

СОДЕРЖАНИЕ

Реферат	5
Введение	6
1 Аналитический раздел	8
1.1 Формализация данных	8
1.2 Описание ролевой модели	10
1.3 Описание понятия СУБД	11
1.3.1 Классификация СУБД по модели данных	12
1.4 Методы построения линии тренда	15
1.4.1 Виды линий трендов	15
1.4.2 Анализ исходных данных для построения линии тренда . .	17
Вывод к аналитическому разделу	17
2 Конструкторский раздел	18
2.1 Разработка базы данных	18
2.2 Разработка объектов базы данных	20
2.3 Разработка системы безопасности БД	22
Вывод к конструкторскому разделу	23
3 Технологический раздел	24
3.1 Средства реализации	24
3.2 Интерфейс ПО	25
Вывод к технологическому разделу	27
4 Экспериментальный раздел	28
4.1 Постановка эксперимента	28
4.1.1 Цель эксперимента	28
4.1.2 Описание эксперимента	28
4.2 Результаты эксперимента	29
Вывод к экспериментальному разделу	30

Заключение	32
Список использованных источников	33
Приложение Б	43

Реферат

Объем РПЗ составляет 36 страниц, содержит 6 иллюстраций, 5 таблиц, 2 приложения и 19 использованных источников.

Ключевые слова: прогнозирование цен, модель для прогнозирования цен, товары, магазины, история цен, приложение для работы с БД, СУБД, PostgreSQL, C#, WPF.

В РПЗ были сформулированы цель и задачи работы, формализованы данные, описана ролевая модель, понятие СУБД и методы построения линии тренда. Также были разработаны БД, объекты БД, система безопасности БД, реализовано вышеперечисленное в ПО, решающее данную задачу, и был проведён эксперимент по оценке точности модели для предсказания цен на товары в магазинах.

Введение

Методы прогнозирования многообразны, как и объекты, прогнозированием которых занимается человек. Ему всегда было интересно узнать будущее – свое, своих близких, государства, экономики, предугадать погоду, определить как изменятся в ближайшее время курсы валют и т. п.

Но прогнозировать ситуацию важно не только из-за того, что это интересно, но и потому, что от предвидения будущего зависят действия человека. Если прогнозируется скачок цен на какой-то товар, то можно заранее закупить его. Можно сказать, что жизнь современного человека невозможно себе представить без прогнозирования.

Прогноз – это результат индуктивного вывода, когда по характеру ограниченного множества значений показателей или взаимосвязи факторов делается вывод о том, что и остальные, еще не наблюдаемые значения этих показателей или взаимосвязи будут обладать аналогичными свойствами [**hse_pred**].

В мировой экономической науке накоплен и апробирован значительный арсенал методов прогнозирования, который дает возможность решать комплекс задач по обоснованию решений в различных областях [**bel_prog**].

Задача прогнозирования цен на потребительские товары актуальна для различных типов пользователей: во-первых, это конечные покупатели, которые принимают решение о покупке того или иного товара; прогноз цены может повлиять не только на выбор момента покупки, но и на сам факт покупки товара как таковой; во-вторых, это владельцы магазинов (необязательно онлайн), которые планируют закупки и ассортимент товаров; в-третьих, маркетологи, формирующие аналитику изменения рынка тех или иных товаров [**met_pred_online**].

Цель работы – реализовать базу данных, хранящую информацию о покупателях, магазинах, ассортименте товаров в магазинах и историях цен товаров в них, и программное обеспечение для работы с информацией из этой базы данных, а также прогнозирования цен на товары в магазинах посредством построения линии тренда на основе истории цен.

Таким образом, необходимо решить следующие задачи:

- проанализировать варианты представления данных и выбрать из них под-

ходящий для решения задачи;

- проанализировать системы управления базами данных и выбрать подходящую систему для хранения данных;
- проанализировать методы построения линии тренда и выбрать подходящий метод для прогнозирования цен;
- спроектировать базу данных, описать её сущности и связи;
- реализовать интерфейс для работы с базой данных;
- реализовать возможность построения линии тренда для прогнозирования цен на товары в магазинах;
- исследовать, к чему стремится значение цены товара на следующий месяц при увеличении выборки истории цен.

1 Аналитический раздел

В данном разделе формализованы данные, хранимые в базе данных, описана ролевая модель, описаны существующие системы управления базами данных и методы построения линии тренда.

1.1 Формализация данных

База данных должна хранить информацию о:

- магазинах;
- товарах
- ассортименте товаров в магазинах;
- историю цен на товары в магазинах за последние полтора года;
- товарные чеки.

ER-диаграмма сущностей базы данных представлена на рисунке 1.1.

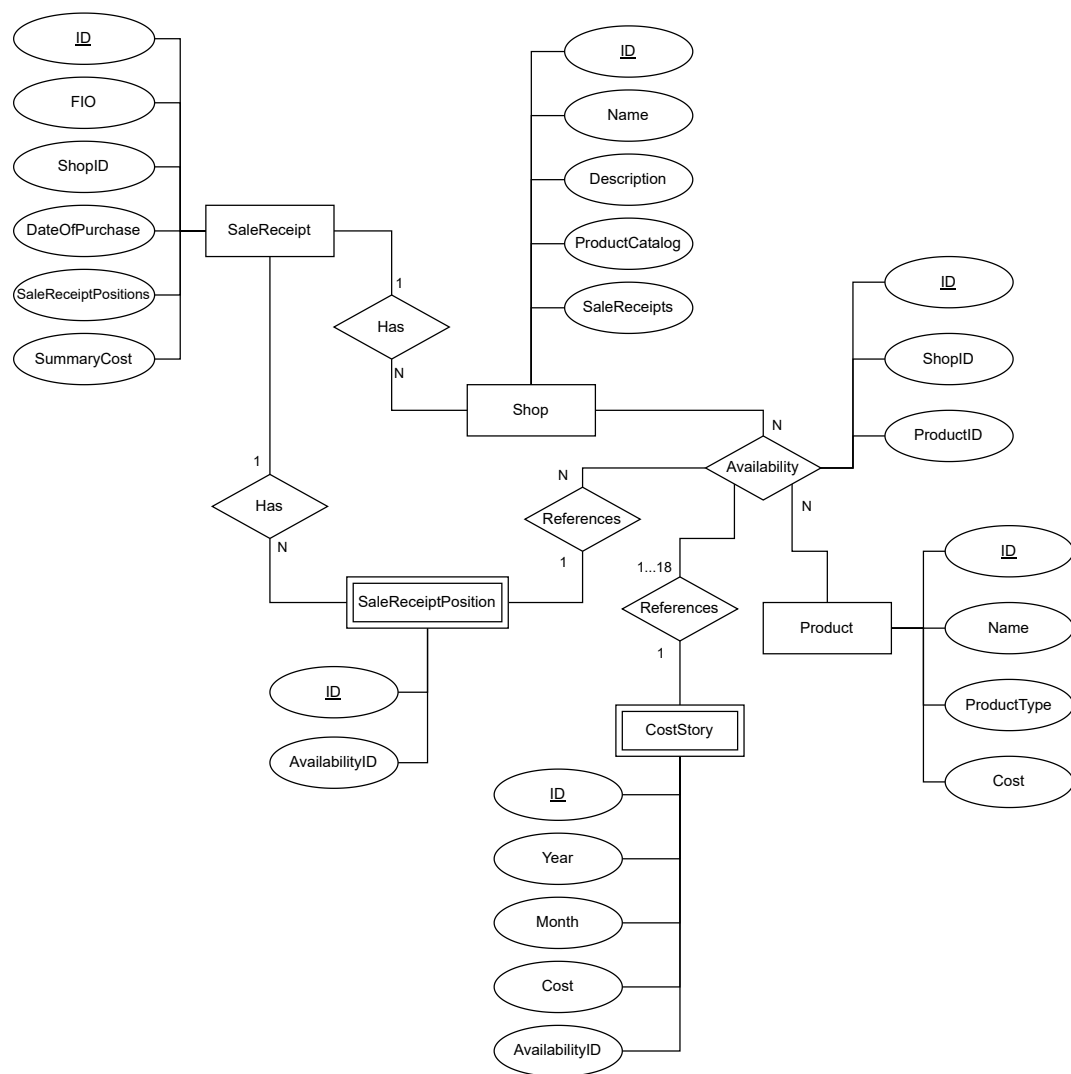


Рисунок 1.1 – ER-диаграмма сущностей базы данных в нотации Цена

Таблица 1.1 – Категории и описание данных

Категория	Описание
Магазин	Название, описание магазина, ассортимент товаров
Товар	Название, тип товара, актуальная цена
Ассортимент товаров	Список товаров
История цен	Год, месяц, цена в этот период
Товарный чек	ФИО, дата покупки, информация о месте покупки(магазине), список купленных товаров, итоговая стоимость

1.2 Описание ролевой модели

Для управления системой введено три роли: пользователь, аналитик, администратор. На рисунке 1.2 представлена Use-case-диаграмма ПО.

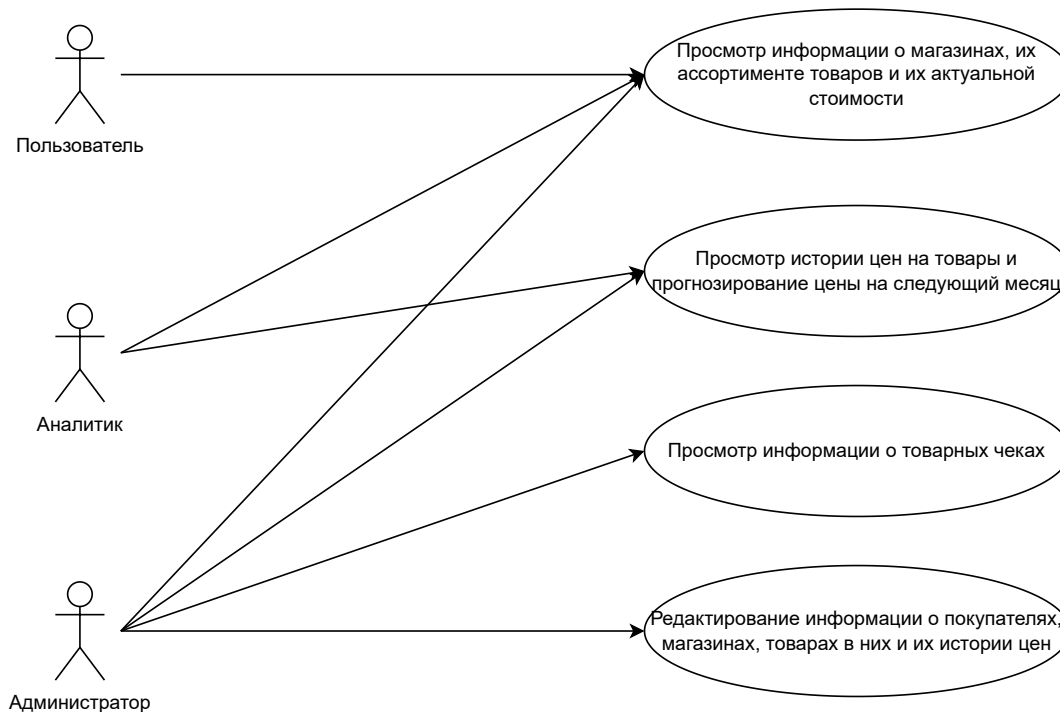


Рисунок 1.2 – Use-case-диаграмма

Таблица 1.2 – Роли и описание их функционала

Роль	Функционал
Пользователь	Просмотр информации о магазинах, их ассортименте товаров и их актуальной стоимости
Аналитик	Просмотр информации о магазинах, их ассортименте товаров и их истории цен. Возможность прогнозирования цены на товар в магазине
Администратор	Просмотр, добавление и удаление информации о магазинах, их ассортименте товаров и их истории цен, товарных чеках. Возможность прогнозирования цены на товар в магазине

Для доступа к ролям «Аналитик» и «Администратор» требуется авторизация (ввод пароля).

1.3 Описание понятия СУБД

Система управления базами данных (сокр. СУБД) – Программная система, предназначенная для создания и хранения базы данных на основе некоторой модели данных, обеспечения логической и физической целостности содержащихся в ней данных, надежного и эффективного использования ресурсов (данных, пространства памяти и вычислительных ресурсов), предоставления к ней санкционированного доступа для приложений и конечных пользователей, а также для поддержки функций администратора базы данных [kogal].

Функции СУБД:

- управление данными во внешней памяти;
- управление данными в оперативной памяти с использованием дискового кэша;

- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД.

1.3.1 Классификация СУБД по модели данных

Модель данных – система типов данных, типов связей между ними и допустимых видов ограничений целостности, которые могут быть для них определены. Здесь имеется в виду современное понимание типа данных как носителя свойств, определяющих и состояние экземпляров типа, и их поведение [kogal].

По этому признаку СУБД делят на:

- дореляционные;
- реляционные;
- постреляционные.

Дореляционные СУБД

К ним относятся иерархические и сетевые СУБД.

Иерархические СУБД

В иерархических СУБД используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

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

Примеры: Caché, Google App Engine Datastore API.

Сетевые СУБД

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

Несмотря на то, что эта модель решает некоторые проблемы, связанные с иерархической моделью, выполнение простых запросов остается достаточно сложным процессом.

Также, поскольку логика процедуры выборки данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Другими словами, если необходимо изменить структуру данных, то нужно изменить и приложение [scienceforum].

Примеры: Caché.

Реляционные СУБД

Реляционные СУБД ориентированы на организацию данных в виде двумерных таблиц. Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

1. каждый элемент таблицы является одним элементом данных;
2. каждый столбец обладает своим уникальным именем;
3. одинаковые строки в таблице отсутствуют;
4. все столбцы в таблице однородные, то есть все элементы в столбце имеют одинаковый тип;
5. порядок следования строк и столбцов может быть произвольным.

Практически все разработчики современных приложений, предусматривающих связь с системами баз данных, ориентируются на реляционные СУБД. По оценке Gartner в 2013 году рынок реляционных СУБД составлял 26 млрд долларов с годовым приростом около 9%, а к 2018 году рынок реляционных СУБД достигнет 40 млрд долларов. В настоящее время абсолютными лидерами рынка СУБД являются компании Oracle, IBM и Microsoft, с общей совокупной

долей рынка около 90%, поставляя такие системы как Oracle Database, IBM DB2 и Microsoft SQL Server [dbms].

Постреляционные СУБД

Постреляционная модель является расширением реляционной модели. Она снимает ограничение неделимости данных, допуская многозначные поля, значения которых состоят из подзначений, и набор значений воспринимается как самостоятельная таблица, встроенная в главную таблицу [post_rel].

С ним относятся объектно-ориентированные и объектно-реляционные СУБД.

Объектно-ориентированные СУБД

Объектно-ориентированные СУБД управляют базами данных, в которых данные моделируются в виде объектов, их атрибутов, методов и классов.

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

Примеры: GemStone.

Объектно-реляционные СУБД

Объектно-реляционные СУБД поддерживают некоторые технологии, реализующие объектно-ориентированный подход: объекты, классы и наследование реализованы в структуре баз данных и языке запросов.

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

В данной работе будет рассмотрен именно этот класс СУБД.

Примеры: PostgreSQL, DB2, Oracle Database, Microsoft SQL Server.

1.4 Методы построения линии тренда

Линия тренда – прямая или кривая линия, аппроксимирующая (приближающая) исходные данные на основе уравнения регрессии или скользящего среднего [lt_excel]. Аппроксимация определяется по методу наименьших квадратов [mnk]. В зависимости от характера поведения исходных данных (убывают, возрастают и т.д.) выбирается метод интерполяции, который следует использовать для построения тренда.

1.4.1 Виды линий трендов

Линейная линия тренда

Линейная линия тренда определяется функцией

$$y = ax + b, \quad (1.1)$$

где a – тангенс угла наклона прямой, b – смещение.

Прямая линия тренда (линейный тренд) наилучшим образом подходит для величин, изменяющихся с постоянной скоростью. Применяется в случаях, когда точки данных расположены близко к прямой [lt_excel].

Логарифмическая линия тренда

Логарифмическая линия тренда определяется функцией

$$y = a \ln x + b, \quad (1.2)$$

где a и b – константы.

Логарифмическая линия тренда соответствует ряду данных, значения которого вначале быстро растут или убывают, а затем постепенно стабилизируются. Может использоваться для положительных и отрицательных данных [lt_excel].

Полиномиальная линия тренда

Полиномиальная линия тренда определяется функцией

$$y = \sum_{i=0}^n a_i x^i, n \leq 6, \quad (1.3)$$

где a_i – коэффициенты полинома.

Полиномиальная линия тренда используется для описания попеременно возрастающих и убывающих данных. Степень полинома подбирают таким образом, чтобы она была на единицу больше количества экстремумов (максимумов и минимумов) кривой [lt_exel].

Экспоненциальная линия тренда

Экспоненциальная линия тренда определяется функцией

$$y = ae^{bx}, \quad (1.4)$$

где a и b – константы.

Экспоненциальный тренд используется в случае непрерывного возрастания изменения данных. Построение указанного тренда невозможно, если в множестве значений членов ряда присутствуют нулевые или отрицательные данные [lt_exel].

Степенная линия тренда

Степенная линия тренда определяется функцией

$$y = ax^b, \quad (1.5)$$

где a и b – константы.

Степенная линия тренда дает хорошие результаты для положительных данных с постоянным ускорением. Для рядов с нулевыми или отрицательными значениями построение указанной линии тренда невозможно [lt_exel].

Линейная фильтрация

Линейная фильтрация определяется формулой

$$F_t = \frac{A_t + A_{t-1} + \dots + A_{t-n+1}}{n}, (2 \leq t < n), \quad (1.6)$$

где n – общее число членов ряда, t – параметр фильтра.

Тренд с линейной фильтрацией позволяет сгладить колебания данных, наглядно демонстрируя характер зависимостей [lt_exel]. Для построения указанной линии тренда пользователь должен задать число t .

Линейная фильтрация использует метод скользящего среднего, при котором каждое значение функции заменяется средним арифметическим A по t соседним точкам, расположенным симметрично относительно данной. Линия тренда в этом случае уравнения не имеет [lin_filt], поэтому для прогноза изменения величины она не подходит.

1.4.2 Анализ исходных данных для построения линии тренда

Цена товара как величина положительная и имеет непостоянный рост: она может как расти, так и падать. Таким образом, для прогнозирования цен на товары подходит полиномиальная линия тренда.

Вывод к аналитическому разделу

В данном разделе были формализованы данные, хранимые в базе данных, описана ролевая модель, рассмотрены существующие СУБД и методы построения линии тренда.

2 Конструкторский раздел

В данном разделе представлена разработанная база данных для поставленной задачи, описаны объекты этой БД и ее система безопасности.

2.1 Разработка базы данных

В базе данных существует 6 таблиц, одна из которых является развязочной (Availability), и одно представление. Они представлены на диаграмме в виде сущностей. На рисунке 2.1 представлена ER-диаграмма сущностей БД.

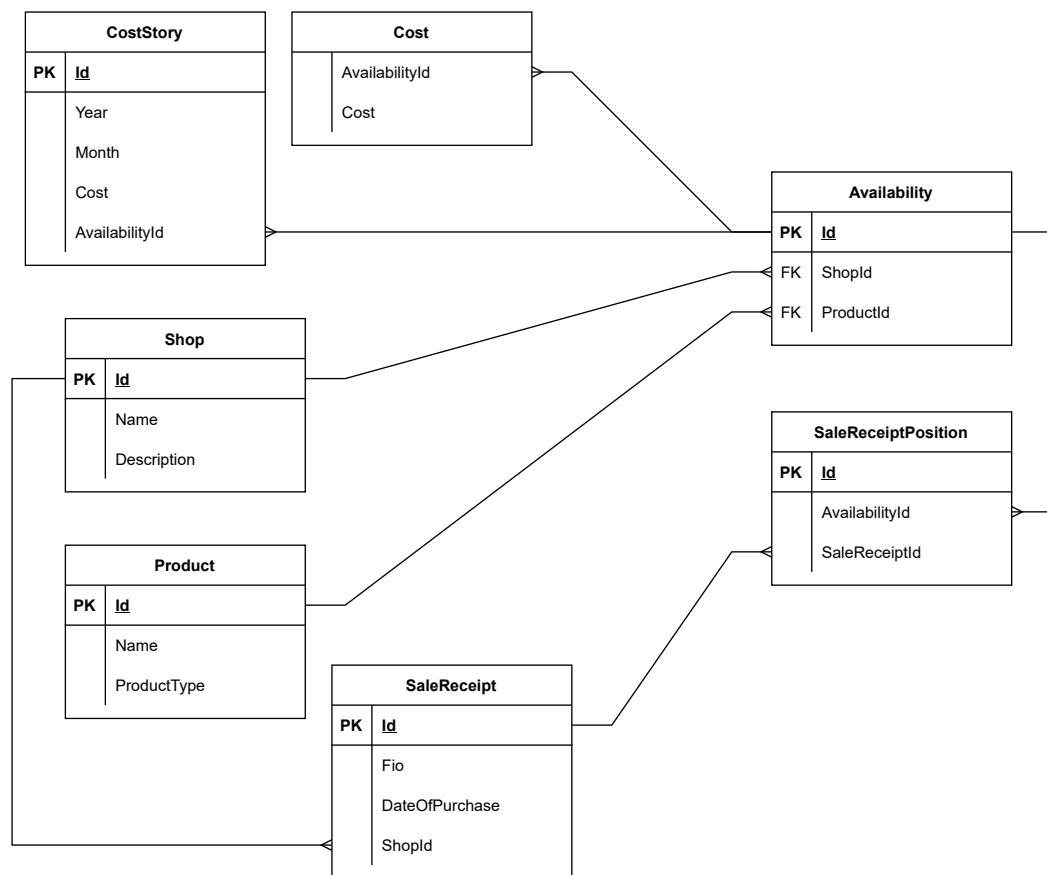


Рисунок 2.1 – ER-диаграмма сущностей БД

Таблица **Shop** хранит данные о магазинах и содержит следующие поля:

- ID – идентификатор магазина;
- Name – название магазина;
- Description – описание магазина.

Таблица **Product** хранит данные о продуктах и содержит следующие поля:

- ID – идентификатор продукта;
- Name – название продукта;
- ProductType – тип продукта.

Таблица **Availability** хранит отношения наличия товара в каталоге магазина и содержит следующие поля:

- ID – идентификатор отношения;
- ShopID – идентификатор магазина;
- ProductID – идентификатор продукта.

Таблица **CostStory** хранит историю цен на товары в магазинах и содержит следующие поля:

- ID – идентификатор значения цены в конкретный год и месяц;
- Year – год;
- Month – месяц;
- Cost – значение цены в конкретный год и месяц;
- AvailabilityID – идентификатор отношения наличия товара в каталоге магазина.

Представление **Cost** хранит актуальное значение цены на товар в магазине и содержит следующие поля:

- AvailabilityID – идентификатор отношения наличия товара в каталоге магазина;
- Cost – значение актуальной цены.

Таблица **SaleReceipt** хранит данные о товарных чеках и содержит следующие поля:

- ID – идентификатор товарного чека;

- FIO – ФИО покупателя;
- ShopID – идентификатор магазина, в котором была совершена покупка;
- DateOfPurchase – дата покупки.

Таблица **SaleReceiptPosition** хранит информацию о позиции в чеке и содержит следующие поля:

- ID – идентификатор позиции чека;
- AvailabilityID – идентификатор отношения наличия товара в каталоге магазина.

Таблицы **SaleReceiptPosition** и **CostStory** содержат поле идентификатора отношения наличия товара в каталоге магазина в целях борьбы с рассинхронизацией: товарный чек и история цен на товар в магазине не могут ссылаться на товары, которых нет в данном магазине.

Актуальная стоимость товара в магазине формируется как последнее по дате значение цены товара в истории цен.

2.2 Разработка объектов базы данных

Для разработанной базы данных были созданы 4 хранимые функции на выборку и триггер для обновления истории цен. Далее представлены их реализации и описание.

Листинг 2.1 – Реализация хранимой функции `get_products_by_shopid`

```
create or replace function get_products_by_shopid(shop_id integer)
returns table(ProdID integer, ProdName text, ProdType text, Cost
float)
as $$
    select APC.ProductID, APC.Name, APC.ProductType, APC.Cost
    from (((select Availability.ID as AvailID,
        Availability.ShopID, Availability.ProductID from
        Availability where Availability.ShopID = shop_id) as A
    join Products on A.ProductID = Products.ID) as AP join Costs
        on AP.AvailID = Costs.AvailabilityID) as APC;
$$ language sql;
```

Функция **get_products_by_shopid** возвращает список товаров, продающиеся в указанном магазине с идентификатором shop_id.

Листинг 2.2 – Реализация хранимой функции get_coststory_by_shopid_prodid

```
create or replace function get_coststory_by_shopid_prodid(shop_id
    integer, prod_id integer)
returns table(Year integer, Month integer, Cost integer)
as $$
    select CostStory.Year, CostStory.Month, CostStory.Cost
    from CostStory
    where AvailabilityID = (select ID
    from Availability
    where ShopID = shop_id and ProductID = prod_id);
$$ language sql;
```

Функция **get_coststory_by_shopid_prodid** предназначена для поиска полной истории цен на указанный товар в указанном магазине посредством их идентификаторов prod_id и shop_id соответственно.

Листинг 2.3 – Реализация хранимой функции get_salereceipts_by_shopid

```
create or replace function get_salereceipts_by_shopid(shop_id
    integer)
returns table(SR_ID integer, FIO text, DateOfPurchase date,
    SummaryCost integer)
as $$
    select S.S_ID, max(FIO), max(DateOfPurchase), sum(Cost) as
        SummaryCost
    from ((select ID as S_ID, FIO, ShopID, DateOfPurchase from
        SaleReceipts where ShopID = shop_id) as SR join
        SaleReceiptPositions on SaleReceiptPositions.SaleReceiptID
        = SR.S_ID) as S join Costs on S.AvailabilityID =
        Costs.AvailabilityID
    group by S.S_ID;
$$ language sql;
```

Функция **get_salereceipts_by_shopid** возвращает список товарных чеков в указанном магазине посредством его идентификатора shop_id.

Листинг 2.4 – Реализация хранимой функции get_content_from_salereceipt

```
create or replace function get_content_from_salereceipt(sr_id
    integer)
```

```

returns table(ProdID integer, Name text, ProductType text, Cost
integer)
as $$
    select SAP.ProductID as ProdID, Name, ProductType,
        Cost::integer
    from ((SaleReceiptPositions join Availability on
        SaleReceiptPositions.AvailabilityID = Availability.ID) as
        SA join Products on Products.ID = SA.ProductID) as SAP join
        Costs on SAP.AvailabilityID = Costs.AvailabilityID
    where SaleReceiptID = sr_id;
$$ language sql;

```

Функция **get_content_from_salereceipt** возвращает список товаров товарного чека с идентификатором `sr_id`.

Сценарий создания всей базы данных (вместе с триггером) расположен в Приложении А.

2.3 Разработка системы безопасности БД

В соответствии с описанной в Аналитическом разделе ролевой моделью, были созданы три пользователя с различными уровнями доступа к данным в БД.

- «Пользователь»: имеет доступ только на чтение к таблицам магазинов `Shops`, товаров `Products`, стоимостей `Costs` и отношения наличия `Availability`.
- «Аналитик»: имеет доступ только на чтение к таблицам магазинов `Shops`, товаров `Products`, стоимостей `Costs`, истории цен на товары в магазинах `CostStory` и отношения наличия `Availability`.
- «Администратор»: имеет доступ ко всем таблицам БД с наличием прав на редактирование информации в них.

Как было сказано в Аналитическом разделе, для доступа к ролям «Аналитик» и «Администратор» требуется авторизация (ввод пароля).

Вывод к конструкторскому разделу

В данном разделе была представлена разработанная база данных для поставленной задачи, описаны объекты этой БД и ее система безопасности.

3 Технологический раздел

В данном разделе представлены средства реализации и интерфейс программного обеспечения.

3.1 Средства реализации

Обзор и выбор СУБД

Для выбора СУБД формулируются следующие критерии:

- открытость;
- поддержка хранимых процедур и триггеров;
- кроссплатформенность;
- поддержка БД неограниченного размера;

Далее представлена таблица сравнения рассматриваемых СУБД по заданным критериям [orc_db; mssql; pg; db2].

Таблица 3.1 – Сравнение СУБД по критериям

	Oracle DB	SQL Server	DB2	PostgreSQL
Открытость	-	-	-	+
Поддержка хранимых процедур и триггеров	+	+	-	+
Кроссплатформенность	+	-	+	+
Поддержка БД неограниченного размера	+	-	-	+

Основываясь на таблице сравнения СУБД, для решения задачи была выбрана СУБД PostgreSQL.

Выбор и обоснование средств реализации ПО

В качестве языка программирования для реализации программного обеспечения для работы с БД был выбран C# [**c_sharp**]. Выбор этого языка обусловлен тем, что он обладает удобным синтаксисом, управляемым кодом и сборщиком мусора, благодаря этому не нужно заботиться об утечках памяти, об указателях и о некоторых базовых структурах и алгоритмах – все это уже реализовано. Это позволит ускорить разработку и отладку кода. Также данный язык предоставляет большую часть требуемого функционала для решения поставленной задачи, для недостающего функционала существует связанным с ним пакетным менеджер NuGet [**nuget**].

Для разработки пользовательского интерфейса программного обеспечения была выбрана платформа WPF [**wpf**]. Данная платформа обладает декларативным определением элементов интерфейса с помощью языка разметки XAML [**xaml**], независимостью от разрешения экрана. Это значит, что приложение будет корректно масштабироваться под разные экраны с разным разрешением, а также данный интерфейс не будет жёстко зависеть от логики программы.

В качестве среды разработки (IDE) была выбрана Visual Studio [**vs**], обладающая интеллектуальными подсказками, инструментами анализа, отладки и тестирования кода, поставляющаяся вместе с языком C# и пакетным менеджером NuGet.

3.2 Интерфейс ПО

Интерфейс включает в себя 4 секции для работы с БД.

1. Работа с магазинами.
2. Работа с товарами магазинов.
3. Работа с товарными чеками магазинов.
4. Работа с историей цен товаров в магазинах.

На рисунках 3.1-3.3 продемонстрирована работоспособность программы.

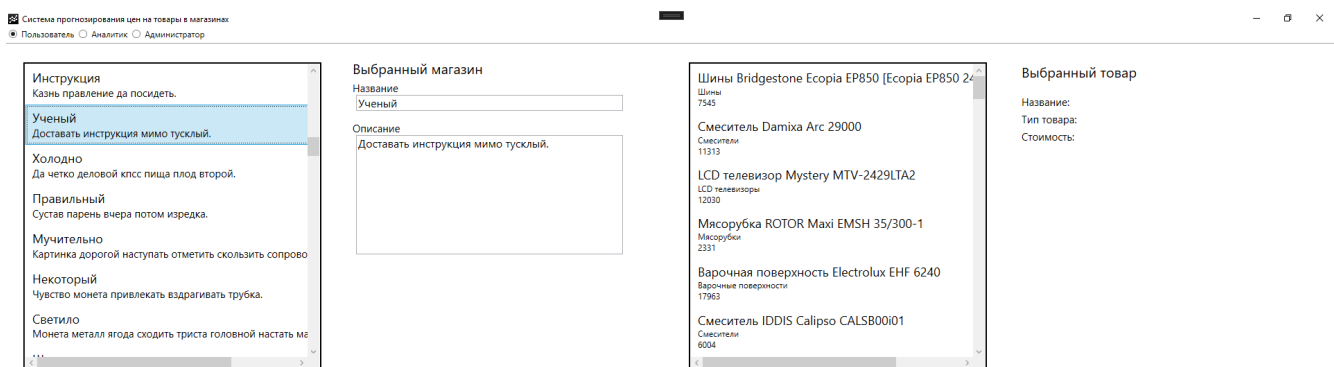


Рисунок 3.1 – Работа приложения в режиме пользователя

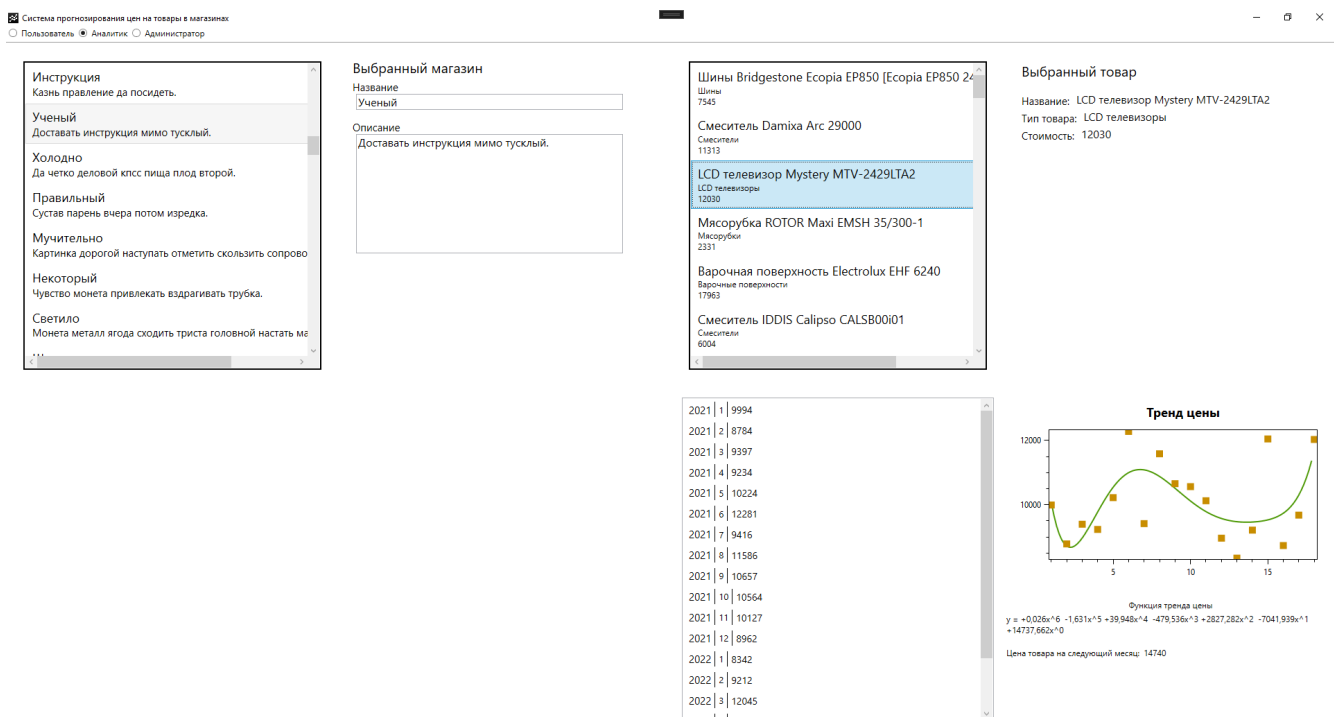


Рисунок 3.2 – Работа приложения в режиме аналитика

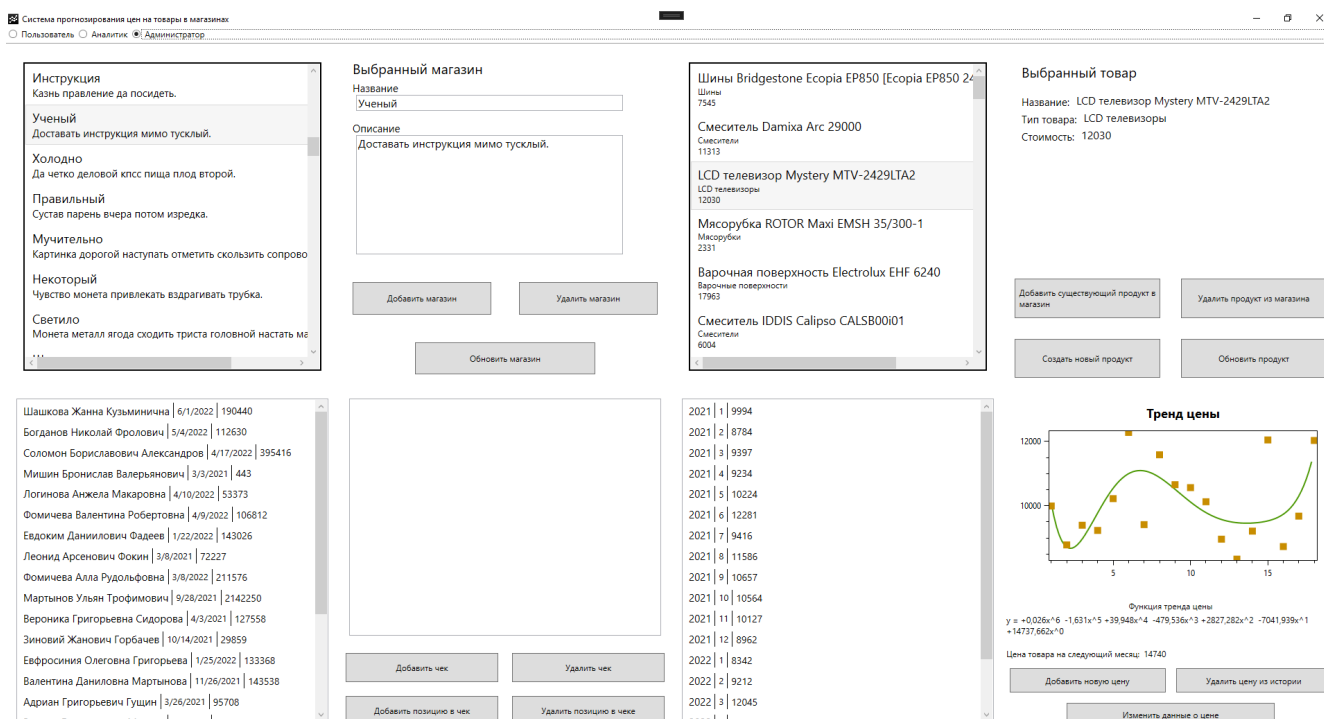


Рисунок 3.3 – Работа приложения в режиме администратора

Вывод к технологическому разделу

В данном разделе были представлены средства реализации и интерфейс программного обеспечения.

4 Экспериментальный раздел

В данном разделе поставлен эксперимент по оценке отклонения значения стоимости товара на следующий месяц при неполной выборке истории цен от реального значения стоимости товара на следующий месяц.

4.1 Постановка эксперимента

4.1.1 Цель эксперимента

Целью эксперимента является проведение анализа отклонения значения стоимости товара на следующий месяц при неполной выборке истории цен от реального значения стоимости товара на следующий месяц.

4.1.2 Описание эксперимента

В данном эксперименте был выбран товар с указанной в таблице 4.1 историей цен. Далее при расчете очередного значения стоимости товара на следующий месяц при неполной выборке истории цен объем выборки будет ограничиваться от 1 до $N = 18$, где N – объем полной выборки.

Таблица 4.1 – История цен выбранного для эксперимента товара

Год	Месяц	Цена
2021	1	16883
2021	2	17283
2021	3	15860
2021	4	17194
2021	5	20262
2021	6	16655
2021	7	16983
2021	8	13709
2021	9	19365
2021	10	17504
2021	11	18579
2021	12	14264
2022	1	14831
2022	2	19490
2022	3	16791
2022	4	22657
2022	5	21632
2022	6	18796

4.2 Результаты эксперимента

В результате эксперимента были получены значения, представленные в таблице 4.2, где

- n – количество используемых значений для вычисления стоимости товара на следующий месяц;
- \hat{y} – аппроксимированное значение стоимости товара на следующий месяц;
- y – реальное значение стоимости товара на следующий месяц;

- $\frac{|\hat{y}-y|}{y}$ – среднее значение относительной погрешности.

Таблица 4.2 – Результаты эксперимента

$n = 1..N$	\hat{y}	y	$\frac{ \hat{y}-y }{y}, \%$
1	16883	17283	2,31
2	17283	15860	8,97
3	15859	17194	7,76
4	17194	20262	15,14
5	20342	16655	22,13
6	16662	16983	1,89
7	16952	13709	23,65
8	13735	19365	29,07
9	19292	17504	10,21
10	17560	18579	5,48
11	18296	14264	28,26
12	14155	14831	4,55
13	14549	19490	25,35
14	19561	16791	16,49
15	17579	22657	22,41
16	22544	21632	4,21
17	22183	18796	18,01
18	18939	18796	0,76

Вывод к экспериментальному разделу

В данном разделе был поставлен эксперимент по оценке отклонения значения стоимости товара на следующий месяц при неполной выборке истории цен от реального значения стоимости товара на следующий месяц.

Из результатов эксперимента видно, что иногда ошибка не превышает 5%, а иногда превышает более 15%, что говорит о некачественности модели.

Можно сделать вывод о том, что использование полинома шестой степени для предсказания следующего значения на основе предыдущих не всегда является лучшим решением: несмотря на большую степень полинома, которая увеличивает точность аппроксимации, то есть уменьшает сумму квадратов, ошибка возрастает из-за наличия больших флуктуаций между точками.

Заключение

В ходе выполнения курсового проекта были формализованы данные, хранимые в базе данных, описана ролевая модель, рассмотрены существующие СУБД и методы построения линии тренда. Была представлена разработанная база данных для поставленной задачи, описаны объекты этой БД и ее система безопасности. Также были представлены средства реализации и интерфейс программного обеспечения для выполнения данной задачи.

Созданная база данных позволяет хранить информацию о магазинах, товарах в них, товарных чеках и их содержимом, а также истории цен на товары в магазинах. Созданный программный продукт для работы с базой данных позволяет работать с информацией из нее: просматривать, добавлять, редактировать, удалять, анализировать.

Также был поставлен эксперимент, в ходе которого были выявлены изъяны модели для предсказания стоимости товара на основе полинома n -ой степени.

В качестве развития проекта можно предложить реализацию более качественных моделей для прогнозирования стоимостей товаров на следующий месяц для лучшей аппроксимации цены на товар в магазине.

Список использованных источников

- [1] Методы и модели социально-экономического прогнозирования : учебник и практикум для академического бакалавриата. В 2-х т. Т. 1. Теория и методология прогнозирования / И. С. Светульников, С. Г. Светульников. — М. : Издательство Юрайт, 2014. — 351 с. — Серия : Бакалавр. Академический курс.
- [2] Касперович С.А. Прогнозирование и планирование экономики : курс лекций для студентов специальностей 1-25 01 07 «Экономика и управление предприятием», 1-25 01 08 «Бухгалтерский учет, анализ и аудит», 1-26 02 02 «Менеджмент», 1-26 02 03 «Маркетинг» / С. А. Касперович. - Минск. : БГТУ, 2007. - 172 с.
- [3] К. В. Богданов. Методика прогнозирования цен на товары в интернет-магазинах [Электронный ресурс]. — Режим доступа: <https://cyberleninka.ru/article/n/metodika-prognozirovaniya-tsen-na-tovary-v-internet-magazinah/viewer> (дата обращения 4.04.2022)
- [4] Когаловский М.Р. Энциклопедия технологий баз данных. — М.: Финансы и статистика, 2002. — 800 с.
- [5] Классификация систем управления базами данных [Электронный ресурс]. — Режим доступа: <https://scienceforum.ru/2016/article/2016019197> (дата обращения 6.04.2022)
- [6] Системы управления базами данных [Электронный ресурс]. — Режим доступа: <https://lecturesdb.readthedocs.io/databases/dbms.html> (дата обращения 8.04.2022)
- [7] Постреляционная, многомерная и объектно-ориентированная модели данных [Электронный ресурс]. — Режим доступа: http://bseu.by/it/tohod/lekci2_4.htm?ysclid=l3njaseoko (дата обращения 8.04.2022)

- [8] Построение линии тренда в Excel [Электронный ресурс]. – Режим доступа: <https://ya-znau.ru/znaniya/zn/202> (дата обращения 9.04.2022)
- [9] Линник Ю. В. Метод наименьших квадратов и основы математико-статистической теории обработки наблюдений. — 2-е изд. — М., 1962. (математическая теория)
- [10] Линейная фильтрация [Электронный ресурс]. – Режим доступа: <https://poznayka.org/s23136t1.html> (дата обращения 9.04.2022)
- [11] Oracle Database [Электронный ресурс]. – Режим доступа: <https://www.oracle.com/database/> (дата обращения 12.04.2022)
- [12] Microsoft SQL-Server [Электронный ресурс]. – Режим доступа: <https://www.microsoft.com/ru-ru/sql-server/sql-server-2019> (дата обращения 12.04.2022)
- [13] PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/> (дата обращения 12.04.2022)
- [14] DB2 [Электронный ресурс]. – Режим доступа: <https://www.ibm.com/analytics/db2> (дата обращения 12.04.2022)
- [15] Документация по C# [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения 14.04.2022)
- [16] NuGet [Электронный ресурс]. – Режим доступа: <https://www.nuget.org/> (дата обращения 14.04.2022)
- [17] Что такое Windows Presentation Foundation (WPF)? [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/designers/getting-started-with-wpf?view=vs-2022> (дата обращения 14.04.2022)
- [18] Обзор XAML (WPF .NET) [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/designers/getting-started-with-wpf?view=vs-2022>

ru/dotnet/desktop/wpf/xaml/?view=netdesktop-6.0 (дата обращения 14.04.2022)

[19] Visual Studio [Электронный ресурс]. – Режим доступа: <https://visualstudio.microsoft.com/ru/> (дата обращения 14.04.2022)

Приложение

Приложение А

Листинг 4.1 – Реализация триггера для обновления истории цен

```
create or replace function remove_too_old_coststory()
returns trigger
as $$
    declare
        old_date_id integer;
        old_date date;
        new_date_id integer;
        new_date date;
        months_diff integer;
        new_avail_id integer;
    begin
        new_avail_id := new.AvailabilityID;

        select min(make_date(prod_coststory.Year,
            prod_coststory.Month, 1)) into old_date
        from (select *
        from CostStory
        where AvailabilityID = new_avail_id) as prod_coststory;

        select prod_coststory.id into old_date_id
        from (select *
        from CostStory
        where AvailabilityID = new_avail_id) as prod_coststory
        where make_date(prod_coststory.Year, prod_coststory.Month,
            1) = old_date;

        select max(make_date(prod_coststory.Year,
            prod_coststory.Month, 1)) into new_date
        from (select *
        from CostStory
        where AvailabilityID = new_avail_id) as prod_coststory;
```

```

select prod_coststory.id into new_date_id
from (select *
from CostStory
where AvailabilityID = new_avail_id) as prod_coststory
where make_date(prod_coststory.Year, prod_coststory.Month,
1) = new_date;

select (date_part('year', new_date) - date_part('year',
old_date)) * 12 + (date_part('month', new_date) -
date_part('month', old_date)) + 1
into months_diff;

if months_diff > 18 then
delete from CostStory where AvailabilityID = new_avail_id
and ID = old_date_id;
end if;
return new;
end
$$ language plpgsql;

drop trigger update_coststory on CostStory;
create trigger update_coststory after insert on CostStory
for each row execute function remove_too_old_coststory();

```

Листинг 4.2 – Сценарий создания БД

```

Drop database if exists spsr_lt_db;
Create database spsr_lt_db;

```

```

create table Shops
(
    ID serial primary key,
    Name text not null,
    Description text not null
);

```

```

create table SaleReceipts
(
    ID serial primary key,

```

```

        FIO text not null,
        ShopID integer not null,
        DateOfPurchase date not null,
        foreign key (ShopID) references Shops(ID) on delete cascade
    );

create table Products
(
    ID serial primary key,
    Name text not null,
    ProductType text not null
);

create table Availability
(
    ID serial primary key,
    ShopID integer not null,
    ProductID integer not null,
    foreign key (ShopID) references Shops(ID) on delete cascade,
    foreign key (ProductID) references Products(ID) on delete
        cascade
);

create table SaleReceiptPositions
(
    ID serial primary key,
    AvailabilityID integer not null,
    SaleReceiptID integer not null,
    foreign key (AvailabilityID) references Availability(ID) on
        delete cascade,
    foreign key (SaleReceiptID) references SaleReceipts(ID) on
        delete cascade
);

create table CostStory
(
    ID serial primary key,
    Year integer not null,
    Month integer not null,

```

```

        Cost integer not null,
        AvailabilityID integer not null,
        foreign key (AvailabilityID) references Availability(ID) on
            delete cascade
    );

create view Costs as
    select T.AvailabilityID, Cost
    from CostStory join (select AvailabilityID,
        max(make_date(Year, Month, 1)) as CostDate
    from CostStory
    group by AvailabilityID) as T on T.AvailabilityID =
        CostStory.AvailabilityID and T.CostDate =
            make_date(CostStory.Year, CostStory.Month, 1);

create or replace function remove_too_old_coststory()
returns trigger
as $$
    declare
        old_date_id integer;
        old_date date;
        new_date_id integer;
        new_date date;
        months_diff integer;
        new_avail_id integer;
    begin
        new_avail_id := new.AvailabilityID;

        select min(make_date(prod_coststory.Year,
            prod_coststory.Month, 1)) into old_date
        from (select *
        from CostStory
        where AvailabilityID = new_avail_id) as prod_coststory;

        select prod_coststory.id into old_date_id
        from (select *
        from CostStory

```

```

where AvailabilityID = new_avail_id) as prod_coststory
where make_date(prod_coststory.Year, prod_coststory.Month,
    1) = old_date;

select max(make_date(prod_coststory.Year,
    prod_coststory.Month, 1)) into new_date
from (select *
from CostStory
where AvailabilityID = new_avail_id) as prod_coststory;

select prod_coststory.id into new_date_id
from (select *
from CostStory
where AvailabilityID = new_avail_id) as prod_coststory
where make_date(prod_coststory.Year, prod_coststory.Month,
    1) = new_date;

select (date_part('year', new_date) - date_part('year',
    old_date)) * 12 + (date_part('month', new_date) -
    date_part('month', old_date)) + 1
into months_diff;

if months_diff > 18 then
delete from CostStory where AvailabilityID = new_avail_id
    and ID = old_date_id;
end if;
return new;
end
$$ language plpgsql;

drop trigger update_coststory on CostStory;
create trigger update_coststory after insert on CostStory
for each row execute function remove_too_old_coststory();

create user "user";
grant connect on database spsr_lt_db to "user";
grant usage on schema public to "user";
grant select on table Shops, Products, Availability, Costs to

```

```

    "user";
alter user "user" with password 'user';

create user "analyst" with password 'analyst';
grant connect on database spsr_lt_db to "analyst";
grant usage on schema public to "analyst";
grant select on table Shops, Products, Availability, CostStory to
    "analyst";
grant select on table Costs to "analyst";

create user "admin" with password 'admin';
grant connect on database spsr_lt_db to "admin";
grant usage on schema public to "admin";
grant select, insert, update, delete on all tables in schema
    public to "admin";

drop user if exists "user";
drop user if exists "analyst";
drop user if exists "admin";

create or replace function get_products_by_shopid(shop_id integer)
returns table(ProdID integer, ProdName text, ProdType text, Cost
    float)
as $$
    select APC.ProductID, APC.Name, APC.ProductType, APC.Cost
    from (((select Availability.ID as AvailID,
        Availability.ShopID, Availability.ProductID from
        Availability where Availability.ShopID = shop_id) as A
    join Products on A.ProductID = Products.ID) as AP join Costs
        on AP.AvailID = Costs.AvailabilityID) as APC;
$$ language sql;

create or replace function get_coststory_by_shopid_prodid(shop_id
    integer, prod_id integer)
returns table(Year integer, Month integer, Cost integer)
as $$
    select CostStory.Year, CostStory.Month, CostStory.Cost
    from CostStory
    where AvailabilityID = (select ID

```

```

        from Availability
        where ShopID = shop_id and ProductID = prod_id);
$$ language sql;

create or replace function get_salereceipts_by_shopid(shop_id
integer)
returns table(SR_ID integer, FIO text, DateOfPurchase date,
SummaryCost integer)
as $$
    select S.S_ID, max(FIO), max(DateOfPurchase), sum(Cost) as
        SummaryCost
    from ((select ID as S_ID, FIO, ShopID, DateOfPurchase from
        SaleReceipts where ShopID = shop_id) as SR join
        SaleReceiptPositions on SaleReceiptPositions.SaleReceiptID
        = SR.S_ID) as S join Costs on S.AvailabilityID =
        Costs.AvailabilityID
    group by S.S_ID;
$$ language sql;

create or replace function get_content_from_salereceipt(sr_id
integer)
returns table(ProdID integer, Name text, ProductType text, Cost
integer)
as $$
    select SAP.ProductID as ProdID, Name, ProductType,
        Cost::integer
    from ((SaleReceiptPositions join Availability on
        SaleReceiptPositions.AvailabilityID = Availability.ID) as
        SA join Products on Products.ID = SA.ProductID) as SAP join
        Costs on SAP.AvailabilityID = Costs.AvailabilityID
    where SaleReceiptID = sr_id;
$$ language sql;

copy Products from '...\tables_data\Products.csv' delimiter ','
CSV HEADER;
copy Shops from '...\tables_data\Shops.csv' delimiter ',' CSV
HEADER;
copy SaleReceipts from '...\tables_data\SaleReceipts.csv'

```



```
delimiter ',' CSV HEADER;  
copy Availability from '...\tables_data\Availability.csv'  
delimiter ',' CSV HEADER;  
copy SaleReceiptPositions from  
    '...\tables_data\SaleReceiptPositions.csv' delimiter ',' CSV  
    HEADER;  
copy CostStory from '...\tables_data\CostStory.csv' delimiter ','  
    CSV HEADER;
```

Приложение Б