МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

База данных “Интернет-магазина мобильных телефонов”

Выполнил студент Круглик Алексей Викторович

(Ф.И.О.)

Руководитель проекта асс. Копыток Д.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс. Копыток Д.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс. Копыток Д.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2022

Содержание

[Введение 3](#_Toc121332368)

[1.Постановка задачи 4](#_Toc121332369)

[1.1. Аналитический обзор 4](#_Toc121332370)

[1.2. Описание используемых технологий 5](#_Toc121332371)

[2.Разработка модели базы данных 7](#_Toc121332372)

[3. Разработка необходимых объектов 9](#_Toc121332373)

[3.1.Таблицы 9](#_Toc121332374)

[3.2.Процедуры 11](#_Toc121332375)

[4. Описание процедур импорта и экспорта 14](#_Toc121332376)

[5. Тестирование производительности 16](#_Toc121332377)

[6. Описание технологии 19](#_Toc121332378)

[6.1.Динамическая маскировка данных 20](#_Toc121332379)

[6.2.Шифрование на уровне столбцов с использованием симметричных ключей 20](#_Toc121332380)

[6.3.Расшифровка данных на уровне столбов 22](#_Toc121332381)

[6.4.Оценка уязвимостей через SSMS 22](#_Toc121332382)

[Заключение 24](#_Toc121332383)

[Список используемых источников 25](#_Toc121332384)

[Приложения A. Листинг создания БД и таблиц 26](#_Toc121332385)

[Приложения B. Листинг процедур 28](#_Toc121332386)

# Введение

Данные всегда играли важную роль в жизни людей и в разные времена их хранили по-разному. В наше время всё больше информации хранится в цифровом виде и для хранения такой информации отлично подходят различные базы данных. Использование базы данных упрощает управление и хранение данных. Одним из примеров успешного внедрения баз данных является использование их различными магазинами.

Сейчас есть множество удобных приложений, позволяющих без особых усилий совершать заказы товаров в магазине, с помощью таких приложений обычно можно совершить полный цикл заказа товара, начиная от выбора фирмы и поставщика по необходимым параметрам и заканчивая оформлением заказа.

Целью моего курсового проекта является создание базы данных сети кинотеатров.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы;
* спроектировать базу данных;
* реализовать функциональность базы данных;
* провести тестирование используемой технологии в базе данных;
* разработать приложение для работы с базой данных;
* написать руководство пользователя.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

Реляционная база данных — база данных, основанная на реляционной модели данных.

В качестве СУБД для базы данных была выбрана «Microsoft SQL Server», в связи с ее простотой, производительностью и надежностью.

# 1.Постановка задачи

В соответствии с заданием курсового проекта следует создать базу данных для сети магазина. Для того чтобы сформировать окончательные требования к проектируемому программному средству сначала рассмотрим аналоги из той же области, а так же технологии, которые используются в данном проекте.

# 1.1. Аналитический обзор

Раньше, для администрирования магазинов, вели бухгалтерский учет, что осложняло весь процесс. Сейчас же используются базы данных с разработанным для них программным обеспечением, позволяя уменьшить объем бумажных работ и оптимизировать работу всей системы.

Постепенно с развитием программного обеспечения ЭВМ появились идеи создания управляющих систем, которые позволяли бы накапливать, хранить и обновлять взаимосвязанные данные по целому комплексу решаемых задач. Эти идеи нашли свое воплощение в системах управления базами данных (СУБД). СУБД взаимодействуют не с локальными, а взаимосвязанными по информации массивами, называемыми базами данных. С появлением персональных компьютеров СУБД становятся наиболее популярным средством обработки табличной информации. Они являются инструментальным средством проектирования банков данных при обработке больших объемов информации.

Можно рассмотреть структуру организации работы магазинов среди приложений, ориентированных на заказ телефонов. Рассмотрим интернет-приложение для сети магазинов «Apple Store».

Рассмотрим в качестве примера магазин данного сайта на рисунке 1.1.

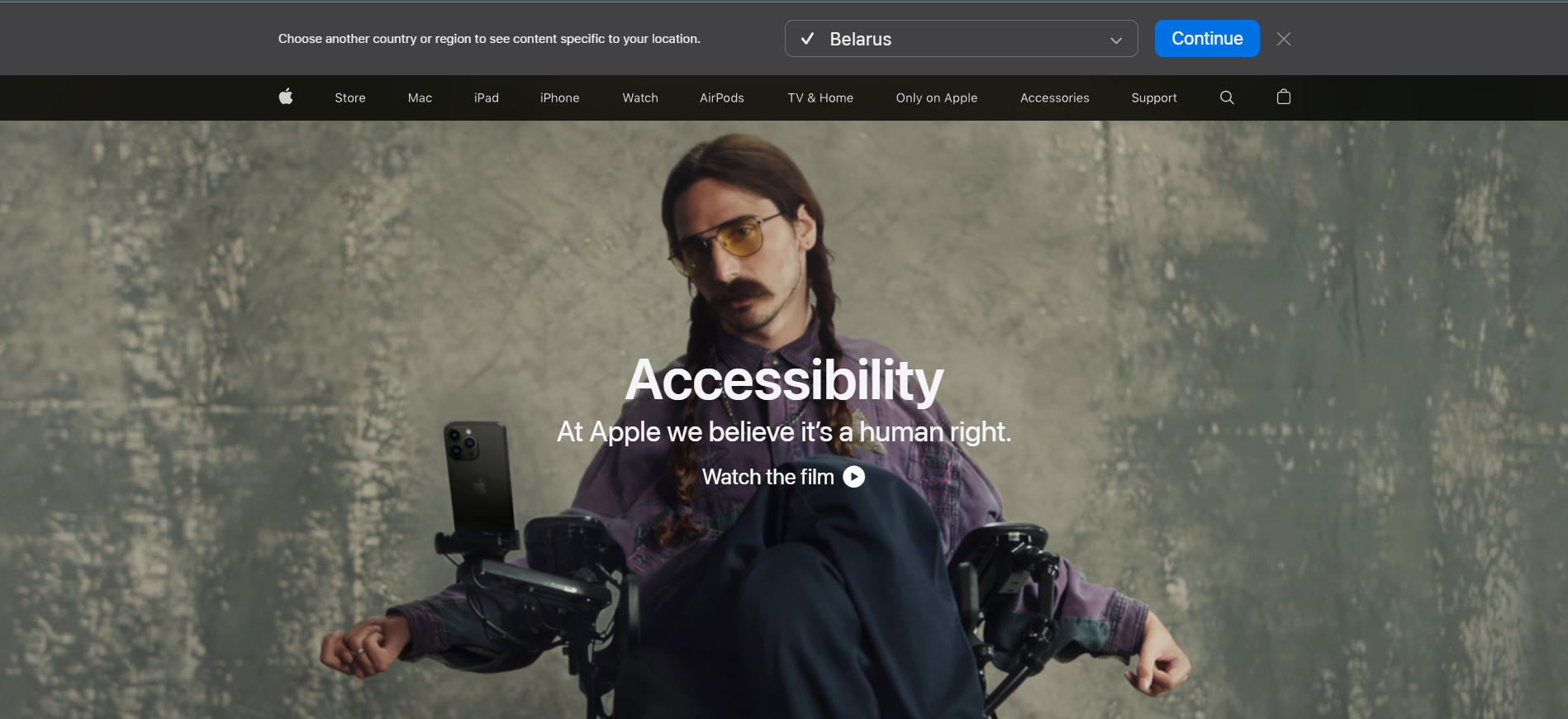


Рисунок 1.1 – Пример магазина «Apple Store»

Тут видно, что в данном магазине присутствует много разных устройств. Кликнув на один из них, мы попадаем на страницу описания телефонов, где можно ознакомиться с подробной информацией о конкретном мобильном устройстве, а также заказать телефон в любое время, что продемонстрировано на рисунке 1.2 и 1.3.



Рисунок 1.2 – Подробное описание выбранного телефона

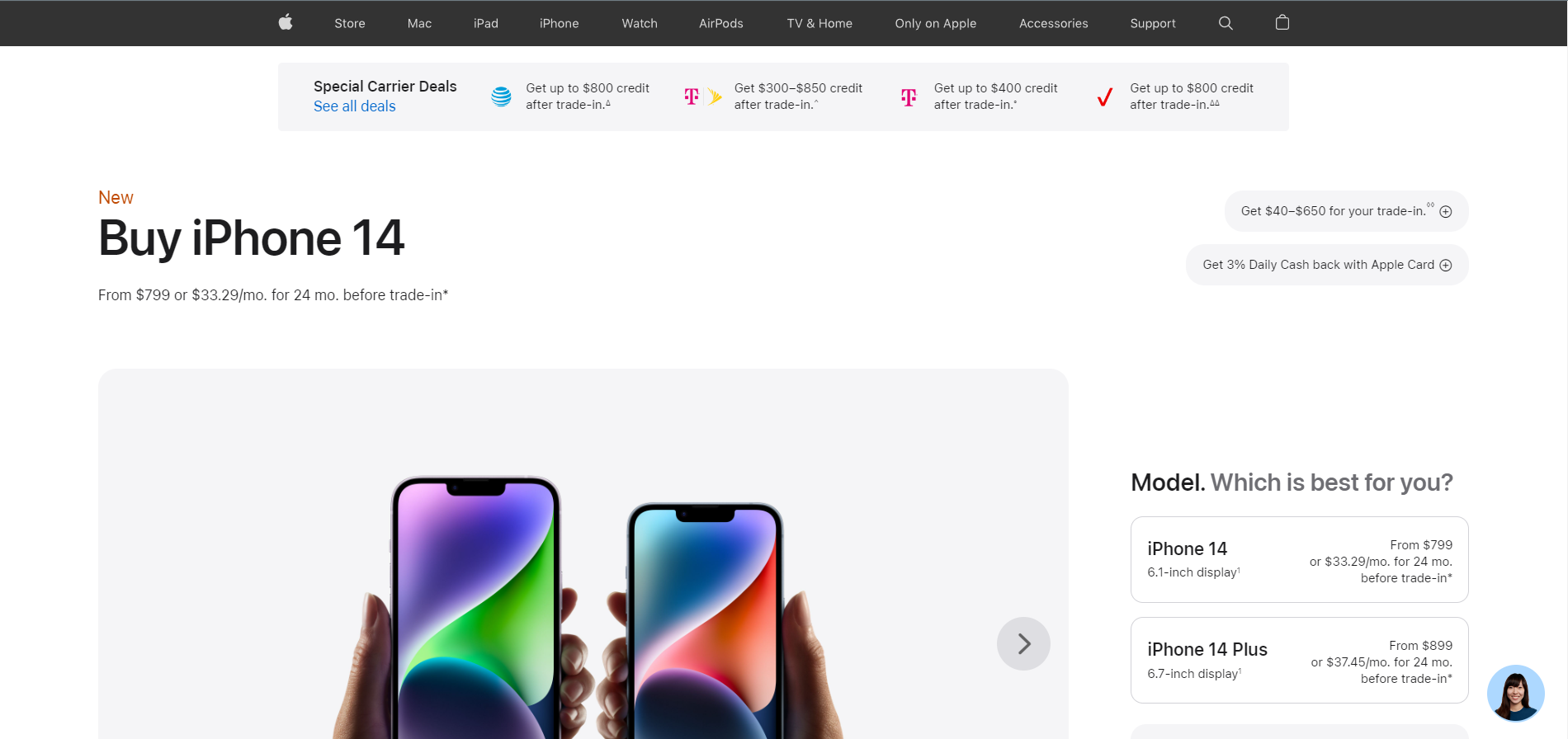


Рисунок 1.3 – Приобретение телефона

# 1.2. Описание используемых технологий

СУБД существует огромное множество: Oracle, MS SQL Server, MySql и так далее. Для организации работы с базой данных в данной работе было решено использовать одну из наиболее популярных СУБД «Microsoft SQL Server», так как она предоставляет необходимые возможности оперирования объектами базы данных.

Также в процессе выбора технологий были рассмотрены технологии «Динамическое маскирование данных и шифрования на уровне столбцов». Динамическое маскирование данных (DDM) ограничивает возможность раскрытия конфиденциальных данных за счет маскирования этих данных для непривилегированных пользователей. Оно позволяет значительно упростить проектирование и написание кода для системы безопасности в приложении.

Динамическое маскирование данных помогает предотвратить несанкционированный доступ к конфиденциальным данным, позволяя клиентам задать объем раскрываемых конфиденциальных данных с минимальным влиянием на уровень приложения. DDM можно настроить для отдельных полей базы данных, чтобы скрыть конфиденциальные данные в результирующих наборах запросов. При использовании DDM данные в базе данных не изменяются. DDM легко использовать с существующими приложениями, так как правила маскирования применяются к результатам запроса. Многие приложения могут маскировать конфиденциальные данные без изменения существующих запросов.

* Центральная политика маскирования данных применяется непосредственно к конфиденциальным полям в базе данных.
* Вы можете назначать привилегированных пользователей или роли, которые имеют доступ к конфиденциальным данным.
* DDM включает функции полного и частичного маскирования, а также возможность использования случайной маски для числовых данных.
* Назначение и использование масок осуществляется простыми командами Transact-SQL.

Назначение динамического маскирования данных — ограничение раскрытия конфиденциальных данных, при котором пользователи, у которых нет доступа к данным, не смогут их просматривать. Динамическое маскирование данных не сможет помешать пользователям подключиться к базе данных напрямую и выполнить запросы для получения фрагментов конфиденциальных данных. Динамическое маскирование данных дополняет другие функции безопасности SQL Server (аудит, шифрование, безопасность на уровне строк и т. д.). Настоятельно рекомендуется использовать эту возможность с этими функциями для лучшей защиты конфиденциальных данных в базе данных.

Шифрование представляет собой способ скрытия данных с помощью ключа или пароля. Это делает данные бесполезными без соответствующего ключа или пароля для дешифрования. Шифрование не решает проблемы управления доступом. Однако оно повышает защиту за счет ограничения потери данных даже при обходе системы управления доступом. Например, если компьютер, на котором установлена база данных, был настроен неправильно и злоумышленник смог получить конфиденциальные данные, то украденная информация будет бесполезна, если она была предварительно зашифрована.

# 2.Разработка модели базы данных

Как отмечалось в предыдущих разделах, одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие – составление такой модели данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для пользователя.

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

Для реализации базы данных для сети магазинов было разработано 6 таблиц.

Таблица worker хранит работников магазина.

Таблица comment содержит комментарии клиента о заказе.

Таблица good хранит определённые товары магазина.

Таблица client хранит клиентов магазина.

Таблица timetable\_worker содержит в себе свободное время доставщиков товара.

Таблица order\_list хранит в себе весь список заказов клиентов и какой работник в свободное время может доставить этот заказ.

На рисунке 2.1 отображена модель базы данных, полученная в ходе анализа предметной области и необходимого функционала. Между сущностями, представленными таблицами, также настроены связи, отражающие их взаимосвязь.

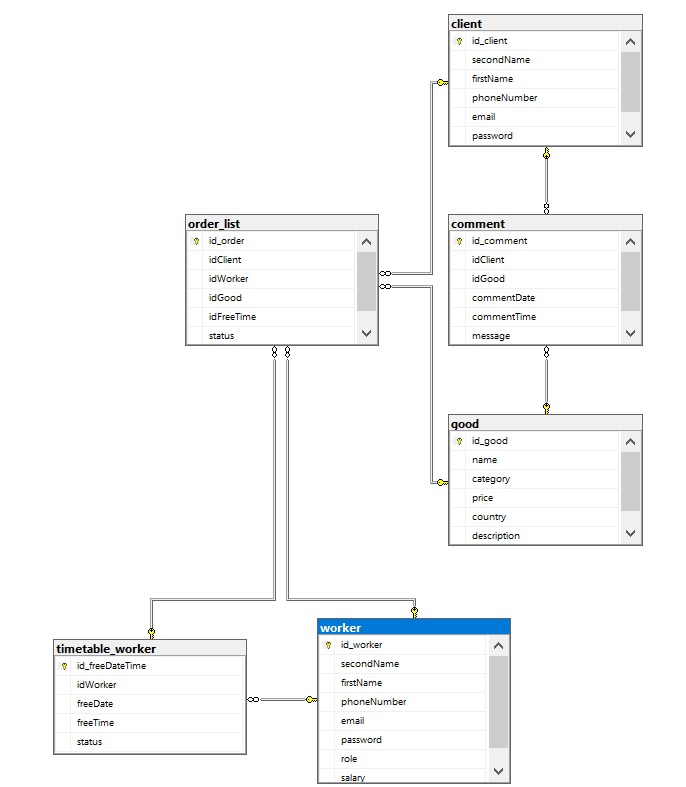


Рисунок 2.1 – Структурная схема базы данных.

# 3. Разработка необходимых объектов

При разработке курсового проекта понадобились следующие объекты:

* Таблицы;
* Хранимые процедуры;
* Индексы.
* Триггеры

# 3.1.Таблицы

Таблицы являются неотъемлемой частью любой реляционной базы данных. Краткая характеристика каждой из таблиц была предоставлена в разделе 2, а код их создания можно увидеть в Приложении А. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица worker состоит из 8 столбцов, рисунок 3.1:

* id\_worker – уникальный идентификатор работника;
* secondName – фамилия работника;
* firstName – Имя работника;
* phoneNumber – номер мобильного телефона работника;
* email – электронная почта работника;
* password – пароль работника ;
* role – роль работника (0 – worker, 1 – admin);
* salary – заработная плата работника.

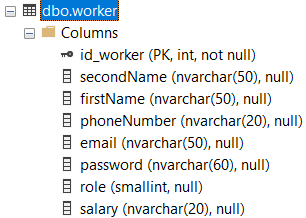


Рисунок 3.1 – Описание таблицы worker

Таблица comment состоит из 6 столбцов, рисунок 3.2:

* id\_comment – уникальный идентификатор комментария;
* idClient – уникальный идентификатор клиента оставившего комментарий;
* idGood – уникальный идентификатор товара про который оставили комментарий;
* commentDate – дата оставления комментария;
* commentTime – время оставления комментария;
* message – сообщение оставленного комментария;

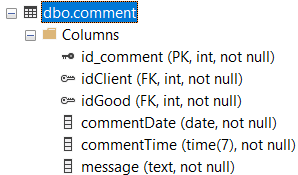


Рисунок 3.2 – Описание таблицы comment

Таблица good состоит из 6 столбцов, рисунок 3.3:

* id\_good – уникальный идентификатор товара;
* name – наименование товара;
* category – категория товара;
* price – цена товара;
* country – страна производитель;
* description – описание товара;

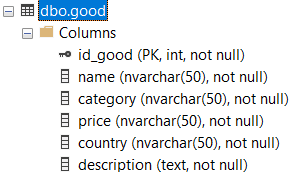


Рисунок 3.3 – Описание таблицы good

Таблица client состоит из 6 столбцов, рисунок 3.4:

* id\_client – уникальный идентификатор клиента;
* secondName – фамилия клиента;
* firstName – имя клиента;
* phoneNumber – номер мобильного телефона клиента;
* email – электронная почта клиента;
* password – пароль клиента;

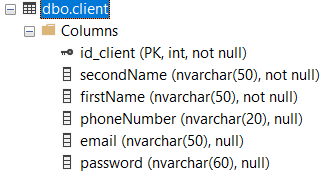


Рисунок 3.4 – Описание таблицы client

Таблица timetable\_worker состоит из 5 столбцов, рисунок 3.5:

* id\_freeDateTime – уникальный идентификатор доставки;
* idWorker – уникальный идентификатор работника;
* freeDate – свободна дата;
* freeTime – свободное время;
* status – статус работника (0 – free);

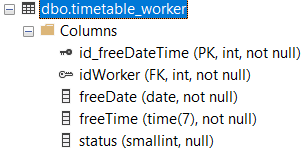


Рисунок 3.5 – Описание таблицы timetable\_worker

Таблица order\_list состоит из 6 столбцов, рисунок 3.6:

* id\_order – уникальный идентификатор заказа;
* idClient – уникальный идентификатор клиента;
* idWorker – уникальный идентификатор работника;
* idGood – уникальный идентификатор товара;
* idFreeTime – уникальный идентификатор доставки;
* status – статус заказа (0 – completed, 1 – not done);

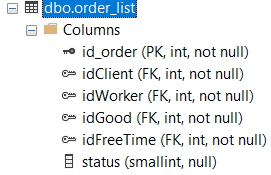


Рисунок 3.6 – Описание таблицы order\_list

# 3.2.Процедуры

Использование хранимых процедур позволяет ограничить либо вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным. Логика, отражающая возможность пользователя взаимодействовать данными, находящимися в базе данных, реализована с помощью хранимых процедур, написанных на языке T-SQL. Операции, которые реализуют процедуры, можно разделить на две большие группы: операции, попадающие под определение CRUD и все остальные, связанные со спецификой предметной области базы данных.

Всего было разработано 39 процедур для работы с данными. Краткое описание процедур описано далее.

Процедуры для получения данных из базы:

* get\_goods\_by\_id – получение товара по уникальному идентификатору;
* get\_workers – получение списка всех работников;
* get\_workers\_by\_name – получение работника по его имени;
* get\_workers\_by\_email – получение работника по его электронной почте;
* get\_id\_worker\_by\_email – получение уникального идентификатора работника по его электронной почте;
* get\_role\_worker – получение роли работника;
* get\_clients – получение всех клиентов магазина;
* get\_client\_by\_name – получение клиента по его имени;
* get\_client\_by\_email – получение уникального идентификатора клиента по его электронной почте;
* get\_goods – получение списка товаров;
* get\_order\_list – получение списка заказов;
* get\_goods\_by\_name – получение товара по его названию;
* get\_info\_client\_by\_id – получение информации о клиенте по его уникальному идентификатору;
* get\_id\_client\_by\_email – получение уникального идентификатора клиента по его электронной почте;
* get\_email\_client\_by\_id – получение электронной почты клиента по его уникальному идентификатору;
* get\_client\_orders\_by\_id – получение списка заказов по уникальному идентификатору;
* get\_good\_comments\_by\_id – получение списка комментариев к товару по уникальному идентификатору;
* get\_count\_comments\_by\_id – получение количества комментариев по уникальному идентификатору;
* get\_datetime\_order\_by\_id – получение списка свободных доставщиков для доставки заказа по уникальному идентификатору;
* get\_freedatetime\_worker – получение свободной даты и время доставщика;
* get\_comment\_by\_id\_good – получение списка комментариев по уникальному идентификатору товара;

Пример создания процедуры можно видеть на листинге 3.1

go

Create procedure GET\_WORKERS\_BY\_NAME

@search\_text nvarchar(40)

as begin

SELECT \* from worker where (upper(secondName) like upper(@search\_text) + '%') or (upper(firstName) like upper(@search\_text) + '%');

end;

Листинг 3.1 – Создание хранимой процедуры GET\_WORKERS\_BY\_NAME

Процедуры для создания, изменения, удаления:

* RegClient – создание записи в таблице client;
* Add\_good – создание записи в таблице good;
* Delete\_good – удаление записи из таблицы good;
* Add\_order – создание записи в таблицу order\_list;
* Edit\_goods – изменение записи в таблице good;
* Add\_comment – создание записи в таблице comment;

Пример создания процедуры можно видеть на листинге 3.2

go

Create procedure ADD\_GOOD

@Name nvarchar(35),

@Category nvarchar(35),

@Price nvarchar(20),

@Country nvarchar(50),

@Description TEXT

as begin

insert into good(name, category, price, country, description)

values (@Name, @Category, @Price, @Country, @Description);

end;

Листинг 3.2 – Создание хранимой процедуры ADD\_GOOD

Код создания всех процедур продемонстрирован в приложении В.

Также была создана процедура заполнения БД, а точнее таблицы client записями, количеством равным 100 000 записей.

# 4. Описание процедур импорта и экспорта

Одной из задач по разработке инфраструктуры базы данных являлась реализация процедур, которые будут осуществлять экспорт и импорт данных в формате XML.

Процедуры для импорта и экспорта:

* GoodToXml – экспорт Xml;
* XmlToGood – импорт Xml;

Пример создания процедур можно видеть на листингах 4.1-4.2

go

create procedure GoodToXml

as begin

select name, category, price, country, description

from good for xml path('good'), root('goods');

exec master.dbo.sp\_configure 'show advanced options', 1

reconfigure with override

exec master.dbo.sp\_configure 'xp\_cmdshell', 1

reconfigure with override;

declare @cmd nvarchar(255);

select @cmd = '

bcp "use MobilePhone; select name, category, price, country, description from good for xml path(''good''), root(''goods'')" ' +

'queryout "D:\Education\3 курс 1 сем\Kursach\Export.xml" -S .\SQLEXPRESS -T -w -r -t';

exec xp\_cmdshell @cmd;

end;

Листинг 4.1 – Создание хранимой процедуры GoodToXml

go

create procedure XmlToGood

as begin

DECLARE @xml XML;

SELECT @xml = CONVERT(xml, BulkColumn, 2) FROM OPENROWSET(BULK 'D:\Education\3 курс 1 сем\Kursach\Export.xml', SINGLE\_BLOB) AS x

INSERT INTO good(name, category, price, country, description)

SELECT

t.x.query('name').value('.', 'nvarchar(30)'),

t.x.query('category').value('.', 'nvarchar(30)'),

t.x.query('price').value('.', 'nvarchar(30)') ,

t.x.query('country').value('.', 'nvarchar(30)') ,

t.x.query('description').value('.', 'nvarchar(2000)')

FROM @xml.nodes('//goods/good') t(x)

End

Листинг 4.2 – Создание хранимой процедуры XmlToGood

# 5. Тестирование производительности

Оптимизация запросов — процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса. Один и тот же результат может быть получен СУБД различными способами (планами выполнения запросов), которые могут существенно отличаться как по затратам ресурсов, так и по времени выполнения.

В MS SQL Server оптимизация запросом в основном заключается в построение индексов над таблицами, и изменением плана запроса. Индекс – это объект базы данных, предназначенный для ускорения запросов к данным в таблице базы данных. Индекс состоит из набора страниц, узлов индекса, которые организованы в виде древовидной структуры — сбалансированного дерева. Эта структура является иерархической по своей природе и начинается с корневого узла на вершине иерархии и конечных узлов, листьев. Есть два основных типа индексов: кластеризованные и некластеризованные индексы.

При создании кластеризованного индекса данные индексируемой таблицы располагаются в физическом порядке, соответствующем индексу, и становятся частью кластеризованного индекса.

Некластеризованный индекс — это отдельный объект, имеющий указатели на строки таблицы. Максимальное количество некластерированных индексов для одной таблицы не должно превышать 1000.

Рассмотрим следующие рекомендации при планировании стратегии индексирования:

* для таблиц, которые часто обновляются используйте как можно меньше индексов;
* если таблица содержит большое количество данных, но их изменения незначительны, тогда используйте столько индексов, сколько необходимо для улучшения производительности ваших запросов;
* уникальность значений в столбце влияет на производительность индекса, в общем случае, чем больше у вас дубликатов в столбце, тем хуже работает индекс;
* для кластеризованных индексов старайтесь использовать настолько короткие поля насколько это возможно. Наилучшим образом будет применение кластеризованного индекса на столбцах с уникальными значениями и не позволяющими использовать NULL. Вот почему первичный ключ часто используется как кластеризованный индекс.

Важной характеристикой кластеризованного индекса является то, что все значения отсортированы в определенном порядке либо возрастания, либо убывания. Таким образом, таблица или представление может иметь только один кластеризованный индекс.

Для тестирования базы данных проекта была выбрана таблица client. В базе данных данного курсового проекта в каждой таблице находится поле с уникальным значением поля — id типа int, соответственно каждая таблица содержит кластеризованный индекс.

База данных содержит множество процедур с выборкой содержащий оператор WHERE сравнивающий строки. Даже при незначительном заполнении базы данных (около 100000 записей в таблице) время выборки начинает занимать существенное время.

Ниже на рисунке 5.1 представлены общая стоимость и план запроса при выборке из таблицы client.

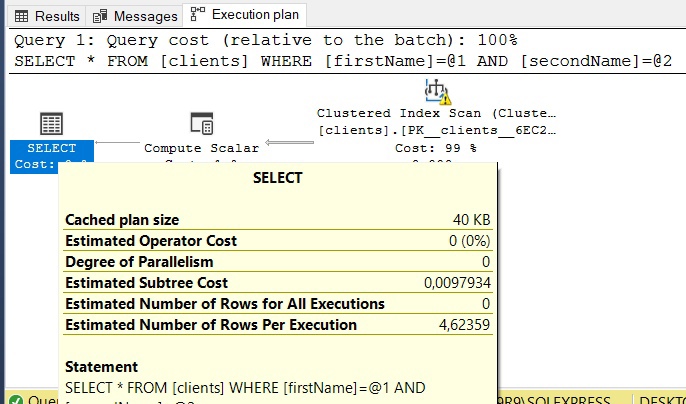


Рисунок 5.1 — Общая стоимость и план запроса

План выполнения запроса — последовательность операций, необходимых для получения результата SQL-запроса в реляционной СУБД.

План в целом разделяется на две стадии:

* Выборка результатов;
* Сортировка и группировка, выполнение агрегаций.

Сортировка и группировка — это опциональная стадия, которая выполняется, если не найдено путей доступа для получения результата в запрошенном порядке.

Выборка результатов выполняется следующими способами:

* Вложенные циклы;
* Слияние.

Вложенные циклы — это вложенные итеративные процессы поиска данных в каждой из соединяемых таблиц.

Если объединяемые таблицы имеют индексы по сравниваемым полям, то объединение может быть выполнено с помощью слияния. Оба индекса сканируются и в них ищутся одинаковые значения.

Для таблицы client был создан некластеризованный индекс покрытия, код создания представлен на рисунке 5.2.

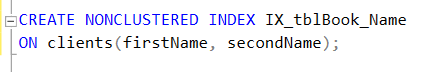


Рисунок 5.2 — Код создания некластеризованного индекса покрытия

Индекс покрытия — это индекс non-clustered. Как кластеризованные, так и некластеризованные индексы используют структуру данных B-дерева для улучшения поиска данных, разница в том, что кластеризованный индекс физически хранит строки, а некластеризованный индекс хранит указатели на строки.

Общая стоимость и план запроса с применением некластеризованного индекса представлены на рисунке 5.3.

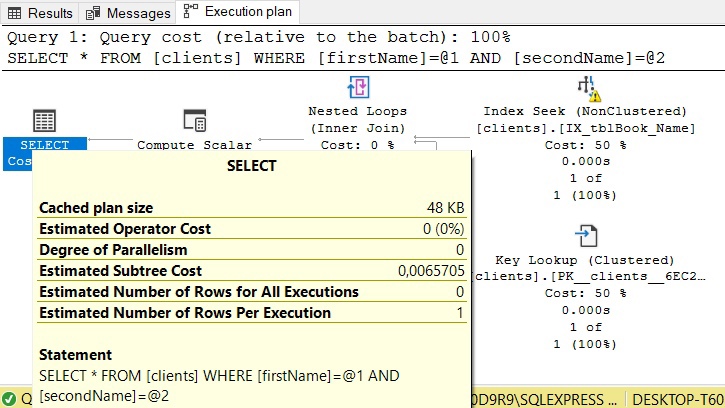


Рисунок 5.3 — Общая стоимость и план запроса с некластеризованным индексом покрытия

Как видно из рисунка 5.3, стоимость запроса уменьшилась примерно в 1.5 раза (пункт «Предполагаемая стоимость поддерева»). В следствии этого можно сделать вывод, что была ускорена выборка данных из таблицы.

# 6. Описание технологии

Тут рассматриваются средства SQL Server для обеспечения безопасности, которые были применены к проекту, связанные с настройкой и обеспечением безопасности в СУБД.

Процедуры для маскирования и шифрования:

* MaskAdminClient – процедура показывающая как будет админ видеть замаскированную таблицу client;
* MaskAdminWorker – процедура показывающая как будет админ видеть замаскированную таблицу worker;
* MaskClient – процедура показывающая как будет обычный пользователь видеть замаскированную таблицу client;
* MaskWorker – процедура показывающая как будет обычный пользователь видеть замаскированную таблицу worker;
* Encrypt – шифрование данных в определённой таблице;
* Decoding – расшифровывание данных в определённой таблице;

Примеры создания процедур можно видеть на листингах 6.1-6.2

go

create procedure Encrypt

AS BEGIN

OPEN SYMMETRIC KEY SymKey\_test

DECRYPTION BY CERTIFICATE Certificate\_test;

UPDATE dbo.good

SET price\_encrypt = EncryptByKey (Key\_GUID('SymKey\_test'), price)

FROM dbo.good;

CLOSE SYMMETRIC KEY SymKey\_test;

END;

Листинг 6.1 – Создание хранимой процедуры Encrypt

go

create procedure Decoding

AS BEGIN

OPEN SYMMETRIC KEY SymKey\_test

DECRYPTION BY CERTIFICATE Certificate\_test;

SELECT name, category, country, price\_encrypt AS 'Encrypted data',

CONVERT(nvarchar, DecryptByKey(price\_encrypt)) AS 'Decrypted Bank account number'

FROM dbo.good;

END;

Листинг 6.2 – Создание хранимой процедуры Decoding

# 6.1.Динамическая маскировка данных

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к базе данных и пользуется услугами информационной системы для получения информации. Пользователи базы данных получают права доступа для чтения, вставки, обновления и удаления конкретных объектов.

На рисунке 6.3 создано два пользователя – администратор и клиент.

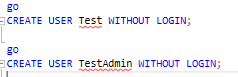


Рисунок 6.3 – Создание пользователей

На рисунке 6.4 мы устанавливаем роли и привилегии нашим пользователям.

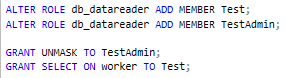


Рисунок 6.4 – Установка роли и привилегий

Администратору разрешено просматривать данные в замаскированных таблиц , но обычному пользователю это сделать невозможно, так как данные будут замаскированы для него.

# 6.2.Шифрование на уровне столбцов с использованием симметричных ключей

Безопасность данных является важной задачей для любой организации, особенно если вы храните личные данные клиентов, такие как контактный номер клиента, адрес электронной почты, номер социального страхования, номера банков и кредитных карт.

На первом шаге мы определяем главный ключ базы данных и предоставляем пароль для его защиты. Это симметричный ключ для защиты закрытых ключей и асимметричных ключей. На приведенной выше диаграмме мы видим, что главный ключ службы защищает этот главный ключ базы данных. SQL Server создает этот главный ключ службы в процессе установки, всё представлено на рисунке 6.5.

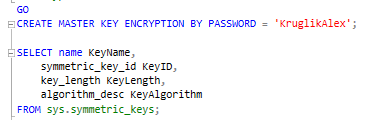


Рисунок 6.5 – Создание главного ключа базы данных

На втором шаге мы создаем самозаверяющий сертификат с помощью оператора CREATE CERTIFICATE. Возможно, вы видели, что организация получает сертификат от центра сертификации и включает его в свою инфраструктуру. В SQL Server мы можем использовать самозаверяющий сертификат без использования сертификата центра сертификации, всё представлено на рисунке 6.6.

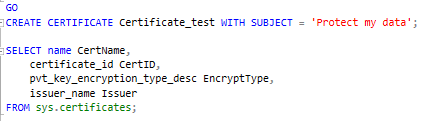


Рисунок 6.6 – Создание сертификата

На третьем шаге мы определим симметричный ключ, который вы также можете увидеть в иерархии шифрования. Симметричный ключ также использует один ключ для шифрования и дешифрования. На представленном выше изображении мы видим симметричный ключ поверх данных. Для шифрования данных рекомендуется использовать симметричный ключ, так как с ним мы получаем отличную производительность. Для шифрования столбцов мы используем многоуровневый подход, который дает преимущества производительности симметричного ключа и безопасности асимметричного ключа, это представлено на рисунке 6.7.

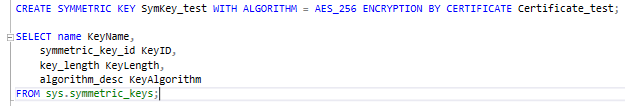


Рисунок 6.7 – Создание и настройка симметричного ключа

Ну и в последнем шаге мы будем шифровать данные.

Для этого в окне запроса открываем симметричный ключ и расшифруем его с помощью сертификата. Нам нужно использовать тот же симметричный ключ и имя сертификата, которые мы создали ранее.

Далее в том же сеансе мы используем оператор UPDATE. Он использует функцию EncryptByKey и использует симметричную функцию для шифрования нашего столбцы и потом обновляет данные во вновь созданном столбце.

В самом последнем действии мы закрываем симметричный ключ с помощью оператора CLOSE SYMMETRIC KEY. Если мы не закроем ключ, он останется открытым до завершения сеанса.

Все действия представлены на рисунке 6.8.

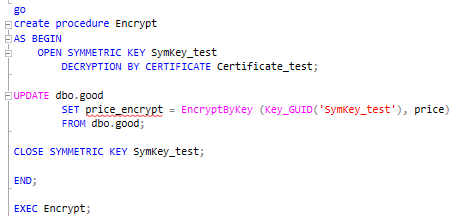


Рисунок 6.8 – Шифрование данных

# 6.3.Расшифровка данных на уровне столбов

Для того чтобы расшифровать зашифрованные данные на уровне столбцов на надо в окне запроса открыть симметричный ключ и расшифровать с помощью сертификата. Нам нужно использовать тот же симметричный ключ и имя сертификата, который создали ранее.

Далее нам надо использовать оператор SELECT и расшифровать зашифрованные данные с помощью функции DecryptByKey().

Все действия представлены на рисунке 6.9.

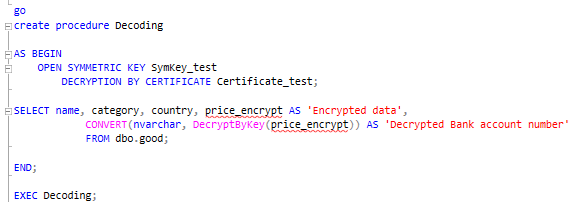


Рисунок 6.9 – Расшифровка данных

# 6.4.Оценка уязвимостей через SSMS

В SQL Server Management Studio есть функция оценки уязвимостей для базы данных, рисунок 6.10.

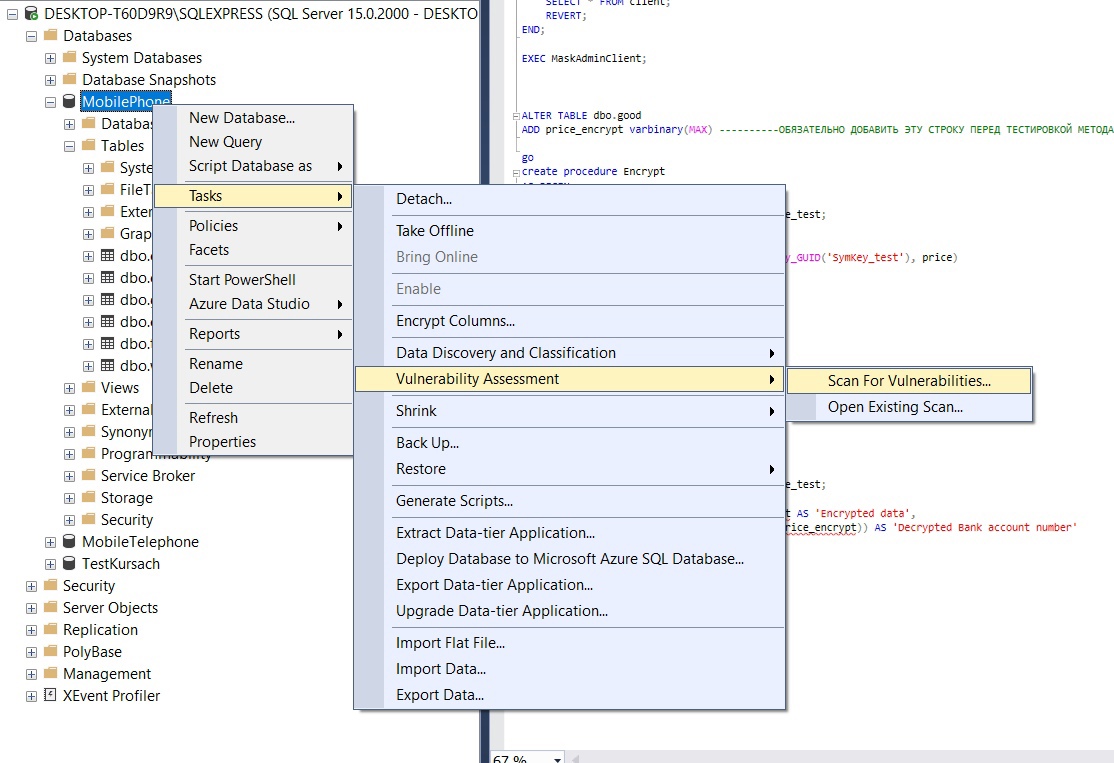


Рисунок 6.10 – Открытие оценки уязвимостей в SSMS

Сканнер оценит базу данных на предмет популярных ошибок в конфигурации безопасности и даст соответствующие рекомендации, рисунок 6.11.

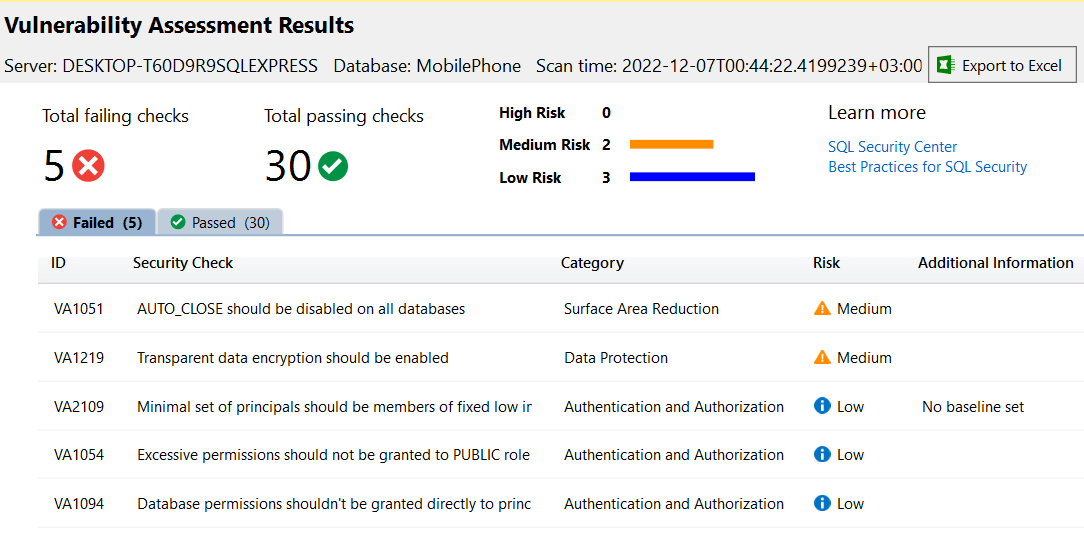


Рисунок 6.11 – Ошибки в конфигурации безопасности

Обязательно стоит проходить этим сканнером по базам данных. Он может выявить скрытые проблемы, которых не видно на первый взгляд.

# Заключение

В ходе выполнения данной работы подробно были закреплены знания для проектирования баз данных, с учётом большого количества хранимых параметров, учётом множества связей и использование процедур, курсоров и функций для осуществления взаимодействий с таблицами. Было проведено тестирование производительности, в котором анализ времени выполнения показал, что база данных работает быстро, даже не в реалистичных условиях.

Разработанная база данных отвечает всем требованиям предметной области, ее таблицы созданной отвечают требованиям нормализации, что позволяет обеспечить целостность и непротиворечивость информации. База данных была разработана с помощью «Microsoft SQL Server». Помимо этого, была изучена и применена при разработке приложения технология настройки системы безопасности сервера СУБД.

Благодаря проведенной работе, была создана база данных, которая в дальнейшем может быть использована на проект и также её можно легко дорабатывать и расширять её возможности.

При разработке базы данных:

* создано 6 таблиц;
* создано 40 хранимые процедуры;
* создано 2 пользователя базы данных;
* включено шифрование на уровне столбцов с использованием симметричных ключей;
* включено расшифровка данных на уровне столбов с использованием симметричных ключей;
* включено динамическое маскирование данных;

В соответствии с полученным результатом работы можно сделать вывод, что разработанная БД работает верно, а требования технического задания выполнены в полном объёме.

# Список используемых источников

1. Документация СУБД «Microsoft SQL Server» [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15. Дата доступа: 10.11.2022.
2. Restore Statements (Transact-SQL) [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/enёus/sql/t-sql/statements/restore-statements-transact-sql?view=sql-server-ver15. Дата доступа: 24.11.2022.
3. Restore a Database Backup Using SSMS [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-a-database-backup-using-ssms?view=sql-server-ver15>. Дата доступа: 15.11.2022.
4. Шифрование в базах данных SQL Server [Электронный ресурс] – Режим доступа: https://www.osp.ru/winitpro/2013/05/13035359. Дата доступа: 20.11.2022.
5. Основные средства обеспечения безопасности в SQL Server [Электронный ресурс] – Режим доступа: https://winitpro.ru/index.php/2020/02/07/bezopasnost-sql-server. Дата доступа: 24.11.2022.
6. Роли уровня базы данных [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver15. Дата доступа: 19.11.2022.
7. Роли уровня сервера [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-ver15. Дата доступа: 17.11.2022.
8. Итеративный учебник по SQL – [Электронный ресурс] – Режим доступа: http://sql-tutorial.ru/ru/content.html. Дата доступа: 24.11.2022.

# Приложения A. Листинг создания БД и таблиц

create database MobilePhone;

Use MobilePhone;

Create Table client

(

id\_client int identity(1,1) primary key,

secondName nvarchar(50) not null,

firstName nvarchar(50) not null,

phoneNumber nvarchar(20) MASKED WITH (FUNCTION = 'partial(1,"XXXXXXX",0)') NULL,

email nvarchar(50) MASKED WITH (FUNCTION = 'email()') NULL,

password nvarchar(60) MASKED WITH (FUNCTION = 'default()') NULL

);

Create Table good

(

id\_good int identity(1,1) primary key,

name nvarchar(50) not null,

category nvarchar(50) not null,

price nvarchar(50) not null,

country nvarchar(50) not null,

description TEXT not null,

);

Create Table worker(

id\_worker int identity(1,1) primary key,

secondName nvarchar(50),

firstName nvarchar(50),

phoneNumber nvarchar(20) MASKED WITH (FUNCTION = 'partial(5,"XXXXXXX",0)') NULL,

email nvarchar(50) MASKED WITH (FUNCTION = 'email()') NULL,

password nvarchar(60) MASKED WITH (FUNCTION = 'default()') NULL,

role smallint, --0 - worker, 1 -admin

salary nvarchar(20)

);

Create Table order\_list

(

id\_order int identity(1,1) primary key,

idClient int foreign key (idClient) references client(id\_client) not null,

idWorker int foreign key (idWorker) references worker(id\_worker) not null,

idGood int foreign key (idGood) references good(id\_good) not null,

idFreeTime int foreign key (idFreeTime) references timetable\_worker(id\_freeDateTime) not null,

status smallint -- 0 - completed, 1 -not done

);

create table timetable\_worker(

id\_freeDateTime int identity(1,1) primary key not null,

idWorker int foreign key (idWorker) references worker(id\_worker) not null,

freeDate date not null,

freeTime time not null,

status smallint -- 0 -free

);

Create Table comment

(

id\_comment int identity(1,1) primary key,

idClient int foreign key (idClient) references client(id\_client) not null,

idGood int foreign key (idGood) references good(id\_good) not null,

commentDate date not null,

commentTime time not null,

message TEXT not null

)

# Приложения B. Листинг процедур

-- поиск товара

go

Create procedure GET\_GOODS\_BY\_NAME

@search\_text nvarchar(40)

as begin

SELECT \* from good where (upper(name) like upper(@search\_text) + '%');

end;

--рестрация клиента

go

Create procedure RegClient

@firstName nvarchar(35),

@secondName nvarchar(35),

@phoneNumber nvarchar(20),

@email nvarchar(50),

@password nvarchar(60)

AS

Begin

insert into client(secondName,firstName, phoneNumber, email, password)

values (@secondName, @firstName, @phoneNumber, @email, @password);

End;

--авторизация клиента

go

Create procedure Login

@log nvarchar(13),

@pass nvarchar(30)

AS

Begin

Declare @passwordClient nvarchar(30);

Declare @passwordWorker nvarchar(30);

set @passwordClient = (select password from client where email= @log);

set @passwordWorker = (select password from worker where email= @log);

if @passwordClient = @pass or @passwordWorker = @pass

select 1 as login;

else

select 0 as login;

End;

--добавление товара в корзину

go

Create procedure ADD\_GOOD

@Name nvarchar(35),

@Category nvarchar(35),

@Price nvarchar(20),

@Country nvarchar(50),

@Description TEXT

as begin

insert into good(name, category, price, country, description)

values (@Name, @Category, @Price, @Country, @Description);

end;

--удаление товара из корзины

go

Create Procedure DELETE\_GOOD

@idGood int

as begin

delete from good where id\_good = @idGood;

end;

--оформление заказа

go

CREATE PROCEDURE AddOrder

@idClient int,

@orderDate date,

@orderTime time,

@name nvarchar(30),

@firstNameWorker nvarchar(35),

@secondNameWorker nvarchar(35)

AS begin

declare @idWorker int, @idGood int, @idFreeTime int;

set @idGood = (select id\_good from good where name = @name);

set @idWorker = (select id\_worker from worker where firstName = @firstNameWorker and

secondName = @secondNameWorker);

set @idFreeTime = (select id\_freeDateTime from timetable\_worker where

freeDate = @orderDate and

freeTime = @orderTime);

INSERT INTO order\_list(idClient, idWorker, idGood, idFreeTime, status)

VALUES(@idClient, @idWorker, @idGood, @idFreeTime, 0);

End

--редактирование товаров

go

Create Procedure EDIT\_GOODS

@Price nvarchar(30),

@idGood int

as begin

update good set price = @Price where id\_good = @idGood;

end;

--просмотр товаров

go

Create procedure GET\_GOODS

as begin

SELECT \* from good;

end;

--просмотр заказов

go

Create procedure GET\_ORDER\_LIST

as begin

SELECT \* from order\_list;

end;

--заполнение 100000 строк

go

create procedure procedure\_ADD\_100\_000\_ROW

AS

BEGIN

DECLARE @secondName NVARCHAR(50);

DECLARE @firstName NVARCHAR(50);

DECLARE @phoneNumber NVARCHAR(50);

DECLARE @email NVARCHAR(50);

DECLARE @password NVARCHAR(50);

DECLARE @countJ INT;

SET @countJ=1;

while @countJ<100000

BEGIN

SET @secondName='secondName\_'+cast(@countJ as varchar);

SET @firstName='firstName\_'+cast(@countJ as varchar);

SET @phoneNumber='phoneNumber\_'+cast(@countJ as varchar);

SET @email='email\_'+cast(@countJ as varchar);

SET @password='password\_'+cast(@countJ as varchar);

exec RegClient @secondName,@firstName,@phoneNumber, @email,@password

SET @countJ=@countJ+1;

END

END