МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Основная профессиональная образовательная программа
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль) «Технологии разработки программного обеспечения»
форма обучения – очная

ОТЧЁТ

по реализации проекта для дисциплины «Базы данных»
по направлению “09.03.01 – Информатика и вычислительная техника”
(профиль: “Технологии разработки программного обеспечения ”)

Преподаватель: к.ф-м.н., доцент кафедры ИТиЭО

_____
(Жуков Н. Н.)

Студенты 2 курса:

Васильева М. А. _____

Балаев Ж. Б. _____

Иванов Н. Р. _____

Шардт М. А. _____

Санкт-Петербург

2023

Оглавление

Ответственные

Васильева М. А. – разработчик проекта. Нормализация базы данных и создание процедур и тестовых данных.

Балаев Ж. Б. – разработчик проекта. Создание тестовых данных, создание индексов, сравнительный анализ NoSql.

Иванов Н. Р. – разработчик проекта. Нормализация, создание триггеров и функций, создание тестовых данных.

Шардт М. А. – разработчик проекта. Нормализация, разработка EF-модели, создание тестовых данных и тестирование, организация кооперативной работы.

Предметная область

Предметная область: онлайн-сервис для заказа и доставки кофе и десертов из кофеен.

Сервис предоставляет возможность пользователям оформлять заказы и просматривать их историю. Заказы могут быть созданы из определенной точки продажи и содержать товары, выбранные из меню. Также предусмотрено два вида получения заказов: в ресторане или доставкой на указанный пользователем адрес.

Каждая точка продажи имеет свое расписание работы, а также может содержать различные товары и иметь различное наличие товаров. Управление товарами в меню каждой точки продажи предусмотрено добавление, изменение или удаление товаров, а также установка их стоимости и описания.

Для работы в конкретной точке продажи могут быть назначены сотрудники. У каждого сотрудника есть определенное расписание работы, позиция заработная плата. Менеджеры точек могут изменять график работы сотрудников, их заработную плату, а также повышать или понижать сотрудников.

Ход выполнения нормализации

Начальные отношения

Для реализации предметной области необходимо хранить в базе данных следующую информацию:

Информация о пользователе включает в себя его имя и телефон, также у каждого пользователя должна быть история заказов и информация о доставке.

Для точек продажи необходимо адрес, график работы, список сотрудников, меню товаров, а также их наличие.

О сотрудниках нужно хранит их имя и фамилию, точку, в которой они работают, а также расписание, заработную плату, телефон и электронную почту.

Каждый товар представляет собой название, описание, цену и изображение.

Заказ же формируется для одной точки и одного клиента, и он состоит из товаров, их общей стоимости и способа доставки. Для доставки на адрес необходим адрес доставки. Также храниться статус заказа:

- ➢ Принят
- ➢ Готовится
- ➢ Готов
- ➢ Выдан
- ➢ Завершен
- ➢ Отменен

А для доставки:

➢ Принят ➢ В Пути ➢ Доставлен ➢ Отменен

Нормализованные отношения

1. Сущность "Точка продажи" (Stores) была разделена на отдельные сущности "Адрес" (Addresses) и "Меню" (Menu). В самой таблице содержится:

    - Первичный ключ (Id) INT
    - Время открытия (OpeningTime) TIME
    - Время закрытия (ClosingTime) TIME
    - Внешний ключ на адрес (AddressId) INT

2. "Адрес" содержит:

    - Первичный ключ (Id) INT
    - Город (Town) VARCHAR(255)
    - Улица (Street) VARCHAR(255)
    - Номер дома (House) VARCHAR(63)
    - Парадная (Entrance) VARCHAR(63)
    - Номер квартиры (Flat) VARCHAR(63)

Для всех полей использован текстовый тип, так как они все могут содержать в себе буквенные значения.

3. "Меню" (Menu) является составной таблицей для реализации связи многие-ко-многим между Точкой Продажи (Stores) и Продуктом (Products). Тем самым оно содержит только товары, доступные в конкретной точке продажи, а не все товары, которые могут быть доступны во всех точках продажи. Это также позволяет легко добавлять или удалять товары из меню каждой точки.

4. Сущность "Продукт" (Products) представляет собой независимую единицу товара, которая содержит следующие атрибуты:

- Первичный ключ (Id) INT
- Название (Title) VARCHAR(255)
- Описание (Description) TEXT
- Цена (Price) DECIMAL(10, 2)
- Изображение (ImageUri) VARCHAR(255)

Таблица продуктов содержит информацию о каждом товаре, которая может использоваться другими таблицами для создания связей и отношений между продуктами и другими сущностями в системе.

5. Сущность "Заказ" (Orders) представляет собой таблицу, которая содержит информацию о точке продажи (Stores) и пользователе (Customers), создавшем заказ, а также о заказанных товарах. Для связи между заказом и товарами используется отношение "многие ко многим", которое реализуется с помощью создания таблицы "Заказ-Товар" (OrderProduct). Также в сущности хранится информация о его статусе (Status).
    - Первичный ключ (Id) INT
    - Внешний ключ на точку продажи (StoreId) INT
    - Внешний ключ на пользователя (CustomerId) INT
    - Статус (Status) VARCHAR(63)
6. "Доставка" содержит информацию о доставке, включая номер заказа, адрес доставки, статус доставки и время доставки. Номер заказа "Id" уникален, "OrderId" и "AddressId" – внешние ключи. "Status" содержит текущий статус доставки, а "DeliveryTime" отображает время, когда доставка была выполнена.
    - Номер заказа (Id) INT
    - Внешний ключ на (OrderId) INT

заказ

- Внешний ключ на адрес (AddressId) INT
- Статус (Status) VARCHAR(63)
- Время доставки (DeliveryTime) TIMESTAMP

7. "Пользователи" содержит информацию о зарегистрированных пользователях. "Phone" и "Email" являются уникальными полями, которые гарантируют уникальность каждого пользователя в системе. "Name" содержит имя пользователя.
   - Первичный ключ (Id) INT
   - Номер телефона (Phone) VARCHAR(255)
   - Email (Email) VARCHAR(255)
   - Имя (Name) VARCHAR(255)
8. "Сотрудники" содержит информацию о сотрудниках, включая "FirstName" и "LastName" содержащие имя и фамилию сотрудника, "Email" и "Phone" являются уникальными полями,
   - Первичный ключ (Id) INT
   - Имя (FirstName) VARCHAR(255)
   - Фамилия (LastName) VARCHAR(255)
   - Email (Email) VARCHAR(255)
   - Номер телефона (Phone) VARCHAR(255)
9. Таблице позиции (JobTitle) состоит из "Title" содержит должность сотрудника, а "Salary" отображает размер заработной платы. "EmployeeId " – внешний ключ на Сотрудника."StoreId" – внешний ключ на Точку продажи; он может быть NULL, так как сотрудник может быть независим от Точки продажи.
   - Позиция (Title) VARCHAR(63)

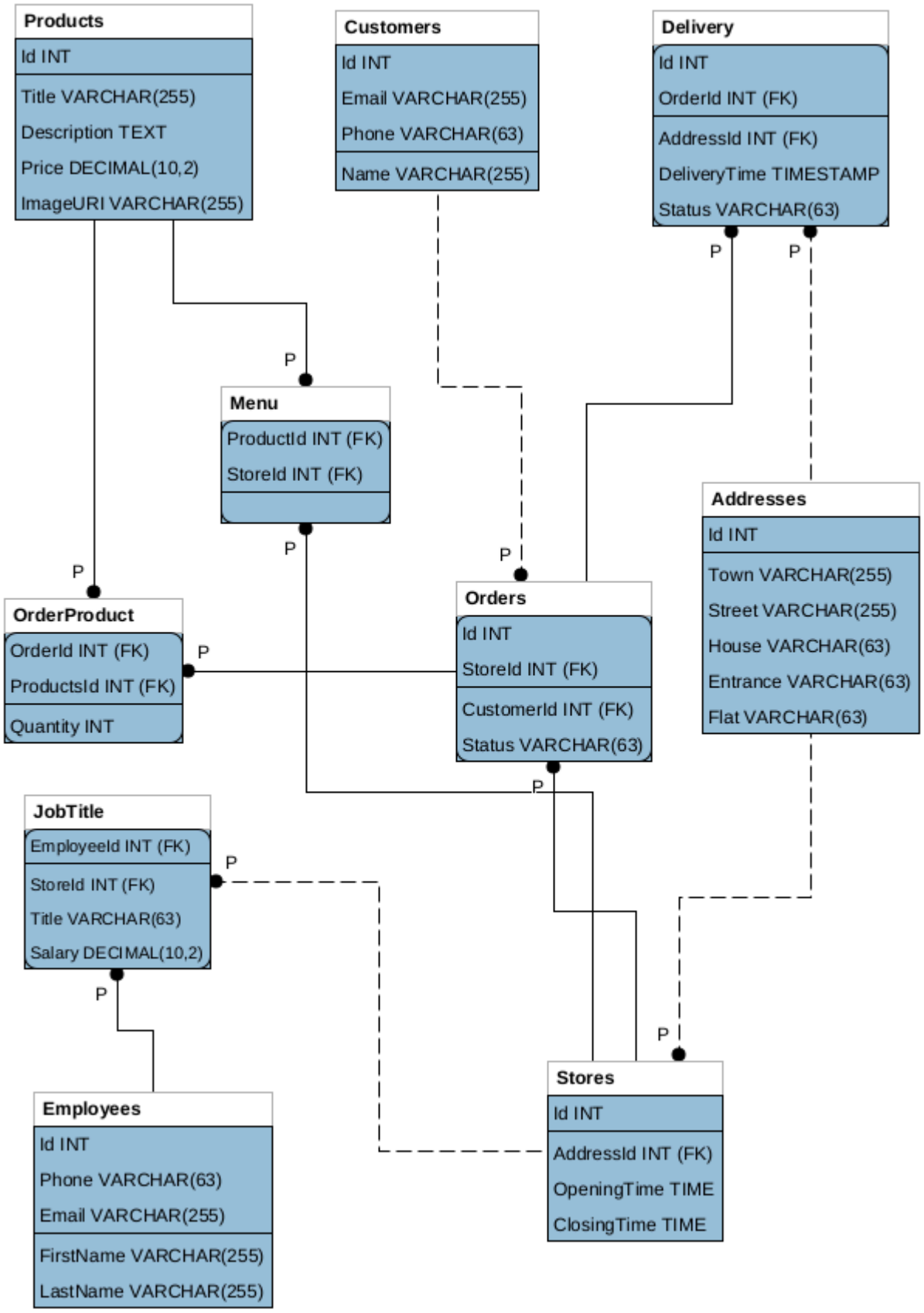
- Заработная плата (Salary) DECIMAL(10, 2)
- Внешний ключ на сотрудника (EmployeeId) INT
- Внешний ключ на точку продажи (StoreId) INT

Объяснение выбранной СУБД

Наш выбор - MySQL. В сравнении с другими СУБД мы выделили несколько преимуществ:

1. **Знакомство команды с технологией.** Использование реляционного MySQL также обусловлено тем, что мы уже знакомы с его основами и изучали его на дисциплине “Базы данных”.
2. **Простота использования.** Синтаксис запросов MySQL является интуитивно понятным как и для начинающих, так и для опытных пользователей. Он также предлагает широкий набор инструментов администрирования и графических интерфейсов, что упрощает работу с базами данных.
3. **Распространенность и совместимость.** MySQL является одной из самых популярных СУБД в мире. Также MySQL поддерживает различные операционные системы, включая Windows и Linux.
4. **Бесплатность.** MySQL является бесплатной и открытой системой управления базами данных (СУБД). Это делает его доступным для широкого круга пользователей и организаций, особенно для небольших и средних проектов без бюджета.

Entity–Relationship диаграмма



Исходный текст запросов

По созданию таблиц и индексов

Создание таблицы “Адреса”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Addresses` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `Town` VARCHAR(255) NOT NULL,
  `Street` VARCHAR(255) NOT NULL,
  `House` VARCHAR(63) NOT NULL,
  `Entrance` VARCHAR(63) NULL DEFAULT NULL,
  `Flat` VARCHAR(63) NULL DEFAULT NULL,
  PRIMARY KEY (`Id`))
ENGINE = InnoDB
AUTO_INCREMENT = 0
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Создание таблицы “Покупатели”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Customers` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `Email` VARCHAR(63) NOT NULL,
  `Phone` VARCHAR(63) NOT NULL,
  `Name` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`Id`, `Email`, `Phone`))
ENGINE = InnoDB
AUTO_INCREMENT = 0
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Создание таблицы “Точка продаж” + индекс

```
CREATE TABLE IF NOT EXISTS `coffee`.`Stores` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `AddressId` INT NOT NULL,
  `OpeningTime` TIME NULL DEFAULT NULL,
  `ClosingTime` TIME NULL DEFAULT NULL,
  PRIMARY KEY (`Id`),
  INDEX `IX_Stores_AddressId` USING BTREE (`AddressId`) VISIBLE,
  CONSTRAINT `FK_Stores_Addresses_AddressId`
    FOREIGN KEY (`AddressId`)
    REFERENCES `coffee`.`Addresses` (`Id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 0
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Создание таблицы “Заказы”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Orders` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `StoreId` INT NOT NULL,
  `CustomerId` INT NOT NULL,
  `Status` VARCHAR(63) NOT NULL,
  PRIMARY KEY (`Id`, `StoreId`),
  INDEX `IX_Orders_CustomerId` USING BTREE (`CustomerId`) VISIBLE,
  INDEX `IX_Orders_StoreId` USING BTREE (`StoreId`) VISIBLE,
  CONSTRAINT `FK_Orders_Customers_CustomerId`
    FOREIGN KEY (`CustomerId`)
    REFERENCES `coffee`.`Customers` (`Id`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT,
  CONSTRAINT `FK_Orders_Stores_StoreId`
    FOREIGN KEY (`StoreId`)
    REFERENCES `coffee`.`Stores` (`Id`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB
AUTO_INCREMENT = 0
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Создание таблицы “Доставка”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Delivery` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `OrderId` INT NOT NULL,
  `AddressId` INT NOT NULL,
  `DeliveryTime` TIMESTAMP NOT NULL,
  `Status` VARCHAR(63) NOT NULL,
  PRIMARY KEY (`Id`, `OrderId`),
  UNIQUE INDEX `OrderId_UNIQUE` (`OrderId` ASC) VISIBLE,
  INDEX `IX_Delivery_AddressId` USING BTREE (`AddressId`) VISIBLE,
  CONSTRAINT `fk_Delivery_1`
    FOREIGN KEY (`OrderId`)
    REFERENCES `coffee`.`Orders` (`Id`),
  CONSTRAINT `FK_Delivery_Addresses_AddressId`
    FOREIGN KEY (`AddressId`)
    REFERENCES `coffee`.`Addresses` (`Id`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Создание таблицы “Сотрудники”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Employees` (
   `Id` INT NOT NULL AUTO_INCREMENT,
   `FirstName` VARCHAR(255) NOT NULL,
   `LastName` VARCHAR(255) NOT NULL,
```

```
   `Phone` VARCHAR(63) NOT NULL,
   `Email` VARCHAR(255) NOT NULL,
   PRIMARY KEY (`Id`, `Phone`, `Email`))
 ENGINE = InnoDB
 AUTO_INCREMENT = 3
 DEFAULT CHARACTER SET = utf8mb4
 COLLATE = utf8mb4_0900_ai_ci;

CREATE TABLE IF NOT EXISTS `coffee`.`JobTitle` (
   `EmployeeId` INT NOT NULL,
   `StoreId` INT NULL,
   `Title` VARCHAR(63) NOT NULL,
   `Salary` DECIMAL(10,2) NOT NULL,
   PRIMARY KEY (`EmployeeId`),
   INDEX `fk_JobTitle_2_idx` (`StoreId` ASC) VISIBLE,
   CONSTRAINT `fk_JobTitle_1`
     FOREIGN KEY (`EmployeeId`)
     REFERENCES `coffee`.`Employees` (`Id`)
     ON DELETE NO ACTION
     ON UPDATE NO ACTION,
   CONSTRAINT `fk_JobTitle_2`
     FOREIGN KEY (`StoreId`)
     REFERENCES `coffee`.`Stores` (`Id`)
     ON DELETE NO ACTION
     ON UPDATE NO ACTION)
 ENGINE = InnoDB;
```

Создание таблицы “Продукты”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Products` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `Title` VARCHAR(255) NOT NULL,
  `Description` TEXT NOT NULL,
  `Price` DECIMAL(10,2) NOT NULL,
  `ImageURI` VARCHAR(255) NULL DEFAULT NULL,
  PRIMARY KEY (`Id`))
ENGINE = InnoDB
AUTO_INCREMENT = 0
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

Создание таблицы “Меню”

```
CREATE TABLE IF NOT EXISTS `coffee`.`Menu` (
  `ProductId` INT NOT NULL,
  `StoreId` INT NOT NULL,
  PRIMARY KEY (`StoreId`, `ProductId`),
  INDEX `IX_ProductStore_StoreId` USING BTREE (`StoreId`) VISIBLE,
  INDEX `FK_ProductStore_Products_MenuId` (`ProductId` ASC) VISIBLE,
  CONSTRAINT `FK_ProductStore_Products_MenuId`
    FOREIGN KEY (`ProductId`)
    REFERENCES `coffee`.`Products` (`Id`)
```

```
    ON DELETE CASCADE,
  CONSTRAINT `FK_ProductStore_Stores_StoreId`
    FOREIGN KEY (`StoreId`)
    REFERENCES `coffee`.`Stores` (`Id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE TABLE IF NOT EXISTS `coffee`.`OrderProduct` (
  `OrderId` INT NOT NULL,
  `ProductsId` INT NOT NULL,
  `Quantity` INT NOT NULL DEFAULT '1',
  PRIMARY KEY (`OrderId`, `ProductsId`),
  INDEX `IX_OrderProduct_ProductsId` USING BTREE (`ProductsId`) VISIBLE,
  INDEX `FK_OrderProduct_Orders_OrderId` (`OrderId` ASC) VISIBLE,
  CONSTRAINT `FK_OrderProduct_Orders_OrderId`
    FOREIGN KEY (`OrderId`)
    REFERENCES `coffee`.`Orders` (`Id`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT,
  CONSTRAINT `FK_OrderProduct_Products_ProductsId`
    FOREIGN KEY (`ProductsId`)
    REFERENCES `coffee`.`Products` (`Id`)
    ON DELETE CASCADE
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

`INDEX IX_Orders_Status USING BTREE (Status) VISIBLE`:

Этот пример индекса созданного для оптимизации запросов, которые фильтруют или сортируют данные по полю `Status` в таблице `Orders`. B-tree индекс позволяет быстрее находить заказы с определенным статусом и ускоряет выполнение таких запросов.

Все подобные индексы используют B-tree структуру, которая является наиболее распространенным типом индекса для большинства СУБД. B-tree индексы хорошо подходят для операций сравнения, таких как равенство, больше, меньше и т.д. Они также поддерживают сортировку данных и могут ускорять выполнение запросов с операторами `ORDER BY` и `GROUP BY`, однако помимо всего этого они значительно замедляют работу в операциях `INSERT`, `UPDATE` и `DELETE`, поэтому не следует злоупотреблять добавлением индексов.

По созданию представлений

```
CREATE VIEW `coffee`.`SalesStatistics` AS
SELECT
  p.Title AS `Product`,
  SUM(op.Quantity) AS `Total Sales`,
  COUNT(DISTINCT o.Id) AS `Total Orders`
FROM
  `coffee`.`Products` p
  JOIN `coffee`.`OrderProduct` op ON p.Id = op.ProductsId
  JOIN `coffee`.`Orders` o ON op.OrderId = o.Id
GROUP BY
  p.Id
ORDER BY
  `Total Sales` DESC;
```

По созданию триггеров

Триггер для проверки соответствуют ли сумма заказа на доставку минимальной

```
CREATE TRIGGER `CheckDeliveryMinAmount`
BEFORE INSERT ON `coffee`.`Delivery`
FOR EACH ROW
BEGIN
    IF GetOrderTotal(NEW.`OrderId`) < 400 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'The minimum order amount for
delivery is 400 rubles.';
    END IF;
END
```

Триггер для проверки открыта точка или нет

```
CREATE TRIGGER `IsStoreOpen`
BEFORE INSERT ON `Orders`
FOR EACH ROW
BEGIN
IF NOT (TIME(NOW()) BETWEEN (
SELECT `OpeningTime`
FROM `Stores` WHERE `Id` = NEW.`StoreId`) AND (SELECT `ClosingTime` FROM `Stores`
WHERE `Id` = NEW.`StoreId`)) THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot insert, the store is
closed.';
   END IF;
END
```

По созданию процедур

Процедура для создания нового пользователя

```
CREATE PROCEDURE `CreateCustomer`(
    IN Name VARCHAR(63),
       IN Phone VARCHAR(63),
       IN Email VARCHAR(255)
)
BEGIN
       INSERT INTO Customers(Email, Phone, Name)
    VALUES (Email, Phone, Name);
END
```

Процедура для создания нового сотрудника

```
CREATE PROCEDURE `CreateEmployee`(
       IN FirstName VARCHAR(255),
    IN LastName VARCHAR(255),
    IN Title VARCHAR(63),
    IN Salary DECIMAL(10,2),
    IN StoreId INT,
       IN Phone VARCHAR(255),
       IN Email VARCHAR(255)
)
BEGIN
       INSERT INTO Employees(FirstName, LastName, Phone, Email)
    VALUES (FirstName, LastName, Phone, Email);
INSERT INTO JobTitle(Title, Salary, StoreId)
    VALUES (Title, Salary, StoreId);
END
```

Процедура для создания новой точки

```
CREATE PROCEDURE `CreateStore`(
    IN `Town` VARCHAR(255),
    IN `Street` VARCHAR(255),
    IN `House` VARCHAR(63),
    IN `Entrance` VARCHAR(63),
    IN `Flat` VARCHAR(63),
    IN `OpeningTime` TIME,
    IN `ClosingTime` TIME
)
BEGIN
    DECLARE v_AddressID INT;

    SELECT Id INTO v_AddressID
    FROM coffee.Addresses a
    WHERE a.Town = Town AND a.Street = Street AND a.House = House AND a.Entrance =
Entrance AND a.Flat = Flat
```

```
    LIMIT 1;

        IF v_AddressID IS NULL THEN
        INSERT INTO `coffee`.`Addresses`(`Town`, `Street`, `House`, `Entrance`,
`Flat`)
        VALUES (Town, Street, House, Entrance, Flat);

        SET v_AddressID = LAST_INSERT_ID();
    END IF;

        INSERT INTO `coffee`.`Stores`(AddressId, OpeningTime, ClosingTime)
    VALUES (v_AddressID, OpeningTime, ClosingTime);

    SELECT v_AddressID AS AddressID;
END$$
```

Процедура для обновление должности и/или зарплаты сотрудника

```
CREATE PROCEDURE `UpdateEmployee`(
    IN p_Id INT,
    IN p_Title VARCHAR(63),
    IN p_Salary DECIMAL(10,2)
)
BEGIN
        IF p_Title IS NOT NULL THEN
        UPDATE JobTitle
        SET Title = p_Title
        WHERE Id = p_Id;

        ELSEIF p_Salary IS NOT NULL THEN
        UPDATE JobTitle
        SET Salary = p_Salary
        WHERE Id = p_Id;

    ELSE
            UPDATE JobTitle
            SET Title = p_Title, Salary = p_Salary
            WHERE Id = p_Id;
      END IF;
END$$
```

Процедура для обновления статуса заказа

```
CREATE PROCEDURE `UpdateOrderStatus`(
    IN p_Id INT,
    IN p_Status VARCHAR(63)
)
BEGIN
        UPDATE Orders o
    SET o.`Status` = p_Status
    WHERE Id = p_Id;
END
```

По созданию функций

Функция для получения стоимости заказа:

```
CREATE FUNCTION `GetOrderTotal`(order_id INT) RETURNS int
    READS SQL DATA
BEGIN
  DECLARE total_price INT;

  SELECT SUM(`Quantity` * Price) INTO total_price
  FROM `coffee`.`OrderProduct`
  JOIN `coffee`.`Products` ON `coffee`.`OrderProduct`.`ProductsId` =
`Products`.`Id`
  WHERE `coffee`.`OrderProduct`.`OrderId` = order_id;

  RETURN total_price;
END
```

SQL и NoSQL оценка оптимальности использования

Использование NoSQL баз данных имеет смысл в тех случаях, когда необходимо хранить и управлять большим количеством неструктурированных данных, таких как документы, изображения, видео и т.д. NoSQL базы данных обеспечивают гибкость и масштабируемость, что позволяет эффективно хранить и обрабатывать данные в режиме реального времени.

Однако, использование NoSQL баз данных не всегда является необходимым. Если данные структурированные и нет необходимости в гибкости и масштабируемости, то использование SQL баз данных может быть более подходящим вариантом.

Для кофейни можно использовать как SQL, так и NoSQL базы данных. Рассмотрим плюсы и минусы каждого вида БД:

SQL базы данных

Плюсы	*Минусы*
Хорошо подходят для хранения структурированных данных, таких как данные клиентов, закупок и работников.	Могут быть менее гибкими и масштабируемыми, чем NoSQL базы данных.
Позволяют эффективно использовать язык SQL для выполнения запросов и анализа данных.	Могут быть менее эффективными для хранения неструктурированных данных, таких как изображения и видео.
Обеспечивают высокую степень целостности и безопасности данных.	Могут требовать более высокой стоимости обслуживания и управления.

Хорошо подходят для выполнения сложных запросов и агрегирования данных.	

NoSQL базы данных

Плюсы	*Минусы*
Хорошо подходят для хранения неструктурированных данных, таких как изображения и видео.	Могут быть менее безопасными и менее целостными, чем SQL базы данных.
Обеспечивают высокую гибкость и масштабируемость, что позволяет эффективно хранить и обрабатывать данные в режиме реального времени.	Могут быть менее эффективными для выполнения сложных запросов и агрегирования данных.
Могут быть более доступными и отказоустойчивыми, чем SQL базы данных.	Могут требовать более высокой стоимости обучения и разработки.
Могут быть более экономичными в обслуживании и управлении.	

В целом, выбор между SQL и NoSQL базами данных зависит от конкретных потребностей и характера данных, которые необходимо хранить и управлять. Если данные структурированные, то SQL базы данных могут быть более подходящим вариантом. Если неструктурированные, то NoSQL.

Учитывая все вышеперечисленные достоинства и недостатки каждой из видов баз данных, можно сделать вывод, что в данном случае данные отлично структурированы и выбор можно сделать в пользу SQL базы данных.

Гитхаб