

FILENAME=

'c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\data\Mebel\_shop\_Prm.mdf',

SIZE=4MB,

MAXSIZE=10MB,

FILEGROWTH=1MB),

FILEGROUP Mebel\_shop\_FG1

( NAME = 'Mebel\_shop\_FG1\_Dat1',

FILENAME =

'c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\data\Mebel\_shop\_FG1\_1.ndf',

SIZE = 1MB,

MAXSIZE=10MB,

FILEGROWTH=1MB)

Листинг 3.1 – Cкрипт для создания файловых групп

## 3.2 Разработка индексов базы данных

Индекс в базе данных представляет собой объект, который используется для ускорения поиска данных. Если таблица содержит большое количество строк, то последовательный поиск данных может занимать много времени. Индекс создается на основе значений одного или нескольких столбцов таблицы и указывает на соответствующие строки таблицы. Использование индексов помогает улучшить производительность базы данных, поскольку они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Один из наиболее распространенных алгоритмов индексации – это B-дерево (B-tree). B-дерево – это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе.

Для быстрого поиска товаров были добавлены индексы на следующие столбцы: Product\_name, Price, Seller в таблице Products, Seller\_name в таблице Seller и ID\_user,Time\_buy в таблице Store\_Cart. Создание индексов представлено в листинге 3.2.

CREATE INDEX PRODUCT\_INDEX ON Products (Product\_name,Price)

CREATE INDEX SELLER\_INDEX ON Products(Seller)

CREATE INDEX SELLER\_NAME\_INDEX ON Seller(Seller\_name)

CREATE INDEX CART\_INDEX ON Store\_Cart (ID\_user,Time\_buy)

CREATE INDEX AMOUNT\_INDEX ON Products (Amount)

Листинг 3.3 – Индексы таблиц

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах. Поэтому необходимо балансировать количество и типы индексов, чтобы обеспечить оптимальную производительность базы данных. А также использовать индексы на значениях, которые изменяются редко или вообще не меняются.

## 3.3 Разработка процедур базы данных

Подпрограммы PL / SQL называются блоками PL / SQL, которые могут быть вызваны с помощью набора параметров. PL / SQL предоставляет два вида подпрограмм: функции и процедуры. Процедуры не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. На листинге 3.3 представлен пример реализации одной из процедур.

CREATE PROCEDURE ADD\_PRODUCT(@id int,@seller int,@price int,@product\_name varchar(50),@color varchar(20),@type varchar(20),@amount int)

AS

BEGIN

INSERT Products values (@id,@seller,@price,@product\_name,@color,@type,@amount);

END;

Листинг 3.3 – Процедура добавления товара

Данная процедура производит добавление товара в каталог. Правильность введенных данных коррелируется триггером на вставку значений.

## 3.4 Разработка триггеров базы данных

Триггер в базе данных – это объект, который автоматически выполняет определенные действия при возникновении определенных событий в таблице или представлении базы данных. Триггер может быть настроен на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для поддержки целостности данных, контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице.

в листинге 3.4 приведен код триггера, уведомляющего о добавлении товара.

use Mebel\_shop

go

CREATE TRIGGER PRODUCT\_INSERT

ON dbo.Products

FOR INSERT

AS

declare @Pric int

SELECT @Pric=Price

FROM inserted

IF @Pric<=0

Begin

print 'Incorrect data. Price should be higter that 0'

Rollback transaction

end;

declare @AMN int

SELECT @AMN=Amount

FROM inserted

IF @AMN<=0

Begin

print 'Incorrect data. Amount should be higter that 0'

Rollback transaction

end;

declare @id int

SELECT @id=ID\_product

FROM inserted

IF @id<=0

Begin

print 'Incorrect data. ID should be higter that 0'

Rollback transaction

end;

declare @sel int

SELECT @sel=Seller

FROM inserted

IF @sel<=0

Begin

print 'Incorrect data. Seller should be higter that 0'

Rollback transaction

end;

Листинг 3.4 – Скрипт триггера Add\_Trigger

Данный триггер создается в таблице Products и сообщает о добавлении товара.

Также был реализован триггер для проверки изменяемых значений. Это позволяет избежать внесения некорректных данных таких как ввод отрицательных значений. Его пример представлен в Приложении В.

## 3.5 Разработка пользователей

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 3 типа пользователей для подключения к базе данных: покупатель, продавец и разработчик.

Создание пользователей представлено в листинге 3.5.

CREATE LOGIN seller WITH PASSWORD ='12345678xD';

CREATE USER Seller FOR LOGIN seller;

CREATE LOGIN client WITH PASSWORD ='12345678xD';

CREATE USER Client FOR LOGIN client;

Листинг 3.5 – Создание пользователей

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала клиента.

## 3.6 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для интернет-магазина компьютерных комплектующих. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

# 3.6 разработка представлений

Представление в базе данных – это подмножество базы данных, основанное на запросе, который запущен для одной или нескольких таблиц базы данных. Представления базы данных хранятся в базе данных как именованные запросы и могут использоваться для сохранения часто используемых сложных запросов. Представления позволяют объединять данные из нескольких таблиц в одну, не изменяя исходных таблиц и их структуры. Представления используются для обеспечения удобства доступа и управления данными в базе данных, а также для изоляции пользователя от оригинальных таблиц. В данном проекте были созданы представления:

Product\_catalog- показывает весь каталог товаров

Clients-показывает текущих зарегистрированных клиентов

STORE- показывает текущие покапки в корзине

Реализация представлений показана в рисунке 3.6

USE Mebel\_shop

GO

CREATE VIEW Product\_catalog

AS

SELECT

Product\_name,Price,Type,Color,Amount,Seller\_name FROM Products INNER JOIN Seller ON Products.Seller=Seller.ID\_Seller

CREATE VIEW Clients

AS SELECT

ID,Name,mail FROM Users

CREATE VIEW STORE

AS SELECT

Name,Seller\_name,Product\_name,ID\_Adress,Time\_buy,Stat FROM Store\_Cart INNER JOIN Products ON Store\_Cart.ID\_Product=Products.ID\_Product INNER JOIN Users ON Store\_Cart.ID\_user=Users.ID INNER JOIN Seller ON Store\_Cart.ID\_Seller=Seller.ID\_Seller

Рисунок 3.6- представления базы данных

# 4 Описание процедур импорта и экспорта

Для таблицы Products в базе данных доступна функция экспорта и импорта данных в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана процедура, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат. Скрипт импорта представлен в листинге 4.1.

SELECT [ID\_Seller]

,[Seller\_name]

,[Seller\_exp]

,[Seller\_quot]

FROM [dbo].[Seller]

FOR JSON PATH,ROOT('Sellers');

GO

Листинг 4.1 – Скрипт импорта

Для обратного экспорта данные в JSON формат из файла, реализован запросом, который обращается к JSON, и выбирает данные в обычные строки, это можно реализовать с помощью функции, либо по мере надобности записать данные во временную таблицу, либо таблицу в другой базе данных.

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах xml, что делает управление базой данных более удобным и эффективным.

# 5 Тестирование производительности

Одной из ключевых задач в процессе разработки является тестирование производительности базы данных. Такое тестирование позволяет оценить способность базы данных обрабатывать запросы и возвращать результаты с высокой скоростью. Оценка производительности важна для определения эффективности базы данных и выявления возможных узких мест, которые могут вызывать задержки или проблемы в работе приложения.

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данной задачи мы создали анонимный блок, и так как работаем именно с действующей базой данных, то можем использовать готовую процедуру для вставки данных в таблицу. Разработанный анонимный блок позволяет добавить большое количество строк за одно выполнение (листинг 5.1).

USE Mebel\_shop

go

DECLARE @Cou int SET @Cou=0

DECLARE @id int SET @id=1

DECLARE @seller int SET @seller=1

DECLARE @price int SET @price=100

DECLARE @product\_name varchar(20) SET @product\_name='Big Bed'

DECLARE @type varchar(10) SET @type='RoomMebel'

DECLARE @color varchar(10) SET @color='Red'

DECLARE @amount int SET @amount=150

WHILE (@Cou<=100000)

BEGIN

SET @id=@id+1

SET @seller=@seller+1

SET @price=@price+100

SET @product\_name=@product\_name+CAST(@id as varchar)

SET @type=@type+CAST(@id as varchar)

SET @color=@color+CAST(@id as varchar)

SET @amount=@amount+5

INSERT Products values (@id,@seller,@price,@product\_name,@type,@color,@amount);

SET @Cou=@Cou+1

if @seller=3 SET @seller=1

END

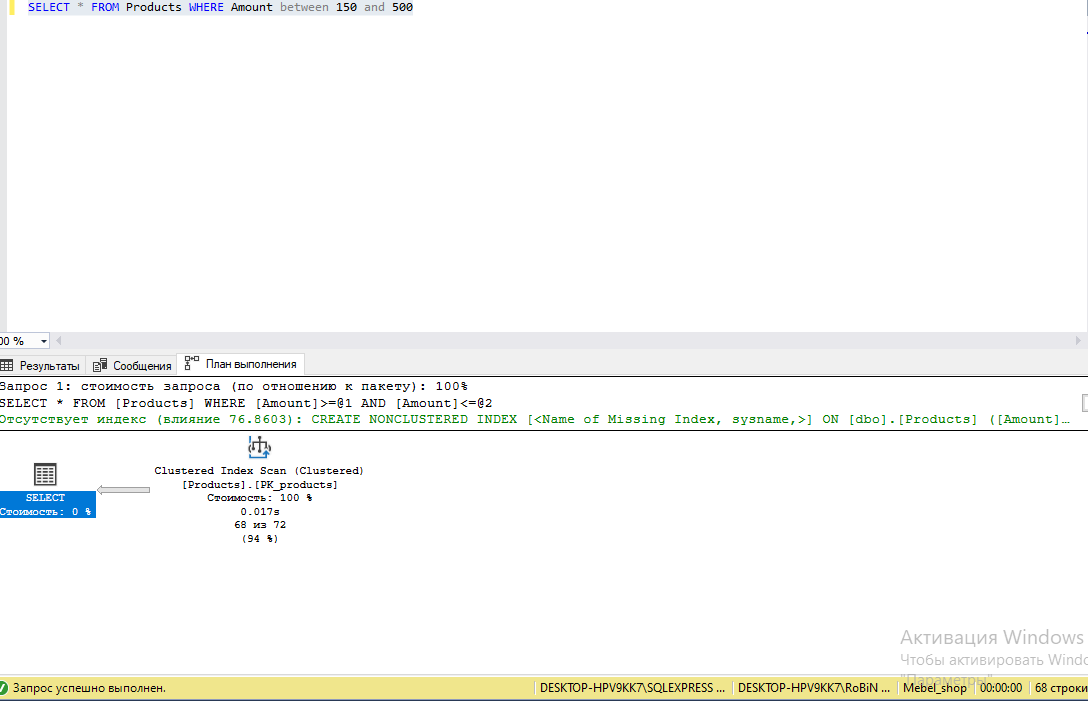
Листинг 5.1 – Заполнение таблицы 100000 строк

Для получения выборки данных использовался запрос, который представлен на листинге 5.2.

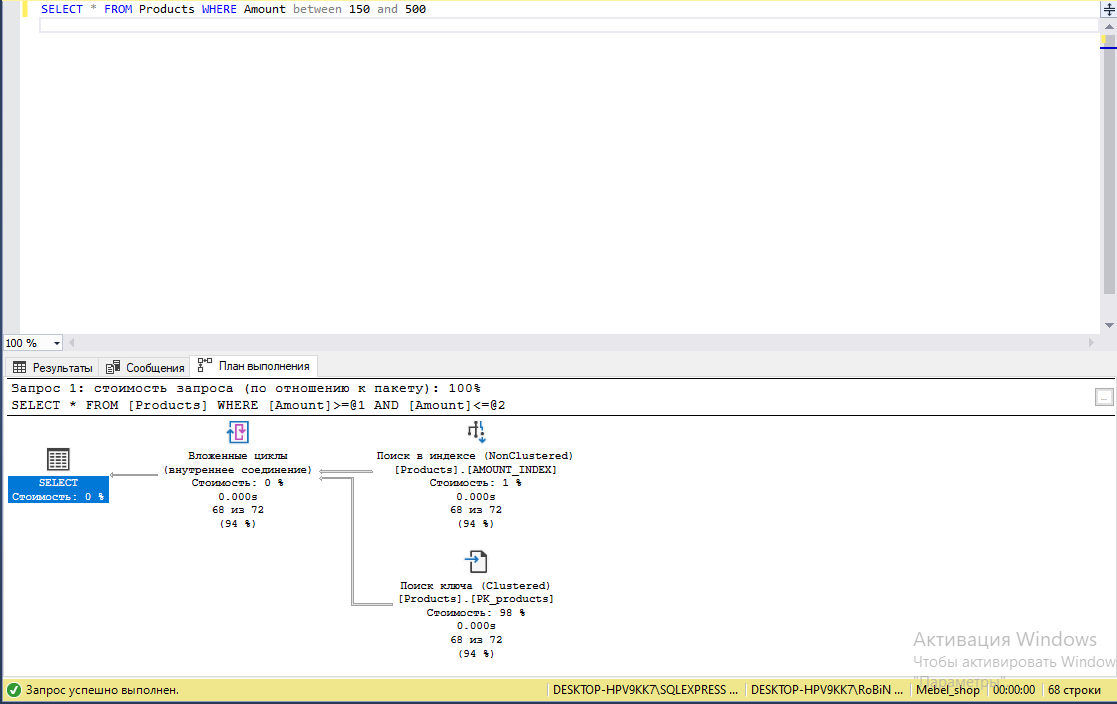
SELECT \* FROM Products WHERE Amount between 150 and 500

Листинг 5.2 – Запрос к таблице Products

Результаты запроса к таблице показывают что на поиски затрачивается время и ресурсы сервера, при учете что при выполнении запроса происходит сканирование всей таблицы и приминение фильтра. Время планирования- 17мс. Результат запроса представлен на рисунке 5.3

 Листинг 5.3 – Результат выполнения запроса

Для такого быстрого поиска был добавлен индекс на столбец Amount. Тестирование запроса с использованием индекса приведено на рисунке 5.4



Листинг 5.4 – Результат выполнения запроса

В результате видно что добавление индекса значительно ускорило время выполнения запроса при применении фильтрации.

# 6 Описание технологии и ее применение в базе данных

Полнотекстовый поиск в базе данных представляет собой использование полнотекстового индекса столбцов таблицы. Полнотекстовые запросы выполняют лингвистический поиск по текстовым данным в полнотекстовых индексах, оперируя словами и фразами на основе правил определенного языка, например английского или русского. Полнотекстовые запросы могут включать простые слова и фразы или несколько форм слова или фразы. После добавления столбцов в полнотекстовый индекс пользователи и приложения могут выполнять полнотекстовые запросы к тексту в столбцах.

На рисунке 6.1 показана процедура поиска с использованием полнотекстового поиска.

CREATE PROCEDURE SEARCH\_PRODUCT(@product\_name varchar(50))

AS

BEGIN

SELECT Seller,Product\_name,Price,Color,Type,Amount FROM Products WHERE CONTAINS(Product\_name,@product\_name)

END;

Рисунок 6.1 – Процедура полнотекстового поиска

Но перед создания полнотекстового запроса необходимо создать полнотекстовый индекс для обеспечения поиска по столбцам. На рисунке 6.2

Показан запрос на создание полнотекстового индекса

CREATE FULLTEXT CATALOG product\_catalog;

go

CREATE FULLTEXT INDEX ON Products(

Product\_name)

KEY INDEX PK\_Products ON product\_catalog;

Рисунок 6.1 – Скрипт создания полнотекстового индекса

Применение технологии полнотекстового ввода при помозщи создания понотекстового индекса обладает рядом преимуществ:

1. **FTS**исполняет индексацию каждого слова отдельно, обеспечивая быстрый и эффективный доступ к текстовым данным. Благодаря интегрированному лексическому и морфологическому анализу, полнотекстовый поиск обнаруживает не только точные совпадения, но и различные формы слов и синонимы.
2. FTS показывает лучшую производительность на индексированных столбцах, сочетая в себе высокую точность и эффективность. Это особенно важно при работе с большими текстовыми базами данных. В сравнении Оператор LIKE проигрывает по эффективности, если столбцы не проиндексированы или используются в начале шаблона символы подстановки.
3. FTS эффективно справляется с запросами начинающимися символа подстановки,что очень полезно при выполнении запросов. Для примера тот же оператор LIKE имеет трудности при работе с запросами начинающимися с символа подстановки.
4. FTS очень полезен для интегрирования в будущую разработку проекта. Сочетая в себе высокую эффективность и точность он очень сильно скажется на повышении производительнолсти запросов в дальнейшем росте и усложнении структуры базы данных.

Технология полнотекстового поиска позволяет реализовать эффек-тивный и быстрый поиск по базе данных, а также стать весомой оптимизацией для таблиц базы данных.

# 7 Руководство пользователя

При входе в магазин, мы будем иметь статус клиента и нам будут доступны все стандартные возможности покупателя. Мы сможем посмотреть имеющийся каталог товаров, произвести поиск конкретного, совершить покупку, оформить возврат.

Если вход был произведен с аккаунта продавца, то мы сможем редактировать статус заказа, добавлять товары в ассортимент, выводить их оттуда, возвращать их обратно и изменять любые необходимые данные в каталоге.

Реализация каждой из вышеупомянутой роли приведена в [приложении Д](#_Приложение_Д).

# Заключение

В данном проекте была поставлена задача разработки базы данных для интернет-магазина мебельного магазина с использованием технологии полнотекстового поиска в СУБД SSMS.

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы триггеры, специальные функции, чтобы обеспечить структурированное хранение данных, изолированность данных таблиц от прямого вмешательства и удобный доступ к ним. В итоге, поставленная цель была успешно достигнута, и на выходе получена готовая база данных.

В рамках тестирования базы данных был использован поток данных, и результаты тестирования оказались положительными. Также были разработаны процедуры импорта и экспорта данных в формате JSON, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная база данных для интернет-магазина мебели имеет ряд важных особенностей, которые позволяют эффективно управлять ей. Одной из главных особенностей является технология полнотекстового поиска, которая обеспечивает быстродействие поисковых запросов и сокращение времени на поиск. Кроме того, база данных обладает гибкой системой привилегий, которая позволяет разграничивать доступ к различным функциям и данным в зависимости от роли пользователя. Это обеспечивает безопасность данных и предотвращает несанкционированный доступ.

Делая вывод, можно сказать, что поставленные цели были выполнены. интернет-магазина мебели выполняет необходимую функцию хранения и управления данными.

# Список используемых источников

1. Использование диаграммы вариантов использования UML при проектировании программного обеспечения [Электронный ресурс] – https://habr.com/ru/articles/566218/ – Дата доступа 25.04.2024
2. Microsoft [Электронный ресурс] – https://learn.microsoft.com/en-us/sql/relational-databases/search/full-text-search?view=sql-server-ver16 – Дата доступа 19.04.2024
3. Metanit [Электронный ресурс] – https://metanit.com/sql/sqlserver/11.1.php – Дата доступа 23.04.2024
4. Администрирование баз данных SSMS [Электронный ресурс] – https://metanit.com/sql/sqlserver/1.1.php – Дата доступа 24.04.2024

## Приложение А

CREATE TABLE Products(

ID\_Product int CONSTRAINT [PK\_products] Primary key,

Seller int,

Price int,

Product\_name varchar(20),

Type varchar(15),

Color varchar(15),

Amount int

)

CREATE TABLE Status(

Status\_id int CONSTRAINT [PK\_status] Primary key,

Status\_name varchar(10)

)

CREATE TABLE Seller(

ID\_Seller int CONSTRAINT [PK\_seller] Primary key,

Seller\_name varchar(50),

Seller\_exp int,

Seller\_quot int

)CREATE TABLE Users(

ID int CONSTRAINT [PK\_users] Primary key,

Name varchar(20),

mail varchar,

Password varchar(20)

)

CREATE TABLE Store\_Cart(

op\_id int CONSTRAINT [PK\_cart] Primary key,

ID\_user int,

ID\_Seller int,

ID\_product int,

ID\_Adress int,

Time\_buy datetime,

Amount int,

Stat int,

FOREIGN KEY(ID\_user) REFERENCES Users(ID),

FOREIGN KEY(ID\_Seller) REFERENCES Seller(ID\_Seller),

FOREIGN KEY(ID\_product) REFERENCES Products(ID\_Product),

FOREIGN KEY(ID\_Adress) REFERENCES Adress(ID\_Adress),

FOREIGN KEY(Stat) REFERENCES Status(Status\_id),

Листинг 1- Создание таблиц

## Приложение Б

CREATE PROCEDURE ADD\_PRODUCT(@id int,@seller int,@price int,@product\_name varchar(50),@color varchar(20),@type varchar(20),@amount int)

AS

BEGIN

INSERT Products values (@id,@seller,@price,@product\_name,@color,@type,@amount);

END;

CREATE PROCEDURE UPDATE\_PRODUCT(@id int,@seller int,@price int,@product\_name varchar(50),@color varchar(20),@type varchar(20),@amount int)

AS

BEGIN

if (Select ID\_product FROM Products)!=null

UPDATE Products SET Seller=@seller,Price=@price,Product\_name=@product\_name,Color=@color,Type=@type,Amount=@amount

WHERE ID\_product=@id;

else Print('No goods to update')

END

CREATE PROCEDURE DELETE\_PRODUCT(@name varchar(20))

as

begin

DELETE FROM Products WHERE Product\_name=@name;

end;

CREATE FULLTEXT CATALOG product\_catalog;

go

CREATE FULLTEXT INDEX ON Products(

Product\_name)

KEY INDEX PK\_Products ON product\_catalog;

CREATE PROCEDURE SEARCH\_PRODUCT(@product\_name varchar(50))

AS

BEGIN

SELECT Seller,Product\_name,Price,Color,Type,Amount FROM Products WHERE CONTAINS(Product\_name,@product\_name)

END;

CREATE PROCEDURE RETURN\_PRODUCT(@user int,@Name varchar(20),@amount int)

AS

BEGIN

BEGIN TRY

declare @id\_prod int SET @id\_prod=(SELECT ID\_Product FROM Products WHERE Product\_name=@Name);

UPDATE Store\_Cart SET Stat=2 WHERE ID\_user=@user and ID\_product=@id\_prod

UPDATE Products SET Amount=Amount+@amount WHERE ID\_product=@id\_prod and Product\_name=@Name

print('Product was returned')

END TRY

BEGIN CATCH

print('Incorrect operation')

END CATCH

END

CREATE PROCEDURE BUY\_PRODUCT(@Client\_Name varchar(50),@Name varchar(20),@Color varchar(10),@Amount int)

AS

BEGIN

BEGIN TRY

declare @id\_op int; SELECT @id\_op=count(\*) FROM Store\_Cart

declare @id\_seller int; SELECT @id\_seller=Seller FROM Products Where Product\_name=@Name and Color=@Color

declare @id\_prod int; SELECT @id\_prod= ID\_Product FROM Products Where Product\_name=@Name and Color=@Color

declare @id\_ckin int; SELECT @id\_ckin= ID FROM Users Where Name=@Client\_Name

declare @adress int; SELECT @adress= ID\_Adress FROM Adress Where Client=@id\_ckin

declare @date datetime; SET @date=GETDATE();

INSERT INTO Store\_Cart values(@id\_op,@id\_ckin,@id\_seller,@id\_prod,@adress,@Amount, @date,1);

UPDATE Products SET Amount=Amount-@Amount WHERE Product\_name=@Name and Color=@Color

print ('Product purchased')

END TRY

BEGIN CATCH

print ERROR\_NUMBER();

print ERROR\_SEVERITY();

print ERROR\_STATE();

print ERROR\_PROCEDURE();

print ERROR\_LINE();

print ERROR\_MESSAGE();

END CATCH

End

Листинг 2- Создание процедур

## Приложение В

CREATE TRIGGER PRODUCT\_INSERT

ON dbo.Products

FOR INSERT

AS

declare @Pric int

SELECT @Pric=Price

FROM inserted

IF @Pric<=0

Begin

print 'Incorrect data. Price should be higter that 0'

Rollback transaction

end;

declare @AMN int

SELECT @AMN=Amount

FROM inserted

IF @AMN<=0

Begin

print 'Incorrect data. Amount should be higter that 0'

Rollback transaction

end;

declare @id int

SELECT @id=ID\_product

FROM inserted

IF @id<=0

Begin

print 'Incorrect data. ID should be higter that 0'

Rollback transaction

end;

declare @sel int

SELECT @sel=Seller

FROM inserted

IF @sel<=0

Begin

print 'Incorrect data. Seller should be higter that 0'

Rollback transaction

end;

Create TRIGGER PRODUCT\_DELETE

ON Products

AFTER DELETE

AS

print 'Product was deleted'

Create TRIGGER PRODUCT\_UPDATE

ON Products

for UPDATE

AS

declare @Pric int

SELECT @Pric=Price

FROM inserted

IF @Pric<=0

Begin

print 'Incorrect data. Price should be higter that 0'

Rollback transaction

end;

declare @AMN int

SELECT @AMN=Amount

FROM inserted

IF @AMN<=0

Begin

print 'Incorrect data. Amount should be higter that 0'

Rollback transaction

end;

declare @id int

SELECT @id=ID\_product

FROM inserted

IF @id<=0

Begin

print 'Incorrect data. ID should be higter that 0'

Rollback transaction

end;

declare @sel int

SELECT @sel=Seller

FROM inserted

IF @sel<=0

Begin

print 'Incorrect data. Seller should be higter that 0'

Rollback transaction

end;