МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ НИЖЕГОРОДСКОЙ ОБЛАСТИ

Государственное бюджетное профессиональное образовательное учреждение НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Специальность 09.02.07 Информационные системы и программирование Квалификация Программист

# КУРСОВАЯ РАБОТА

МДК 11.01 Технология разработки и защиты баз данных

Тема: «Проектирование и разработка базы данных для магазина компьютерной техники»

|  |  |
| --- | --- |
| Выполнил студент группы  3ИСиП-22-2 | Проверил преподаватель Гутянская Е.М. |
| Пронин Дмитрий Алексеевич | Проект защищен с оценкой  Дата защиты Подпись |

Нижний Новгород, 2024

# Оглавление

[КУРСОВАЯ РАБОТА 1](#_Toc1)

[Оглавление 2](#_Toc2)

[Введение 3](#_Toc3)

[Задачи работы: 3](#_Toc4)

[Ожидаемый результат работы: 3](#_Toc5)

# Введение

В современном мире компьютерная техника занимает важное место в жизни пользователей и организаций. С каждым годом увеличивается спрос на компьютеры, что ставит перед магазинами задачу эффективного управления товарными запасами, учётом продуктов и обслуживанием клиентов.

Ключевым элементом для достижения этих целей является создание и функционирование базы данных, специально разработанной для магазина компьютерной техники.

Целью данной курсовой работы является проектирование и разработка базы данных и информационной системы для магазина компьютерной техники. Это позволит оптимизировать управление складскими операциями, автоматизировать процессы учёта товаров и обслуживания клиентов, а также повысить общую эффективность работы магазина.

## Задачи работы:

1. Анализ требований и потребностей магазина компьютерной техники.
2. Проектирование структуры базы данных с учётом специфики магазина.
3. Разработка функциональной части информационной системы для автоматизации бизнес-процессов.
4. Реализация и интеграция базы данных и информационной системы.
5. Оценка эффективности использования базы данных и информационной системы в работе магазина.

## Ожидаемый результат работы:

Готовая функционирующая база данных и информационная система для магазина компьютерной техники.В процессе выполнения данной курсовой работы будут применены современные методы и инструменты проектирования баз данных и информационных систем, что позволит создать удобную, надёжную и эффективную систему управления информацией для магазина компьютерной техники. Исследование лучших практик в области проектирования баз данных поможет учесть все необходимые функции и требования, обеспечивая оптимальное использование ресурсов и повышая конкурентоспособность магазина на рынке.

# Теоретические основы разрабатываемой темы

## Анализ проектируемой системы

В рамках разработанной системы для магазина компьютеров будут реализованы следующие функции:

* Многопользовательский доступ
* Разграничение функционала по ролям
* Клиент и сотрудник магазина могут осуществлять следующие действия:
  + получать оперативную информацию о наличии, описании, фото и стоимости техники;
  + подбирать технику по заявленным критериям;
  + осуществлять отбор техники по категориям, производителям;
  + находить необходимую информацию, содержащую описание техники и комплектующих;
* Сотрудник магазина может осуществлять следующие действия:
  + вести справочники (добавление, удаление, редактирование);
  + оформлять заказ на товар (в одном заказе может быть несколько различных товаров в разном количестве);
  + стоимость заказа рассчитывается динамически.

Данные в системе будут храниться в структурированном формате в базе данных:

* Пользователи. В этой таблице хранится информация о пользователях системы, включая их имя пользователя, адрес электронной почты, имя, фамилию, пароль и роль.
* Процессоры. В этой таблице хранится информация о процессорах, включая их название, бренд, количество ядер и потоков, базовую и турбочастоты.
* Видеокарты. В этой таблице хранится информация о графических картах, включая название, бренд, объем памяти и тип памяти.
* Блоки питания. В этой таблице хранится информация о блоках питания, включая название, бренд, мощность и рейтинг эффективности.
* Материнские платы. В этой таблице хранится информация о материнских платах, включая название, бренд, тип сокета и форм-фактор.
* Охладители. В этой таблице хранится информация об охладителях, включая название, бренд, тип и мощность охлаждения.
* Корпуса. В этой таблице хранится информация о корпусах компьютеров, включая название, бренд, форм-фактор и цвет.
* Оперативная память (RAM). В этой таблице хранится информация об оперативной памяти, включая название, бренд, объем и скорость.
* Компьютеры. В этой таблице хранится информация обо всех компьютерах, доступных для продажи, включая описание, цену и ссылки на компоненты (процессор, видеокарта и т.д.).
* Корзина покупок. В этой таблице хранится информация о товарах в корзине пользователей, включая владельца корзины и количество каждого компьютера.
* Заказы. В этой таблице хранится информация о заказах клиентов, включая дату заказа, общую сумму и статус заказа.
* Элементы заказа. В этой таблице хранится информация о конкретных товарах в каждом заказе, включая количество и цену каждого товара.

Рассмотрим функции, которые будут реализованы в нашем приложении:

* Многопользовательский доступ:

Это возможность одновременного доступа нескольких пользователей к сервису.

* Разграничение функционала по ролям:  
  Это метод управления доступом, при котором права доступа к различным функциям системы предоставляются на основе ролей, которые назначаются пользователям. Каждая роль определяет свой набор прав доступа. Этот подход позволяет эффективно управлять правами доступа и обеспечивает безопасность информации в системе.

В нашей программе сотрудник магазина будет иметь доступ ко всем функциям и данным, в то время как клиенты будут иметь ограниченный доступ к определенным функциям или данным.

Функции, доступные только для сотрудника магазина:

* + Получение любой информации о товаре;
  + Ведение справочников (добавление, удаление, редактирование);
  + Оформление заказа за пользователя (доступно поле комментария при оформлении заказа, в которое можно указать контактную информацию о заказчике);

Функции, доступные для всех:

* + Получение информации о названии, описании, стоимости компьютера;
  + Получение информации о комплектующих, установленных в ПК;
  + Осуществлять фильтрацию продуктов в каталоге;
  + Составление своей корзины покупок;
  + Просмотр информации о всех своих заказах

# Проектирование базы данных информационной системы

## Информационно-логическая модель базы данных

Информационно-логическая модель отображает данные предметной области в виде совокупности информационных объектов и связей между ними. Эта модель представляет данные, подлежащие хранению в базе данных.

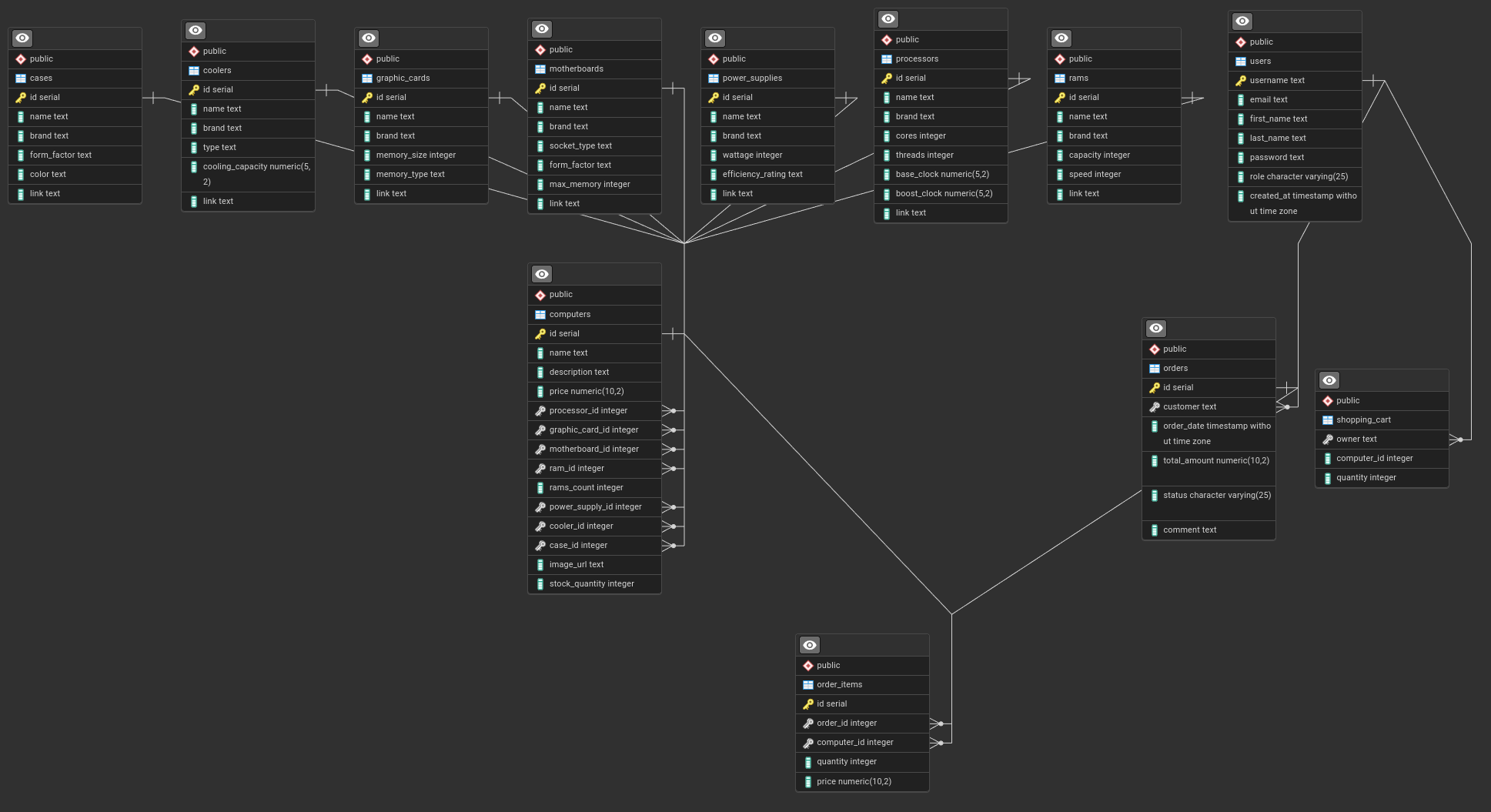


Рисунок 1. Информационно-логическая модель базы данных

## Словарь данных

В приведённых ниже таблицах описаны поля всех таблиц базы данных, используемых в разработанной информационной системе.

| **users** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| username | Имя пользователя | TEXT | PK |
| email | Электронная почта | TEXT |  |
| first\_name | Имя | TEXT |  |
| last\_name | Фамилия | TEXT |  |
| password | Пароль | TEXT |  |
| role | Роль | VARCHAR(25) |  |
| created\_at | Дата создания | TIMESTAMP |  |

| **processors** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| cores | Количество ядер | INT |  |
| threads | Количество потоков | INT |  |
| base\_clock | Базовая частота | DECIMAL(5,2) |  |
| boost\_clock | Турбочастота | DECIMAL(5,2) |  |
| link | Ссылка | TEXT |  |

| **graphic\_cards** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| memory\_size | Объем памяти | INT |  |
| memory\_type | Тип памяти | TEXT |  |
| link | Ссылка | TEXT |  |

| **power\_supplies** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| wattage | Мощность | INT |  |
| efficiency\_rating | Рейтинг эффективности | TEXT |  |
| link | Ссылка | TEXT |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| socket\_type | Тип сокета | TEXT |  |
| form\_factor | Форм-фактор | TEXT |  |
| max\_memory | Максимальный объем памяти | INT |  |
| link | Ссылка | TEXT |  |

| **coolers** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| type | Тип охладителя | TEXT |  |
| cooling\_capacity | Мощность охлаждения | DECIMAL(5,2) |  |
| link | Ссылка | TEXT |  |

| **cases** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| form\_factor | Форм-фактор | TEXT |  |
| color | Цвет | TEXT |  |
| link | Ссылка | TEXT |  |

| **rams** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| brand | Бренд | TEXT |  |
| capacity | Объем памяти в ГБ | INT |  |
| speed | Скорость в МГц | INT |  |
| link | Ссылка | TEXT |  |

| **computers** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор | SERIAL | PK |
| name | Название | TEXT |  |
| description | Описание | TEXT |  |
| price | Цена | DECIMAL(10,2) |  |
| processor\_id | Идентификатор процессора | INT NOT NULL REFERENCES processors(id) | FK |
| graphic\_card\_id | Идентификатор видеокарты | INT NOT NULL REFERENCES graphic\_cards(id) | FK |
| motherboard\_id | Идентификатор материнской платы | INT NOT NULL REFERENCES motherboards(id) | FK |
| ram\_id | Идентификатор оперативной памяти | INT NOT NULL REFERENCES rams(id) | FK |
| rams\_count | Количество модулей оперативной памяти | INT NOT NULL |  |
| power\_supply\_id | Идентификатор блока питания | INT NOT NULL REFERENCES power\_supplies(id) | FK |
| cooler\_id | Идентификатор охладителя | INT NOT NULL REFERENCES coolers(id) | FK |
| case\_id | Идентификатор корпуса | INT NOT NULL REFERENCES cases(id) | FK |
| image\_url | URL изображения | TEXT |  |
| stock\_quantity | Количество на складе | INT NOT NULL DEFAULT 0 |  |

| **shopping\_cart** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| owner | Владелец корзины (username) | TEXT NOT NULL REFERENCES users(username) | FK |
| computer\_id | Идентификатор компьютера | INT NOT NULL REFERENCES computers(id) | FK |
| quantity | Количество товара в корзине | INT NOT NULL DEFAULT 1 |  |

| **orders** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор заказа | SERIAL | PK |
| customer | Клиент (username) | TEXT NOT NULL REFERENCES users(username) | FK |
| order\_date | Дата заказа | TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP |  |
| total\_amount | Общая сумма заказа | DECIMAL(10,2) NOT NULL |  |
| status | Статус заказа | VARCHAR(25) NOT NULL DEFAULT 'pending' |  |
| comment | Комментарий к заказу | TEXT |  |

| **order\_items** | | | |
| --- | --- | --- | --- |
| **Имя атрибута** | **Значение атрибута** | **Тип данных** | **Ключ (ссылка)** |
| id | Идентификатор элемента заказа | SERIAL | PK |
| order\_id | Идентификатор заказа | INT NOT NULL REFERENCES orders(id) | FK |
| computer\_id | Идентификатор компьютера | INT NOT NULL REFERENCES computers(id) | FK |
| quantity | Количество товара в заказе | INT NOT NULL |  |
| price | Цена товара на момент заказа | DECIMAL(10,2) NOT NULL |  |

## Ограничения ссылочной целостности

1. **Ограничение на таблице «users»:** Идентификатор пользователя (username) является первичным ключом. Это ограничение гарантирует, что каждый пользователь будет иметь уникальное имя пользователя, что позволяет избежать дублирования записей и обеспечивает целостность данных в системе.
2. **Ограничение на таблице «processors»:** Идентификатор процессора (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи о процессоре и позволяет точно идентифицировать каждый процессор в базе данных.
3. **Ограничение на таблице «graphic\_cards»:** Идентификатор видеокарты (id) является первичным ключом. Это ограничение гарантирует, что каждая видеокарта будет иметь уникальный идентификатор, что позволяет точно отслеживать информацию о каждой видеокарте.
4. **Ограничение на таблице «power\_supplies»:** Идентификатор блока питания (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи о блоке питания и позволяет избежать путаницы между различными моделями блоков питания.
5. **Ограничение на таблице «motherboards»:** Идентификатор материнской платы (id) является первичным ключом. Это ограничение гарантирует, что каждая материнская плата будет иметь уникальный идентификатор, что позволяет точно отслеживать информацию о каждой материнской плате.
6. **Ограничение на таблице «coolers»:** Идентификатор охладителя (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи об охладителе и позволяет точно идентифицировать каждый охладитель в базе данных.
7. **Ограничение на таблице «cases»:** Идентификатор корпуса (id) является первичным ключом. Это ограничение гарантирует, что каждый корпус будет иметь уникальный идентификатор, что позволяет избежать дублирования записей и обеспечивает целостность данных.
8. **Ограничение на таблице «rams»:** Идентификатор оперативной памяти (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи об оперативной памяти и позволяет точно отслеживать информацию о каждом модуле RAM.
9. **Ограничение на таблице «computers»:**

Идентификатор процессора (processor\_id) является внешним ключом, связанным с таблицей «processors» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительный процессор, представленный в таблице «processors». Такая связь помогает контролировать совместимость компонентов.

Идентификатор видеокарты (graphic\_card\_id) является внешним ключом, связанным с таблицей «graphic\_cards» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительную видеокарту, представленную в таблице «graphic\_cards».

Идентификатор материнской платы (motherboard\_id) является внешним ключом, связанным с таблицей «motherboards» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительную материнскую плату.

Идентификатор оперативной памяти (ram\_id) является внешним ключом, связанным с таблицей «rams» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительную оперативную память.

Идентификатор блока питания (power\_supply\_id) является внешним ключом, связанным с таблицей «power\_supplies» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительный блок питания.

Идентификатор охладителя (cooler\_id) является внешним ключом, связанным с таблицей «coolers» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительное охлаждение.

Идентификатор корпуса (case\_id) является внешним ключом, связанным с таблицей «cases» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительный корпус.

1. **Ограничение на таблице «shopping\_cart»:** Идентификатор владельца корзины (owner) является внешним ключом, связанным с таблицей «users» по полю идентификатора пользователя (username). Это ограничение гарантирует, что каждая корзина принадлежит действительному пользователю. Идентификатор компьютера (computer\_id) является внешним ключом, связанным с таблицей «computers» по полю идентификатора (id). Это ограничение гарантирует, что каждая запись в корзине содержит действительный компьютер.
2. **Ограничение на таблице «orders»:** Идентификатор клиента (customer) является внешним ключом, связанным с таблицей «users» по полю идентификатора пользователя (username). Это ограничение гарантирует, что каждый заказ принадлежит действительному клиенту.
3. **Ограничение на таблице «order\_items»:** Идентификатор заказа (order\_id) является внешним ключом, связанным с таблицей «orders» по полю идентификатора заказа (id). Это ограничение гарантирует, что каждый элемент заказа относится к действительному заказу. Идентификатор компьютера (computer\_id) является внешним ключом, связанным с таблицей «computers» по полю идентификатора компьютера (id). Это ограничение гарантирует, что каждый элемент заказа содержит действительный компьютер.

## Обоснование выбора СУБД

## Система управления базами данных (СУБД) – это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в ней. СУБД служит посредником между пользователем и БД. В качестве СУБД для моей информационной системы я выбрал PostgreSQL. Это решение обусловлено рядом преимуществ, которые PostgreSQL предлагает по сравнению с другими СУБД:

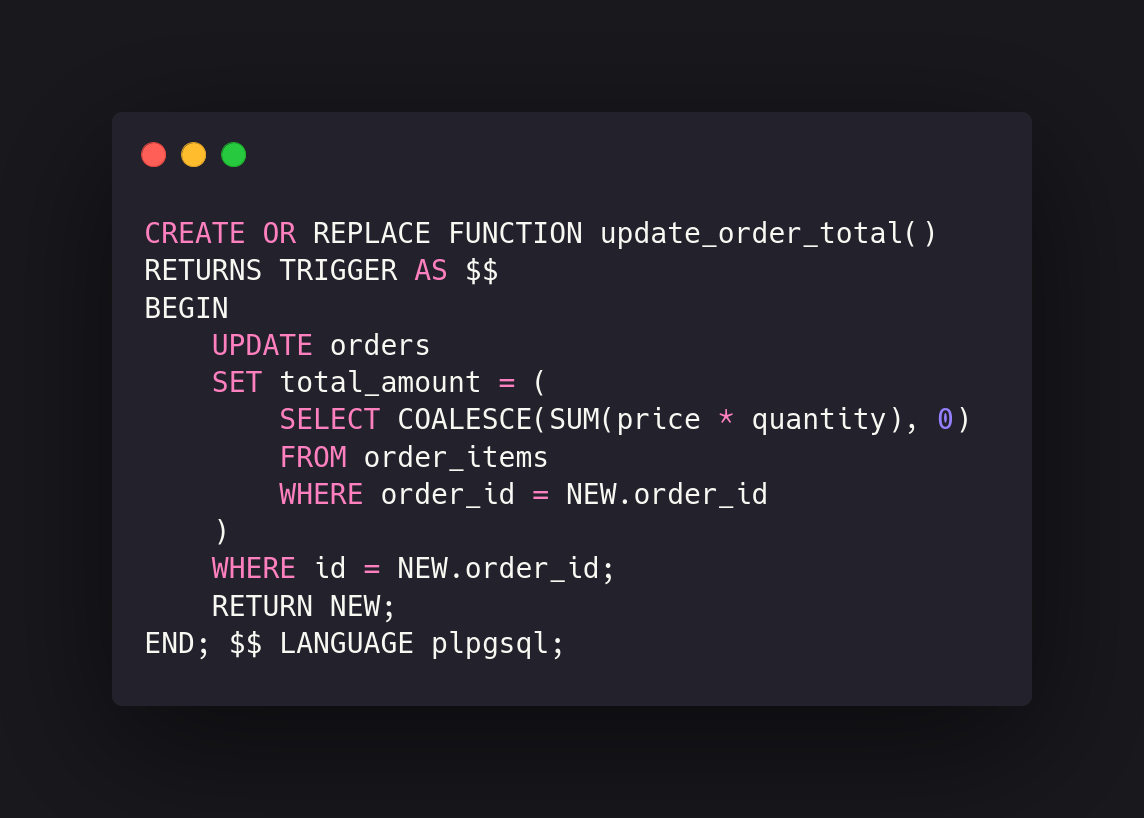
1. **Расширяемость:** PostgreSQL поддерживает пользовательские типы данных, функции и операторы, что позволяет адаптировать систему под специфические требования приложения. Это делает PostgreSQL особенно подходящим для сложных и нестандартных проектов.
2. **Поддержка стандартов SQL:** PostgreSQL строго следует стандартам SQL, что обеспечивает совместимость с другими системами и упрощает миграцию данных. Это также облегчает обучение пользователей, знакомых с SQL.
3. **Высокая производительность:** PostgreSQL использует эффективные механизмы индексации и оптимизации запросов, что обеспечивает высокую производительность при работе с большими объемами данных и сложными запросами.
4. **Надежность и целостность данных:** PostgreSQL поддерживает ACID-транзакции (атомарность, согласованность, изолированность, долговечность), что гарантирует надежность и целостность данных. Это особенно важно для критически важных приложений.
5. **Многоуровневая безопасность:** PostgreSQL предлагает расширенные функции безопасности, включая аутентификацию на уровне базы данных, шифрование данных и управление правами доступа на уровне строк и столбцов. Это обеспечивает защиту данных от несанкционированного доступа.
6. **Активное сообщество и поддержка:** PostgreSQL имеет большое и активное сообщество разработчиков и пользователей, что обеспечивает доступ к обширной документации, руководствам и онлайн-ресурсам. Пользователи могут легко получать поддержку от опытных специалистов.
7. **Частые обновления и новые функции:** PostgreSQL активно разрабатывается и регулярно получает обновления, которые включают новые функции, исправления ошибок и улучшения производительности. Это гарантирует, что система будет оставаться актуальной и безопасной.
8. **Личный опыт использования:** Я постоянно использую PostgreSQL в своих проектах, так как он мне хорошо знаком. Его удобный интерфейс и многофункциональность позволяют эффективно решать задачи различной сложности. Например, поддержка массивов и JSON-данных предоставляет возможности для работы с более сложными структурами данных, которые отсутствуют в других СУБД. Это делает его идеальным выбором для разработки современных приложений.

## Триггеры и хранимые процедуры

## Триггеры и хранимые процедуры для базы данных магазина компьютерной техники могут быть созданы, чтобы автоматизировать определенные операции или проверки при выполнении изменений в базе данных.

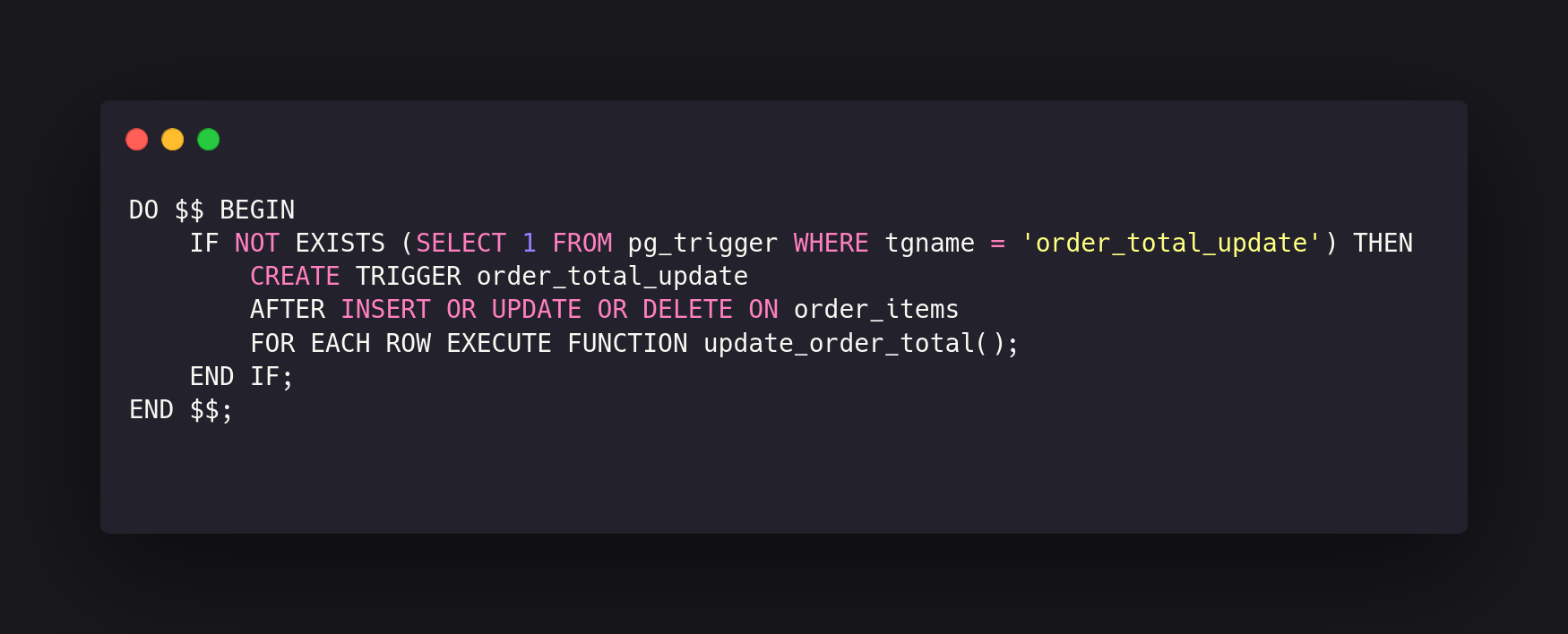
## Хранимая функция для обновления общей суммы заказа

## Хранимая функция **update\_order\_total** предназначена для автоматического обновления общей суммы заказа в таблице **«orders»**. Она вычисляет новую общую сумму заказа, суммируя стоимость всех элементов заказа из таблицы **«order\_items»** и обновляет соответствующую запись в таблице **«orders»**. Функция будет вызываться триггером после вставки, обновления или удаления элементов заказа.



## Триггер для обновления общей суммы заказа

## Триггер **order\_total\_update** будет активироваться после каждой вставки, обновления или удаления записи в таблице **«order\_items»**. Он будет вызывать функцию **update\_order\_total**, чтобы обеспечить автоматическое обновление общей суммы заказа при любых изменениях в связанных элементах заказа, что позволяет поддерживать целостность данных и актуальность информации о заказах.



# Разработка информационной системы

## Описание структуры системы

## В рамках данного проекта была разработана информационная система, представляющая собой приложение, предназначенное для функционирования в качестве магазина компьютерной техники. Данная система состоит из восьми взаимосвязанных страниц, каждая из которых выполняет свою уникальную функцию и обеспечивает пользователю доступ к различным аспектам работы с приложением. Основные страницы приложения включают:

## Регистрация – страница, на которой пользователи могут создать новый аккаунт, заполнив необходимые поля с личной информацией.

## Авторизация – интерфейс для входа в систему, где пользователи вводят свои учетные данные для доступа к функционалу приложения.

## Каталог товаров – основная страница, на которой представлены все доступные товары, сгруппированные по категориям.

## Информация о товаре – страница, предоставляющая детальную информацию о выбранном товаре, включая его характеристики и цену.

## Фильтры – инструмент для сортировки и фильтрации товаров по различным критериям, таким как цена, производитель и технические характеристики.

## Корзина товаров – страница, на которой пользователи могут просмотреть добавленные в корзину товары перед оформлением заказа.

## Заказы – интерфейс для просмотра истории заказов пользователя и их статусов.

## Админ панель – специализированный интерфейс для администраторов, позволяющий управлять данными в системе.

## Административная панель включает в себя дополнительные одиннадцать страниц, каждая из которых соответствует отдельной таблице базы данных. Эти таблицы содержат информацию о различных аспектах работы магазина и включают:

## **users** – таблица с данными пользователей приложения.

## **processors** – информация о процессорах, доступных в магазине.

## **graphic\_cards** – данные о видеокартах.

## **motherboards** – сведения о материнских платах.

## **power\_supplies** – информация о блоках питания.

## **rams** – данные об оперативной памяти.

## **coolers** – информация о системах охлаждения.

## **cases** – сведения о корпусах компьютеров.

## **computers** – таблица с информацией о готовых сборках компьютеров.

## **orders** – данные о заказах пользователей.

## **order\_items** – информация о товарах в каждом заказе.

## Переходы между страницами приложения реализованы с помощью кнопок и контекстного меню, что обеспечивает удобство навигации для пользователей.

## При запуске приложения пользователь сразу попадает на страницу авторизации. Здесь он может пройти процедуру идентификации, введя свои учетные данные. В случае отсутствия аккаунта имеется возможность перейти к регистрации нового пользователя.

## После успешного ввода данных и нажатия на кнопку **"Login"**, пользователь автоматически перенаправляется на основной интерфейс приложения, где ему становится доступен полный функционал системы.

## 

## На главной странице пользователя встречает меню каталога товаров. В этом меню представлено обычное перечисление компьютеров с карточками товаров, включающими фотографии, названия и описания.

## Каждая карточка содержит информацию о цене и кнопку **"Add to cart"**, нажатие на которую позволяет добавить выбранный компьютер в корзину. Если пользователь нажимает на область с названием или описанием товара, он попадает на страницу просмотра информации о продукте. На данной странице отображается вся информация из карточки товара с добавлением детальных характеристик компьютера, таких как процессор, видеокарта, материнская плата, блок питания, оперативная память, кулер и корпус.

## Страница фильтров предоставляет пользователю возможность отфильтровать каталог товаров по необходимым комплектующим или интересующим ценовым диапазонам.

## Страница **"Cart" (корзина товаров)** аналогична каталогу товаров, но с дополнительными функциями. Здесь пользователь видит количество товара на складе и виджет для изменения количества выбранного компьютера в заказе.

## Также присутствует кнопка для удаления товара из корзины. В верхней части списка отображается итоговая стоимость заказа и кнопка **"Make an order" (сделать заказ)**.

## Если пользователь имеет роль администратора, то данная страница дополнительно включает текстовое поле для указания контактов пользователя, который сделал данный заказ. Это может быть полезно в случаях, когда клиент связывается с магазином по телефону и предоставляет информацию для оформления заказа.

## После завершения оформления заказа пользователь перенаправляется на страницу **"Orders"(заказы)**, где он может ознакомиться с подробной информацией обо всех совершенных заказах. Доступная информация включает:

## дату оформления заказа

## итоговую цену заказа

## статус выполнения заказа

## краткие сведения о компьютерах, приобретенных в конкретном заказе (цены указаны на момент покупки).

## Если у пользователя приложения установлен статус администратора, он получает доступ к кнопке "admin panel", которая открывает контекстное меню с выбором таблицы для редактирования. После выбора соответст2вующей таблицы открывается страница, на которой отображается таблица с данными во весь экран. В верхней части страницы расположены несколько кнопок управления таблицей:

## **Add row** – добавление новой строки в таблицу,

## **Delete row** – удаление выделенной строки,

## **Save changes** – сохранение всех внесенных изменений в таблице,

## **Refresh** – перезагрузка страницы с обновлением актуальных данных из базы данных.

## Разработка класса – точки входа

Листинг класса Main (главный класс, с которого начинается работа программы)

```

package com.shop;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

import javafx.stage.StageStyle;

import java.io.IOException;

import com.shop.database.DbConnection;

/\*\*

\* JavaFX App

\*/

public class App extends Application {

private static Scene scene;

@Override

public void start(@SuppressWarnings("exports") Stage stage) throws IOException {

DbConnection dbConnection = DbConnection.getDatabaseConnection();

dbConnection.createTablesIfNotExists();

scene = new Scene(loadFXML("auth/LoginView"), 640, 480);

stage.setScene(scene);

stage.initStyle(StageStyle.UNDECORATED);

stage.show();

stage.setAlwaysOnTop(true);

}

static void setRoot(String fxml) throws IOException {

scene.setRoot(loadFXML(fxml));

}

private static Parent loadFXML(String fxml) throws IOException {

FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));

return fxmlLoader.load();

}

public static void main(String[] args) {

launch();

}

}

```

## Хранение общей информации

Листинг класса SharedData (класс для хранения промежуточной информации, использующейся разными сценами приложения)

```

package com.shop.controllers;

import com.shop.database.models.Case;

import com.shop.database.models.Computer;

import com.shop.database.models.Cooler;

import com.shop.database.models.GraphicCard;

import com.shop.database.models.Motherboard;

import com.shop.database.models.PowerSupply;

import com.shop.database.models.Processor;

import com.shop.database.models.RAM;

import com.shop.database.models.User;

public class SharedData {

public static User authenticatedUser;

public static MainPanelController controller;

public static Computer selectedComputer;

public static Integer minCostFilter = 1;

public static Integer maxCostFilter = 1000000000;

public static Processor processorFilter;

public static GraphicCard graphicCardFilter;

public static Motherboard motherboardFilter;

public static PowerSupply powerSupplyFilter;

public static RAM ramFilter;

public static Integer countRAMFilter = 0;

public static Cooler coolerFilter;

public static Case caseFilter;

public static void setAuthenticatedUser(User value) {

authenticatedUser = value;

}

public static User getAuthenticatedUser() {

return authenticatedUser;

}

public static void setMainController(MainPanelController value) {

controller = value;

}

public static MainPanelController getMainController() {

return controller;

}

public static void setSelectedComputer(Computer value) {

selectedComputer = value;

}

public static Computer getSelectedComputer() {

return selectedComputer;

}

public static Integer getMinCostFilter() {

return minCostFilter;

}

public static void setMinCostFilter(Integer minCost) {

minCostFilter = minCost;

}

public static Integer getMaxCostFilter() {

return maxCostFilter;

}

public static void setMaxCostFilter(Integer maxCost) {

maxCostFilter = maxCost;

}

public static Processor getProcessorFilter() {

return processorFilter;

}

public static void setProcessorFilter(Processor processor) {

processorFilter = processor;

}

public static GraphicCard getGraphicCardFilter() {

return graphicCardFilter;

}

public static void setGraphicCardFilter(GraphicCard gpu) {

graphicCardFilter = gpu;

}

public static Motherboard getMotherboardFilter() {

return motherboardFilter;

}

public static void setMotherboardFilter(Motherboard motherboard) {

motherboardFilter = motherboard;

}

public static PowerSupply getPowerSupplyFilter() {

return powerSupplyFilter;

}

public static void setPowerSupplyFilter(PowerSupply ps) {

powerSupplyFilter = ps;

}

public static RAM getRamFilter() {

return ramFilter;

}

public static void setRamFilter(RAM ram) {

ramFilter = ram;

}

public static Integer getCountRAMFilter() {

return countRAMFilter;

}

public static void setCountRAMFilter(Integer countRAM) {

countRAMFilter = countRAM;

}

public static Cooler getCoolerFilter() {

return coolerFilter;

}

public static void setCoolerFilter(Cooler cooler) {

coolerFilter = cooler;

}

public static Case getCaseFilter() {

return caseFilter;

}

public static void setCaseFilter(Case value) {

caseFilter = value;

}

public static void resetFilters() {

processorFilter = null;

graphicCardFilter = null;

motherboardFilter = null;

powerSupplyFilter = null;

ramFilter = null;

countRAMFilter = 0;

coolerFilter = null;

caseFilter = null;

minCostFilter = 1;

maxCostFilter = 1000000000;

}

}

```

## Модели

Листинги классов, олицетворяющих колонки определенных таблиц. (нужны для представления полученных данных в виде объектов, для удобного взаимодействия приложения с данными)

```

package com.shop.database.models;

public interface Identifiable {

Integer getId();

String toString();

}

```

```

package com.shop.database.models;

import java.sql.Timestamp;

public class User {

private String username;

private String new\_username;

private String email;

private String firstName;

private String lastName;

private String password;

private String role;

private Timestamp createdAt;

public User(String username, String email, String firstName, String lastName, String password, String role) {

// Конструктор без createdAt

this.username = username;

this.new\_username = username;

this.email = email;

this.firstName = firstName;

this.lastName = lastName;

this.password = password;

this.role = role;

}

public User(String username, String email, String firstName, String lastName, String password, String role, Timestamp createdAt) {

this.username = username;

this.new\_username = username;

this.email = email;

this.firstName = firstName;

this.lastName = lastName;

this.password = password;

this.role = role;

this.createdAt = createdAt;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.new\_username = username;

}

public String getNewUsername() {

return new\_username;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

public Timestamp getCreatedAt() {

return createdAt;

}

public void setCreatedAt(Timestamp createdAt) {

this.createdAt = createdAt;

}

}

```

```

package com.shop.database.models;

import java.math.BigDecimal;

public class Processor implements Identifiable {

private Integer id;

private String name;

private String brand;

private Integer cores;

private Integer threads;

private BigDecimal baseClock; // Using BigDecimal for decimal values

private BigDecimal boostClock; // Using BigDecimal for decimal values

private String link;

public Processor(Integer id, String name, String brand, Integer cores, Integer threads, BigDecimal baseClock, BigDecimal boostClock, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.cores = cores;

this.threads = threads;

this.baseClock = baseClock;

this.boostClock = boostClock;

this.link = link;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public Integer getCores() {

return cores;

}

public void setCores(Integer cores) {

this.cores = cores;

}

public Integer getThreads() {

return threads;

}

public void setThreads(Integer threads) {

this.threads = threads;

}

public BigDecimal getBaseClock() {

return baseClock;

}

public void setBaseClock(BigDecimal baseClock) {

this.baseClock = baseClock;

}

public BigDecimal getBoostClock() {

return boostClock;

}

public void setBoostClock(BigDecimal boostClock) {

this.boostClock = boostClock;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

public class GraphicCard implements Identifiable {

private Integer id;

private String name;

private String brand;

private Integer memorySize;

private String memoryType;

private String link;

public GraphicCard(Integer id, String name, String brand, Integer memorySize, String memoryType, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.memorySize = memorySize;

this.memoryType = memoryType;

this.link = link;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public Integer getMemorySize() {

return memorySize;

}

public void setMemorySize(Integer memorySize) {

this.memorySize = memorySize;

}

public String getMemoryType() {

return memoryType;

}

public void setMemoryType(String memoryType) {

this.memoryType = memoryType;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

public class Motherboard implements Identifiable {

private Integer id;

private String name;

private String brand;

private String socketType;

private String formFactor;

private Integer maxMemory;

private String link;

public Motherboard(Integer id, String name, String brand, String socketType, String formFactor, Integer maxMemory, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.socketType = socketType;

this.formFactor = formFactor;

this.maxMemory = maxMemory;

this.link = link;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public String getSocketType() {

return socketType;

}

public void setSocketType(String socketType) {

this.socketType = socketType;

}

public String getFormFactor() {

return formFactor;

}

public void setFormFactor(String formFactor) {

this.formFactor = formFactor;

}

public Integer getMaxMemory() {

return maxMemory;

}

public void setMaxMemory(Integer maxMemory) {

this.maxMemory = maxMemory;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

public class PowerSupply implements Identifiable {

private Integer id;

private String name;

private String brand;

private Integer wattage;

private String efficiencyRating;

private String link;

public PowerSupply(Integer id, String name, String brand, Integer wattage, String efficiencyRating, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.wattage = wattage;

this.efficiencyRating = efficiencyRating;

this.link = link;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public Integer getWattage() {

return wattage;

}

public void setWattage(Integer wattage) {

this.wattage = wattage;

}

public String getEfficiencyRating() {

return efficiencyRating;

}

public void setEfficiencyRating(String efficiencyRating) {

this.efficiencyRating = efficiencyRating;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

public class RAM implements Identifiable {

private Integer id;

private String name;

private String brand;

private Integer capacity; // in GB

private Integer speed; // in MHz

private String link;

public RAM(Integer id, String name, String brand, Integer capacity, Integer speed, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.capacity = capacity;

this.speed = speed;

this.link = link;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public Integer getCapacity() {

return capacity;

}

public void setCapacity(Integer capacity) {

this.capacity = capacity;

}

public Integer getSpeed() {

return speed;

}

public void setSpeed(Integer speed) {

this.speed = speed;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

import java.math.BigDecimal;

public class Cooler implements Identifiable {

private Integer id;

private String name;

private String brand;

private String type;

private BigDecimal coolingCapacity; // Using BigDecimal for precision

private String link;

public Cooler(Integer id, String name, String brand, String type, BigDecimal coolingCapacity, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.type = type;

this.coolingCapacity = coolingCapacity;

this.link = link;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public BigDecimal getCoolingCapacity() {

return coolingCapacity;

}

public void setCoolingCapacity(BigDecimal coolingCapacity) {

this.coolingCapacity = coolingCapacity;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

public class Case implements Identifiable {

private Integer id;

private String name;

private String brand;

private String formFactor;

private String color;

private String link;

public Case(Integer id, String name, String brand, String formFactor, String color, String link) {

this.id = id;

this.name = name;

this.brand = brand;

this.formFactor = formFactor;

this.color = color;

this.link = link;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public String getFormFactor() {

return formFactor;

}

public void setFormFactor(String formFactor) {

this.formFactor = formFactor;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getLink() {

return link;

}

public void setLink(String link) {

this.link = link;

}

@Override

public String toString() {

return name;

}

}

```

```

package com.shop.database.models;

public class ShoppingCartItem {

private String owner;

private Integer computerId;

private Integer quantity;

public ShoppingCartItem(String owner, Integer computerId, Integer quantity) {

this.owner = owner;

this.computerId = computerId;

this.quantity = quantity;

}

public String getOwner() {

return owner;

}

public void setOwner(String owner) {

this.owner = owner;

}

public Integer getComputerId() {

return computerId;

}

public void setComputerId(Integer computerId) {

this.computerId = computerId;

}

public Integer getQuantity() {

return quantity;

}

public void setQuantity(Integer quantity) {

this.quantity = quantity;

}

}

```

```

package com.shop.database.models;

import java.math.BigDecimal;

import java.sql.Timestamp;

import java.util.List;

public class Order {

private Integer id;

private String customer;

private Timestamp orderDate;

private BigDecimal totalAmount;

private String comment;

private String status;

private List<OrderItem> items;

public Order(Integer id, String customer, Timestamp orderDate, BigDecimal totalAmount, String status, String comment, List<OrderItem> items) {

this.id = id;

this.customer = customer;

this.orderDate = orderDate;

this.totalAmount = totalAmount;

this.status = status;

this.items = items;

this.comment = comment;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getCustomer() {

return customer;

}

public void setCustomer(String customer) {

this.customer = customer;

}

public Timestamp getOrderDate() {

return orderDate;

}

public void setOrderDate(Timestamp orderDate) {

this.orderDate = orderDate;

}

public BigDecimal getTotalAmount() {

return totalAmount;

}

public void setTotalAmount(BigDecimal totalAmount) {

this.totalAmount = totalAmount;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public void setItems(List<OrderItem> itms) {

this.items = itms;

}

public List<OrderItem> getItems() {

return items;

}

public void setComment(String value) {

this.comment = value;

}

public String getComment() {

return comment;

}

}

```

```

package com.shop.database.models;

public class OrderItem {

private Integer id;

private Integer orderId;

private Computer computer;

private Integer quantity;

public OrderItem(Integer id, Integer orderId, Computer pc, Integer quantity) {

this.id = id;

this.orderId = orderId;

this.computer = pc;

this.quantity = quantity;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getOrderId() {

return orderId;

}

public void setOrderId(Integer orderId) {

this.orderId = orderId;

}

public Computer getComputer() {

return computer;

}

public void setComputer(Computer pc) {

this.computer = pc;

}

public Integer getQuantity() {

return quantity;

}

public void setQuantity(Integer quantity) {

this.quantity = quantity;

}

}

```

```

package com.shop.database.models;

public class OrderResult {

private boolean success;

private String message;

public OrderResult(boolean success, String message) {

this.success = success;

this.message = message;

}

public boolean isSuccess() {

return success;

}

public String getMessage() {

return message;

}

}

```

## Разработка класса подключения к базе данных

Листинг класса DbConnection (класс для взаимодействия с базой данных)

## ```

## package com.shop.database;

## import java.io.FileInputStream;

## import java.math.BigDecimal;

## import java.security.Timestamp;

## import java.sql.Connection;

## import java.sql.DriverManager;

## import java.sql.PreparedStatement;

## import java.sql.ResultSet;

## import java.sql.SQLException;

## import java.sql.Statement;

## import java.util.ArrayList;

## import java.util.List;

## import java.util.Map;

## import java.util.Properties;

## import java.util.logging.Level;

## import java.util.logging.Logger;

## import com.shop.database.models.User;

## import com.shop.helper.AlertHelper;

## import com.shop.database.models.Processor;

## import com.shop.database.models.GraphicCard;

## import com.shop.database.models.Motherboard;

## import com.shop.database.models.Order;

## import com.shop.database.models.OrderItem;

## import com.shop.database.models.OrderResult;

## import com.shop.database.models.PowerSupply;

## import com.shop.database.models.RAM;

## import com.shop.database.models.ShoppingCartItem;

## import com.shop.database.models.Cooler;

## import com.shop.database.models.Case;

## import com.shop.database.models.Computer;

## public class DbConnection {

## private Connection con;

## private static DbConnection dbc;

## public void createTablesIfNotExists() {

## String createUsersTable = "CREATE TABLE IF NOT EXISTS users (" +

## "username TEXT PRIMARY KEY NOT NULL, " +

## "email TEXT NOT NULL, " +

## "first\_name TEXT NOT NULL, " +

## "last\_name TEXT NOT NULL, " +

## "password TEXT NOT NULL, " +

## "role VARCHAR(25) NOT NULL DEFAULT 'user', " +

## "created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP" +

## ");";

## String createProcessorsTable = "CREATE TABLE IF NOT EXISTS processors (" +

## "id SERIAL PRIMARY KEY, " +

## "name TEXT NOT NULL, " +

## "brand TEXT NOT NULL, " +

## "cores INT NOT NULL, " +

## "threads INT NOT NULL, " +

## "base\_clock DECIMAL(5,2) NOT NULL, " +

## "boost\_clock DECIMAL(5,2) NOT NULL, " +

## "link TEXT NOT NULL" +

## ");";

## String createGraphicsCardsTable = "CREATE TABLE IF NOT EXISTS graphic\_cards (" +

## "id SERIAL PRIMARY KEY, " +

## "name TEXT NOT NULL, " +

## "brand TEXT NOT NULL, " +

## "memory\_size INT NOT NULL, " +

## "memory\_type TEXT NOT NULL, " +

## "link TEXT NOT NULL" +

## ");";

## String createPowerSuppliesTable = "CREATE TABLE IF NOT EXISTS power\_supplies (" +

## "id SERIAL PRIMARY KEY, " +

## "name TEXT NOT NULL, " +

## "brand TEXT NOT NULL, " +

## "wattage INT NOT NULL, " +

## "efficiency\_rating TEXT NOT NULL, " +

## "link TEXT NOT NULL" +

## ");";

## String createMotherboardsTable = "CREATE TABLE IF NOT EXISTS motherboards (" +

## "id SERIAL PRIMARY KEY, " +

## "name TEXT NOT NULL, " +

## "brand TEXT NOT NULL, " +

## "socket\_type TEXT NOT NULL, " +

## "form\_factor TEXT NOT NULL, " +

## "max\_memory INT NOT NULL, " +

## "link TEXT NOT NULL" +

## ");";

## String createCoolersTable = "CREATE TABLE IF NOT EXISTS coolers (" +

## "id SERIAL PRIMARY KEY, "+

## "name TEXT NOT NULL," +

## "brand TEXT NOT NULL," +

## "type TEXT NOT NULL," +

## "cooling\_capacity DECIMAL(5,2) NOT NULL," +

## "link TEXT NOT NULL" +

## ");";

## String createCasesTable = "CREATE TABLE IF NOT EXISTS cases (" +

## "id SERIAL PRIMARY KEY," +

## "name TEXT NOT NULL," +

## "brand TEXT NOT NULL," +

## "form\_factor TEXT NOT NULL," +

## "color TEXT," +

## "link TEXT NOT NULL" +

## ");";

## String createRAMTable = "CREATE TABLE IF NOT EXISTS rams (" +

## "id SERIAL PRIMARY KEY," +

## "name TEXT NOT NULL," +

## "brand TEXT NOT NULL," +

## "capacity INT NOT NULL," + // Объем памяти в ГБ

## "speed INT NOT NULL," + // Скорость в МГц

## "link TEXT NOT NULL" +

## ");";

## 

## String createComputersTable = "CREATE TABLE IF NOT EXISTS computers (" +

## "id SERIAL PRIMARY KEY, " +

## "name TEXT NOT NULL, " +

## "description TEXT NOT NULL, " +

## "price DECIMAL(10,2) NOT NULL, " +

## "processor\_id INT NOT NULL REFERENCES processors(id), " +

## "graphic\_card\_id INT NOT NULL REFERENCES graphic\_cards(id), " +

## "motherboard\_id INT NOT NULL REFERENCES motherboards(id), " +

## "ram\_id INT NOT NULL REFERENCES rams(id), " +

## "rams\_count INT NOT NULL, " +

## "power\_supply\_id INT NOT NULL REFERENCES power\_supplies(id), " +

## "cooler\_id INT NOT NULL REFERENCES coolers(id), " +

## "case\_id INT NOT NULL REFERENCES cases(id), " +

## "image\_url TEXT NOT NULL, " +

## "stock\_quantity INT NOT NULL DEFAULT 0" +

## ");";

## String createShoppingCartTable = "CREATE TABLE IF NOT EXISTS shopping\_cart (" +

## "owner TEXT NOT NULL REFERENCES users(username), " +

## "computer\_id INT NOT NULL REFERENCES computers(id), " +

## "quantity INT NOT NULL DEFAULT 1" +

## ");";

## String createOrdersTable = "CREATE TABLE IF NOT EXISTS orders (" +

## "id SERIAL PRIMARY KEY, " +

## "customer TEXT NOT NULL REFERENCES users(username), " +

## "order\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP, " +

## "total\_amount DECIMAL(10,2) NOT NULL, " +

## "status VARCHAR(25) NOT NULL DEFAULT 'pending', " +

## "comment TEXT" +

## ");";

## String createOrderItemsTable = "CREATE TABLE IF NOT EXISTS order\_items (" +

## "id SERIAL PRIMARY KEY, " +

## "order\_id INT NOT NULL REFERENCES orders(id), " +

## "computer\_id INT NOT NULL REFERENCES computers(id), " +

## "quantity INT NOT NULL, " +

## "price DECIMAL(10,2) NOT NULL" +

## ");";

## String createOrderFunction = "CREATE OR REPLACE FUNCTION update\_order\_total() " +

## "RETURNS TRIGGER AS $$ " +

## "BEGIN " +

## " UPDATE orders " +

## " SET total\_amount = ( " +

## " SELECT COALESCE(SUM(price \* quantity), 0) " +

## " FROM order\_items " +

## " WHERE order\_id = NEW.order\_id " +

## " ) " +

## " WHERE id = NEW.order\_id; " +

## " RETURN NEW; " +

## "END; $$ LANGUAGE plpgsql;";

## String createOrderTrigger = "DO $$ BEGIN " +

## "IF NOT EXISTS (SELECT 1 FROM pg\_trigger WHERE tgname = 'order\_total\_update') THEN " +

## "CREATE TRIGGER order\_total\_update " +

## "AFTER INSERT OR UPDATE OR DELETE ON order\_items " +

## "FOR EACH ROW EXECUTE FUNCTION update\_order\_total(); " +

## "END IF; END $$;";

## 

## try (Statement statement = con.createStatement()) {

## statement.executeUpdate(createUsersTable);

## statement.executeUpdate(createProcessorsTable);

## statement.executeUpdate(createGraphicsCardsTable);

## statement.executeUpdate(createPowerSuppliesTable);

## statement.executeUpdate(createMotherboardsTable);

## statement.executeUpdate(createCoolersTable);

## statement.executeUpdate(createCasesTable);

## statement.executeUpdate(createRAMTable);

## statement.executeUpdate(createComputersTable);

## statement.executeUpdate(createShoppingCartTable);

## statement.executeUpdate(createOrdersTable);

## statement.executeUpdate(createOrderItemsTable);

## statement.executeUpdate(createOrderFunction);

## statement.executeUpdate(createOrderTrigger);

## } catch (Exception ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## }

## private DbConnection() {

## try {

## DriverManager.registerDriver(new org.postgresql.Driver());

## FileInputStream fis = new FileInputStream("connection.prop");

## Properties p = new Properties();

## p.load(fis);

## con = DriverManager.getConnection((String) p.get("url"), (String) p.get("username"), (String) p.get("password"));

## } catch (Exception ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## }

## 

## public static DbConnection getDatabaseConnection() {

## if (dbc == null) {

## dbc = new DbConnection();

## }

## return dbc;

## }

## public Connection getConnection() {

## return con;

## }

## //==> USER CRUD

## // ==================================================================================================================================

## public List<User> getAllUsers() {

## List<User> users = new ArrayList<>();

## String query = "SELECT \* FROM users";

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## User user = new User(

## rs.getString("username"),

## rs.getString("email"),

## rs.getString("first\_name"),

## rs.getString("last\_name"),

## rs.getString("password"),

## rs.getString("role"),

## rs.getTimestamp("created\_at")

## );

## users.add(user);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return users;

## }

## public User getUserByUsername(String username) {

## String query = "SELECT \* FROM users WHERE username = ?";

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setString(1, username);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## User user = new User(

## rs.getString("username"),

## rs.getString("email"),

## rs.getString("first\_name"),

## rs.getString("last\_name"),

## rs.getString("password"),

## rs.getString("role"),

## rs.getTimestamp("created\_at")

## );

## return user;

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteUser(String username) {

## String deleteUser = "DELETE FROM users WHERE username = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteUser)) {

## pstmt.setString(1, username);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateUser(User user) {

## String updateUser = "UPDATE users SET email = ?, first\_name = ?, last\_name = ?, password = ?, role = ? WHERE username = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateUser)) {

## pstmt.setString(1, user.getEmail());

## pstmt.setString(2, user.getFirstName());

## pstmt.setString(3, user.getLastName());

## pstmt.setString(4, user.getPassword());

## pstmt.setString(5, user.getRole());

## pstmt.setString(6, user.getUsername());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean addUser(User user) {

## String insertUser = "INSERT INTO users (first\_name, last\_name, email, username, password, role) VALUES (?, ?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertUser)) {

## pstmt.setString(1, user.getFirstName());

## pstmt.setString(2, user.getLastName());

## pstmt.setString(3, user.getEmail());

## pstmt.setString(4, user.getUsername());

## pstmt.setString(5, user.getPassword());

## pstmt.setString(6, user.getRole());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean isUsernameExists(String username) {

## String query = "SELECT \* FROM users WHERE username = ?";

## try (PreparedStatement ps = con.prepareStatement(query)) {

## ps.setString(1, username);

## try (ResultSet rs = ps.executeQuery()) {

## return rs.next();

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean authenticateUser(String username, String password) {

## String query = "SELECT \* FROM users WHERE username = ? AND password = ?";

## try (PreparedStatement ps = con.prepareStatement(query)) {

## ps.setString(1, username);

## ps.setString(2, password);

## try (ResultSet rs = ps.executeQuery()) {

## return rs.next();

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## //==> PROCESSOR CRUD

## // ==================================================================================================================================

## public List<Processor> getAllProcessors() {

## List<Processor> processors = new ArrayList<>();

## String query = "SELECT \* FROM processors";

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## Processor processor = new Processor(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("cores"),

## rs.getInt("threads"),

## rs.getBigDecimal("base\_clock"),

## rs.getBigDecimal("boost\_clock"),

## rs.getString("link")

## );

## processors.add(processor);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return processors;

## }

## public Processor getProcessorById(int id) {

## String query = "SELECT \* FROM processors WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new Processor(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("cores"),

## rs.getInt("threads"),

## rs.getBigDecimal("base\_clock"),

## rs.getBigDecimal("boost\_clock"),

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteProcessor(Integer id) {

## String deleteProcessor = "DELETE FROM processors WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteProcessor)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addProcessor(Processor processor) {

## String insertProcessor = "INSERT INTO processors (name, brand, cores, threads, base\_clock, boost\_clock, link) VALUES (?, ?, ?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertProcessor)) {

## pstmt.setString(1, processor.getName());

## pstmt.setString(2, processor.getBrand());

## pstmt.setInt(3, processor.getCores());

## pstmt.setInt(4, processor.getThreads());

## pstmt.setBigDecimal(5, processor.getBaseClock());

## pstmt.setBigDecimal(6, processor.getBoostClock());

## pstmt.setString(7, processor.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateProcessor(Processor processor) {

## String updateProcessor = "UPDATE processors SET name = ?, brand = ?, cores = ?, threads = ?, base\_clock = ?, boost\_clock = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateProcessor)) {

## pstmt.setString(1, processor.getName());

## pstmt.setString(2, processor.getBrand());

## pstmt.setInt(3, processor.getCores());

## pstmt.setInt(4, processor.getThreads());

## pstmt.setBigDecimal(5, processor.getBaseClock());

## pstmt.setBigDecimal(6, processor.getBoostClock());

## pstmt.setString(7, processor.getLink());

## pstmt.setInt(8, processor.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## 

## //==> GRAPHICSCARD CRUD

## // ==================================================================================================================================

## public List<GraphicCard> getAllGraphicCards() {

## List<GraphicCard> graphicsCards = new ArrayList<>();

## String query = "SELECT \* FROM graphic\_cards";

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## GraphicCard graphicsCard = new GraphicCard(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("memory\_size"),

## rs.getString("memory\_type"),

## rs.getString("link")

## );

## graphicsCards.add(graphicsCard);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return graphicsCards;

## }

## public GraphicCard getGraphicCardById(int id) {

## String query = "SELECT \* FROM graphic\_cards WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new GraphicCard(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("memory\_size"),

## rs.getString("memory\_type"),

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteGraphicCard(Integer id) {

## String deleteGraphicsCard = "DELETE FROM graphic\_cards WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteGraphicsCard)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addGraphicCard(GraphicCard graphicsCard) {

## String insertGraphicsCard = "INSERT INTO graphic\_cards (name, brand, memory\_size, memory\_type, link) VALUES (?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertGraphicsCard)) {

## pstmt.setString(1, graphicsCard.getName());

## pstmt.setString(2, graphicsCard.getBrand());

## pstmt.setInt(3, graphicsCard.getMemorySize());

## pstmt.setString(4, graphicsCard.getMemoryType());

## pstmt.setString(5, graphicsCard.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateGraphicCard(GraphicCard graphicsCard) {

## String updateGraphicsCard = "UPDATE graphic\_cards SET name = ?, brand = ?, memory\_size = ?, memory\_type = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateGraphicsCard)) {

## pstmt.setString(1, graphicsCard.getName());

## pstmt.setString(2, graphicsCard.getBrand());

## pstmt.setInt(3, graphicsCard.getMemorySize());

## pstmt.setString(4, graphicsCard.getMemoryType());

## pstmt.setString(5, graphicsCard.getLink());

## pstmt.setInt(6, graphicsCard.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null ,ex);

## return false;

## }

## }

## 

## 

## //==> POWERSUPPLY CRUD

## // ==================================================================================================================================

## public List<PowerSupply> getAllPowerSupplies() {

## List<PowerSupply> powerSupplies = new ArrayList<>();

## String query = "SELECT \* FROM power\_supplies";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## PowerSupply powerSupply = new PowerSupply(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("wattage"),

## rs.getString("efficiency\_rating"),

## rs.getString("link")

## );

## powerSupplies.add(powerSupply);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return powerSupplies;

## }

## public PowerSupply getPowerSupplyById(int id) {

## String query = "SELECT \* FROM power\_supplies WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new PowerSupply(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("wattage"),

## rs.getString("efficiency\_rating"),

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deletePowerSupply(Integer id) {

## String deletePowerSupply = "DELETE FROM power\_supplies WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deletePowerSupply)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addPowerSupply(PowerSupply powerSupply) {

## String insertPowerSupply = "INSERT INTO power\_supplies (name, brand, wattage, efficiency\_rating, link) VALUES (?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertPowerSupply)) {

## pstmt.setString(1, powerSupply.getName());

## pstmt.setString(2, powerSupply.getBrand());

## pstmt.setInt(3, powerSupply.getWattage());

## pstmt.setString(4, powerSupply.getEfficiencyRating());

## pstmt.setString(5, powerSupply.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updatePowerSupply(PowerSupply powerSupply) {

## String updatePowerSupply = "UPDATE power\_supplies SET name = ?, brand = ?, wattage = ?, efficiency\_rating = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updatePowerSupply)) {

## pstmt.setString(1, powerSupply.getName());

## pstmt.setString(2, powerSupply.getBrand());

## pstmt.setInt(3, powerSupply.getWattage());

## pstmt.setString(4, powerSupply.getEfficiencyRating());

## pstmt.setString(5, powerSupply.getLink());

## pstmt.setInt(6, powerSupply.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## //==> RAM CRUD

## // ==================================================================================================================================

## public List<RAM> getAllRAMs() {

## List<RAM> rams = new ArrayList<>();

## String query = "SELECT \* FROM rams";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## RAM ram = new RAM(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("capacity"), // in GB

## rs.getInt("speed"), // in MHz

## rs.getString("link")

## );

## rams.add(ram);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return rams;

## }

## public RAM getRAMById(int id) {

## String query = "SELECT \* FROM rams WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new RAM(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getInt("capacity"), // in GB

## rs.getInt("speed"), // in MHz

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteRAM(Integer id) {

## String deleteRAM = "DELETE FROM rams WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteRAM)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addRAM(RAM ram) {

## String insertRAM = "INSERT INTO rams (name, brand, capacity, speed, link) VALUES (?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertRAM)) {

## pstmt.setString(1, ram.getName());

## pstmt.setString(2, ram.getBrand());

## pstmt.setInt(3, ram.getCapacity());

## pstmt.setInt(4, ram.getSpeed());

## pstmt.setString(5, ram.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateRAM(RAM ram) {

## String updateRAM = "UPDATE rams SET name = ?, brand = ?, capacity = ?, speed = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateRAM)) {

## pstmt.setString(1, ram.getName());

## pstmt.setString(2, ram.getBrand());

## pstmt.setInt(3, ram.getCapacity());

## pstmt.setInt(4, ram.getSpeed());

## pstmt.setString(5, ram.getLink());

## pstmt.setInt(6, ram.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## //==> MOTHERBOARD CRUD

## // ==================================================================================================================================

## public List<Motherboard> getAllMotherboards() {

## List<Motherboard> motherboards = new ArrayList<>();

## String query = "SELECT \* FROM motherboards";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## Motherboard motherboard = new Motherboard(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getString("socket\_type"),

## rs.getString("form\_factor"),

## rs.getInt("max\_memory"),

## rs.getString("link")

## );

## motherboards.add(motherboard);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return motherboards;

## }

## public Motherboard getMotherboardById(int id) {

## String query = "SELECT \* FROM motherboards WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new Motherboard(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getString("socket\_type"),

## rs.getString("form\_factor"),

## rs.getInt("max\_memory"),

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteMotherboard(Integer id) {

## String deleteMotherboard = "DELETE FROM motherboards WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteMotherboard)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addMotherboard(Motherboard motherboard) {

## String insertMotherboard = "INSERT INTO motherboards (name, brand, socket\_type, form\_factor, max\_memory, link) VALUES (?, ?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertMotherboard)) {

## pstmt.setString(1, motherboard.getName());

## pstmt.setString(2, motherboard.getBrand());

## pstmt.setString(3, motherboard.getSocketType());

## pstmt.setString(4, motherboard.getFormFactor());

## pstmt.setInt(5, motherboard.getMaxMemory());

## pstmt.setString(6, motherboard.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateMotherboard(Motherboard motherboard) {

## String updateMotherboard = "UPDATE motherboards SET name = ?, brand = ?, socket\_type = ?, form\_factor = ?, max\_memory = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateMotherboard)) {

## pstmt.setString(1, motherboard.getName());

## pstmt.setString(2, motherboard.getBrand());

## pstmt.setString(3, motherboard.getSocketType());

## pstmt.setString(4, motherboard.getFormFactor());

## pstmt.setInt(5, motherboard.getMaxMemory());

## pstmt.setString(6,motherboard .getLink());

## pstmt .setInt (7,motherboard .getId());

## return pstmt.executeUpdate() > 0;

## } catch(SQLException ex){

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## //==> COOLER CRUD

## // ==================================================================================================================================

## public List<Cooler> getAllCoolers() {

## List<Cooler> coolers = new ArrayList<>();

## String query = "SELECT \* FROM coolers";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## Cooler cooler = new Cooler(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getString("type"),

## rs.getBigDecimal("cooling\_capacity"),

## rs.getString("link")

## );

## coolers.add(cooler);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return coolers;

## }

## public Cooler getCoolerById(int id) {

## String query = "SELECT \* FROM coolers WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new Cooler(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getString("type"),

## rs.getBigDecimal("cooling\_capacity"),

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteCooler(Integer id) {

## String deleteCooler = "DELETE FROM coolers WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteCooler)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addCooler(Cooler cooler) {

## String insertCooler = "INSERT INTO coolers (name, brand, type, cooling\_capacity, link) VALUES (?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertCooler)) {

## pstmt.setString(1, cooler.getName());

## pstmt.setString(2, cooler.getBrand());

## pstmt.setString(3, cooler.getType());

## pstmt.setBigDecimal(4, cooler.getCoolingCapacity());

## pstmt.setString(5, cooler.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateCooler(Cooler cooler) {

## String updateCooler = "UPDATE coolers SET name = ?, brand = ?, type = ?, cooling\_capacity = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateCooler)) {

## pstmt.setString(1, cooler.getName());

## pstmt.setString(2, cooler.getBrand());

## pstmt.setString(3, cooler.getType());

## pstmt.setBigDecimal(4, cooler.getCoolingCapacity());

## pstmt.setString(5, cooler.getLink());

## pstmt.setInt(6, cooler.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## //==> CASE CRUD

## // ==================================================================================================================================

## public List<Case> getAllCases() {

## List<Case> cases = new ArrayList<>();

## String query = "SELECT \* FROM cases";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## Case computerCase = new Case(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getString("form\_factor"),

## rs.getString("color"),

## rs.getString("link")

## );

## cases.add(computerCase);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return cases;

## }

## public Case getCaseById(int id) {

## String query = "SELECT \* FROM cases WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new Case(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("brand"),

## rs.getString("form\_factor"),

## rs.getString("color"),

## rs.getString("link")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## 

## public boolean deleteCase(Integer id) {

## String deleteCase = "DELETE FROM cases WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteCase)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean addCase(Case computerCase) {

## String insertCase = "INSERT INTO cases (name, brand, form\_factor, color, link) VALUES (?, ?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(insertCase)) {

## pstmt.setString(1, computerCase.getName());

## pstmt.setString(2, computerCase.getBrand());

## pstmt.setString(3, computerCase.getFormFactor());

## pstmt.setString(4, computerCase.getColor());

## pstmt.setString(5, computerCase.getLink());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateCase(Case computerCase) {

## String updateCase = "UPDATE cases SET name = ?, brand = ?, form\_factor = ?, color = ?, link = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateCase)) {

## pstmt.setString(1, computerCase.getName());

## pstmt.setString(2, computerCase.getBrand());

## pstmt.setString(3, computerCase.getFormFactor());

## pstmt.setString(4, computerCase.getColor());

## pstmt.setString(5 ,computerCase .getLink());

## pstmt.setInt (6 ,computerCase .getId());

## return pstmt.executeUpdate()>0;

## } catch(SQLException ex){

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## //==> COMPUTER CRUD

## // ==================================================================================================================================

## public List<Computer> getAllComputers() {

## List<Computer> computers = new ArrayList<>();

## String query = "SELECT \* FROM computers";

## try (PreparedStatement pstmt = con.prepareStatement(query);

## ResultSet rs = pstmt.executeQuery()) {

## while (rs.next()) {

## Computer computer = new Computer(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("description"),

## rs.getBigDecimal("price"),

## rs.getInt("processor\_id"),

## rs.getInt("graphic\_card\_id"),

## rs.getInt("motherboard\_id"),

## rs.getInt("ram\_id"),

## rs.getInt("rams\_count"),

## rs.getInt("power\_supply\_id"),

## rs.getInt("cooler\_id"),

## rs.getInt("case\_id"),

## rs.getString("image\_url"),

## rs.getInt("stock\_quantity")

## );

## computers.add(computer);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return computers;

## }

## public Computer getComputerById(Integer id) {

## String query = "SELECT \* FROM computers WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, id);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new Computer(

## rs.getInt("id"),

## rs.getString("name"),

## rs.getString("description"),

## rs.getBigDecimal("price"),

## rs.getInt("processor\_id"),

## rs.getInt("graphic\_card\_id"),

## rs.getInt("motherboard\_id"),

## rs.getInt("ram\_id"),

## rs.getInt("rams\_count"),

## rs.getInt("power\_supply\_id"),

## rs.getInt("cooler\_id"),

## rs.getInt("case\_id"),

## rs.getString("image\_url"),

## rs.getInt("stock\_quantity")

## );

## } else {

## return null;

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return null;

## }

## }

## public boolean addComputer(Computer computer) {

## String insertComputerSQL = "INSERT INTO computers (name, description, price, processor\_id, graphic\_card\_id, " +

## "motherboard\_id, ram\_id, rams\_count, power\_supply\_id, cooler\_id, case\_id, image\_url, stock\_quantity) " +

## "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

## 

## try (PreparedStatement pstmt = con.prepareStatement(insertComputerSQL)) {

## pstmt.setString(1, computer.getName());

## pstmt.setString(2, computer.getDescription());

## pstmt.setBigDecimal(3, computer.getPrice());

## pstmt.setInt(4, computer.getProcessorId());

## pstmt.setInt(5, computer.getGraphicCardId());

## pstmt.setInt(6, computer.getMotherboardId());

## pstmt.setInt(7, computer.getRamId());

## pstmt.setInt(8, computer.getRamsCount());

## pstmt.setInt(9, computer.getPowerSupplyId());

## pstmt.setInt(10, computer.getCoolerId());

## pstmt.setInt(11, computer.getCaseId());

## pstmt.setString(12, computer.getImageUrl());

## pstmt.setInt(13, computer.getStockQuantity());

## 

## return pstmt.executeUpdate() > 0;

## 

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## public boolean updateComputer(Computer computer) {

## String updateComputer = "UPDATE computers SET name = ?, description = ?, price = ?, processor\_id = ?, graphic\_card\_id = ?, motherboard\_id = ?, ram\_id = ?, rams\_count = ?, power\_supply\_id = ?, cooler\_id = ?, case\_id = ?, image\_url = ?, stock\_quantity = ? WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(updateComputer)) {

## pstmt.setString(1, computer.getName());

## pstmt.setString(2, computer.getDescription());

## pstmt.setBigDecimal(3, computer.getPrice());

## pstmt.setInt(4, computer.getProcessorId());

## pstmt.setInt(5, computer.getGraphicCardId());

## pstmt.setInt(6, computer.getMotherboardId());

## pstmt.setInt(7, computer.getRamId());

## pstmt.setInt(8, computer.getRamsCount());

## pstmt.setInt(9, computer.getPowerSupplyId());

## pstmt.setInt(10, computer.getCoolerId());

## pstmt.setInt(11, computer.getCaseId());

## pstmt.setString(12, computer.getImageUrl());

## pstmt.setInt(13, computer.getStockQuantity());

## pstmt.setInt(14, computer.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## private void updateStockQuantities(Map<Integer, Integer> computerIdsAndQuantities) throws SQLException {

## String updateQuery = "UPDATE computers SET stock\_quantity = stock\_quantity - ? WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {

## for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {

## int computerId = entry.getKey();

## int quantityToReduce = entry.getValue();

## 

## pstmt.setInt(1, quantityToReduce);

## pstmt.setInt(2, computerId);

## pstmt.addBatch();

## }

## pstmt.executeBatch();

## }

## }

## private String checkStockAvailability(Map<Integer, Integer> computerIdsAndQuantities) {

## for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {

## int computerId = entry.getKey();

## int quantityRequested = entry.getValue();

## 

## String stockCheckQuery = "SELECT stock\_quantity, name FROM computers WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(stockCheckQuery)) {

## pstmt.setInt(1, computerId);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## int stockQuantity = rs.getInt("stock\_quantity");

## String computerName = rs.getString("name");

## if (stockQuantity < quantityRequested) {

## return "Out of stock: " + computerName; // недостаточно на складе

## }

## } else {

## return "Computer with ID " + computerId + " not found."; // Компьютер не найден

## }

## } catch (SQLException e) {

## e.printStackTrace();

## return "Error during warehouse verification: " + e.getMessage();

## }

## }

## return null; // Все товары доступны

## }

## public boolean deleteComputer(Integer id) {

## String deleteComputer = "DELETE FROM computers WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(deleteComputer)) {

## pstmt.setInt(1, id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## //==> SHOPPING CART CRUD

## // ==================================================================================================================================

## public List<ShoppingCartItem> getShoppingCartItemsByOwner(String owner) {

## String query = "SELECT \* FROM shopping\_cart WHERE owner = ?";

## List<ShoppingCartItem> items = new ArrayList<>();

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setString(1, owner);

## ResultSet rs = pstmt.executeQuery();

## while (rs.next()) {

## ShoppingCartItem item = new ShoppingCartItem(

## rs.getString("owner"),

## rs.getInt("computer\_id"),

## rs.getInt("quantity")

## );

## items.add(item);

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return items;

## }

## public ShoppingCartItem getShoppingCartItemById(String owner, Integer computer\_id) {

## String query = "SELECT \* FROM shopping\_cart WHERE computer\_id = ? AND owner = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(query)) {

## pstmt.setInt(1, computer\_id);

## pstmt.setString(2, owner);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## return new ShoppingCartItem(

## rs.getString("owner"),

## rs.getInt("computer\_id"),

## rs.getInt("quantity")

## );

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## return null;

## }

## public boolean addShoppingCartItem(ShoppingCartItem item) {

## String insertComputerSQL = "INSERT INTO shopping\_cart (owner, computer\_id, quantity) VALUES (?, ?, ?)";

## 

## try (PreparedStatement pstmt = con.prepareStatement(insertComputerSQL)) {

## pstmt.setString(1, item.getOwner());

## pstmt.setInt(2, item.getComputerId());

## pstmt.setInt(3, item.getQuantity());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean updateShoppingCartItem(ShoppingCartItem item) {

## String updateQuery = "UPDATE shopping\_cart SET quantity = ? WHERE computer\_id = ? AND owner = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {

## pstmt.setInt(1, item.getQuantity());

## pstmt.setInt(2, item.getComputerId());

## pstmt.setString(3, item.getOwner());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean deleteShoppingCartItem(ShoppingCartItem item) {

## String deleteQuery = "DELETE FROM shopping\_cart WHERE computer\_id = ? AND owner = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {

## pstmt.setInt(1, item.getComputerId());

## pstmt.setString(2, item.getOwner());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean clearShoppingCart(String username) {

## String deleteQuery = "DELETE FROM shopping\_cart WHERE owner = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {

## pstmt.setString(1, username);

## int rowsAffected = pstmt.executeUpdate();

## return rowsAffected > 0; // Возвращает true, если были удалены записи

## } catch (SQLException e) {

## e.printStackTrace();

## AlertHelper.showErrorAlert("Ошибка при очистке корзины: " + e.getMessage());

## return false;

## }

## }

## 

## 

## //==> ORDERS CRUD

## // ==================================================================================================================================

## public List<Order> getUserOrders(String username) {

## List<Order> orders = new ArrayList<>();

## 

## String orderQuery = "SELECT \* FROM orders WHERE customer = ?";

## 

## try (PreparedStatement orderStmt = con.prepareStatement(orderQuery)) {

## orderStmt.setString(1, username);

## ResultSet orderRs = orderStmt.executeQuery();

## 

## while (orderRs.next()) {

## Integer orderId = orderRs.getInt("id");

## List<OrderItem> orderItems = getOrderItems(orderId);

## orders.add(new Order(

## orderId,

## orderRs.getString("customer"),

## orderRs.getTimestamp("order\_date"),

## new BigDecimal(orderRs.getDouble("total\_amount")),

## orderRs.getString("status"),

## orderRs.getString("comment"),

## orderItems

## ));

## }

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## 

## return orders;

## }

## public OrderResult createOrder(String customerUsername, String comment, Map<Integer, Integer> computerIdsAndQuantities) {

## String checkStatus = checkStockAvailability(computerIdsAndQuantities);

## if (checkStatus != null) {

## return new OrderResult(false, checkStatus);

## }

## 

## try {

## con.setAutoCommit(false); // Отключаем автоматическое подтверждение

## String orderInsertQuery = "INSERT INTO orders (customer, comment, total\_amount) VALUES (?, ?, ?) RETURNING id";

## PreparedStatement orderStmt = con.prepareStatement(orderInsertQuery);

## orderStmt.setString(1, customerUsername);

## orderStmt.setString(2, comment);

## orderStmt.setDouble(3, calculateTotalAmount(computerIdsAndQuantities));

## 

## ResultSet generatedKeys = orderStmt.executeQuery();

## if (generatedKeys.next()) {

## int orderId = generatedKeys.getInt(1);

## createOrderItems(orderId, computerIdsAndQuantities);

## updateStockQuantities(computerIdsAndQuantities);

## con.commit(); // Подтверждаем транзакцию

## return new OrderResult(true, "The order has been successfully created!");

## }

## } catch (SQLException e) {

## try {

## con.rollback(); // Откат транзакции в случае ошибки

## } catch (SQLException rollbackEx) {

## rollbackEx.printStackTrace();

## return new OrderResult(false, "Error on transaction rollback: " + rollbackEx.getMessage());

## }

## e.printStackTrace();

## return new OrderResult(false, "Error when creating an order");

## } finally {

## try {

## con.setAutoCommit(true); // Включаем автоматическое подтверждение обратно

## } catch (SQLException e) {

## e.printStackTrace();

## }

## }

## return new OrderResult(false, "Failed to create an order. Try again later");

## }

## private double calculateTotalAmount(Map<Integer, Integer> computerIdsAndQuantities) throws SQLException {

## double totalAmount = 0.0;

## for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {

## int computerId = entry.getKey();

## int quantityRequested = entry.getValue();

## String priceQuery = "SELECT price FROM computers WHERE id = ?";

## try (PreparedStatement pstmt = con.prepareStatement(priceQuery)) {

## pstmt.setInt(1, computerId);

## ResultSet rs = pstmt.executeQuery();

## if (rs.next()) {

## double price = rs.getDouble("price");

## totalAmount += price \* quantityRequested;

## }

## }

## }

## return totalAmount;

## }

## public boolean updateOrder(Order order) {

## String updateQuery = "UPDATE orders SET status = ?, comment = ? WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {

## pstmt.setString(1, order.getStatus());

## pstmt.setString(2, order.getComment());

## pstmt.setInt(3, order.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean deleteOrder(Integer order\_id) {

## String deleteQuery = "DELETE FROM orders WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {

## pstmt.setInt(1, order\_id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## 

## 

## 

## //==> ORDER ITEMS CRUD

## // ==================================================================================================================================

## private List<OrderItem> getOrderItems(Integer orderId) throws SQLException {

## List<OrderItem> orderItems = new ArrayList<>();

## 

## String itemQuery = "SELECT \* FROM order\_items WHERE order\_id = ?";

## 

## try (PreparedStatement itemStmt = con.prepareStatement(itemQuery)) {

## itemStmt.setInt(1, orderId);

## ResultSet itemRs = itemStmt.executeQuery();

## 

## while (itemRs.next()) {

## Integer itemId = itemRs.getInt("id");

## Integer computerId = itemRs.getInt("computer\_id");

## Integer quantity = itemRs.getInt("quantity");

## BigDecimal price = new BigDecimal(itemRs.getDouble("price"));

## 

## Computer pc = getComputerById(computerId);

## if (pc!= null) {

## pc.setPrice(price);

## orderItems.add(new OrderItem(

## itemId,

## orderId,

## pc,

## quantity

## ));

## }

## }

## }

## 

## return orderItems;

## }

## public List<OrderItem> getAllOrderItems() {

## List<OrderItem> orderItems = new ArrayList<>();

## 

## String itemQuery = "SELECT \* FROM order\_items";

## 

## try (PreparedStatement itemStmt = con.prepareStatement(itemQuery)) {;

## ResultSet itemRs = itemStmt.executeQuery();

## 

## while (itemRs.next()) {

## Integer orderId = itemRs.getInt("order\_id");

## Integer itemId = itemRs.getInt("id");

## Integer computerId = itemRs.getInt("computer\_id");

## Integer quantity = itemRs.getInt("quantity");

## BigDecimal price = new BigDecimal(itemRs.getDouble("price"));

## 

## Computer pc = getComputerById(computerId);

## if (pc!= null) {

## pc.setPrice(price.multiply(new BigDecimal(quantity)));

## orderItems.add(new OrderItem(

## itemId,

## orderId,

## pc,

## quantity

## ));

## }

## }

## } catch (Exception ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## }

## 

## return orderItems;

## }

## public boolean updateOrderItem(OrderItem item) {

## String updateQuery = "UPDATE order\_items SET order\_id = ?, computer\_id = ?, quantity = ? WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {

## pstmt.setInt(1, item.getOrderId());

## pstmt.setInt(2, item.getComputer().getId());

## pstmt.setInt(3, item.getQuantity());

## pstmt.setInt(4, item.getId());

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## public boolean deleteOrderItem(Integer item\_id) {

## String deleteQuery = "DELETE FROM order\_items WHERE id = ?";

## 

## try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {

## pstmt.setInt(1, item\_id);

## return pstmt.executeUpdate() > 0;

## } catch (SQLException ex) {

## Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

## AlertHelper.showErrorAlert("Unknown Error. Try again");

## return false;

## }

## }

## private void createOrderItems(int orderId, Map<Integer, Integer> computerIdsAndQuantities) throws SQLException {

## String orderItemInsertQuery = "INSERT INTO order\_items (order\_id, computer\_id, quantity, price) VALUES (?, ?, ?, ?)";

## try (PreparedStatement pstmt = con.prepareStatement(orderItemInsertQuery)) {

## for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {

## int computerId = entry.getKey();

## int quantityRequested = entry.getValue();

## 

## Computer computer = getComputerById(computerId);

## if (computer != null) {

## pstmt.setInt(1, orderId);

## pstmt.setInt(2, computerId);

## pstmt.setInt(3, quantityRequested);

## pstmt.setBigDecimal(4, computer.getPrice());

## pstmt.addBatch();

## } else {

## throw new SQLException("Computer with ID " + computerId + " not found.");

## }

## }

## pstmt.executeBatch();

## }

## }

## }

## ```