

СОДЕРЖАНИЕ

ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	2
СТРУКТУРНО-ФУНКЦИОНАЛЬНЫЙ МЕТОД МОДЕЛИРОВАНИЯ	4
Модель в нотации IDEF0.....	4
Модель в нотации DFD.....	7
ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ МЕТОД МОДЕЛИРОВАНИЯ.....	11
Диаграмма вариантов использования (use case) в нотации UML	11
Диаграмма классов (class diagram) в нотации UML	12
Диаграмма последовательности (sequence diagram) и диаграмма коопераций (colaboration diagram) в нотации UML	14
ИНФОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ	19
Разработка логической модели базы данных	19
Разработка модели базы данных в нотации Питера Чена.....	20
Разработка даталогической (физической) модели базы данных.....	21
ПРАКТИЧЕСКАЯ РАБОТА №1	23
ПРАКТИЧЕСКАЯ РАБОТА №2	32
ПРАКТИЧЕСКАЯ РАБОТА №3	40
ПРАКТИЧЕСКАЯ РАБОТА №4	48
ЗАКЛЮЧЕНИЕ	57

ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметной областью выбрано проектирование базы данных кафе.

Само кафе представляет собой здание, в котором находятся помещения, предназначенные для обслуживания потребителей и приготовления блюд. Кафе имеет собственную информационную систему для удобства работы с клиентами.

При посещении кафе клиента встречает официант и размещает его за столиком. Затем официант приносит меню, чтобы у клиента была возможность сделать выбор блюд.

После того как клиент озвучивает заказ, официант передаёт всю необходимую информацию о заказе повару, а также заносит данные в информационную систему кафе.

Приготовленные поваром блюда подаются клиенту официантом. Когда клиент заканчивает употребление пищи, официант производит расчёт.

Сущности, которые будут содержать указанные выше модули, могут быть описаны следующим образом:

Помещения: это может быть таблица, которая хранит информацию о доступных для расположения клиентов залах в кафе. Каждое помещение может иметь уникальный идентификатор, вместимость, количество столиков.

Отзывы: это может быть таблица, которая хранит информацию об отзывах, оставленных клиентами. Эта таблица может содержать идентификатор отзыва, идентификатор заказа, комментариев и оценку.

Управление клиентами: это может быть таблица, которая хранит информацию о всех посетителях кафе и сделанных ими заказах. Эта таблица может содержать такие поля, как идентификатор клиента, идентификатор заказа, ФИО, пол, номер телефона и т.д.

Блюда: это может быть таблица с данными о всех блюдах, подаваемых в кафе. Эта таблица может содержать такие поля, как идентификатор блюда, название блюда, категория блюда и цена блюда.

Основное меню: это может быть таблица с данными об основных блюдах, подаваемых в кафе в любое время. Эта таблица может содержать такие поля, как идентификатор основного меню, названия блюд основного меню, состав блюд основного меню и стоимость блюд основного меню.

Сезонное меню: это может быть таблица с данными об особенных блюдах, подаваемых в кафе только в определённое время. Эта таблица может содержать такие поля, как идентификатор сезонного меню, названия блюд сезонного меню, состав блюд сезонного меню и стоимость блюд сезонного меню.

Информация о заказах: таблица, содержащая информацию о сделанных в кафе заказах. Эта таблица может содержать такие поля, как идентификатор заказа, содержание заказа, номер столика, идентификатор официанта, идентификатор повара и идентификатор блюда.

Сотрудники: таблица с информацией о сотрудниках, работающих в кафе. Эта таблица может содержать такие поля, как имя, фамилия, идентификатор сотрудника и другие.

Таким образом, предложенная база данных позволит кафе удобно управлять операциями по обслуживанию клиентов. Это позволит кафе обеспечивать более комфортный сервис и повышать удовлетворенность клиентов. Также эти системы могут использоваться для генерации отчетов и анализа динамики посещаемости.

СТРУКТУРНО-ФУНКЦИОНАЛЬНЫЙ МЕТОД МОДЕЛИРОВАНИЯ

Модель в нотации IDEF0

IDEF0 (Integrated Definition for Function Modeling) - это графическая нотация, которая используется для моделирования функциональных процессов в системе. Она представляет собой способ описания того, как система работает, путем разбиения ее функций на более простые составляющие части.

IDEF0 состоит из блоков и стрелок, которые образуют блок-схему процесса. Блоки представляют собой функциональные элементы системы, а стрелки - связи между различными функциями. Стрелки указывают направление перехода данных и информацию о том, какая функция именно обрабатывает данные. Блоки содержат в себе информацию о контексте, цели и результирующих действиях.

IDEF0 позволяет структурировать функциональные процессы системы и выделить проблемные моменты, которые могут понадобиться для оптимизации и улучшения ее работы.

Для модели работы кафе, используя нотацию IDEF0 входящим потоком будет клиент, выходящим – клиент, удовлетворённый или неудовлетворённый обслуживанием кафе, прибыль кафе и пищевые отходы.

Управляющими потоками будут являться график работы кафе, Закон «О защите прав потребителей», стандарты обслуживания клиентов.

Ресурсами будут являться работники кафе (официанты, повара), информационная система кафе, оборудование, меню, помещение и продукты.

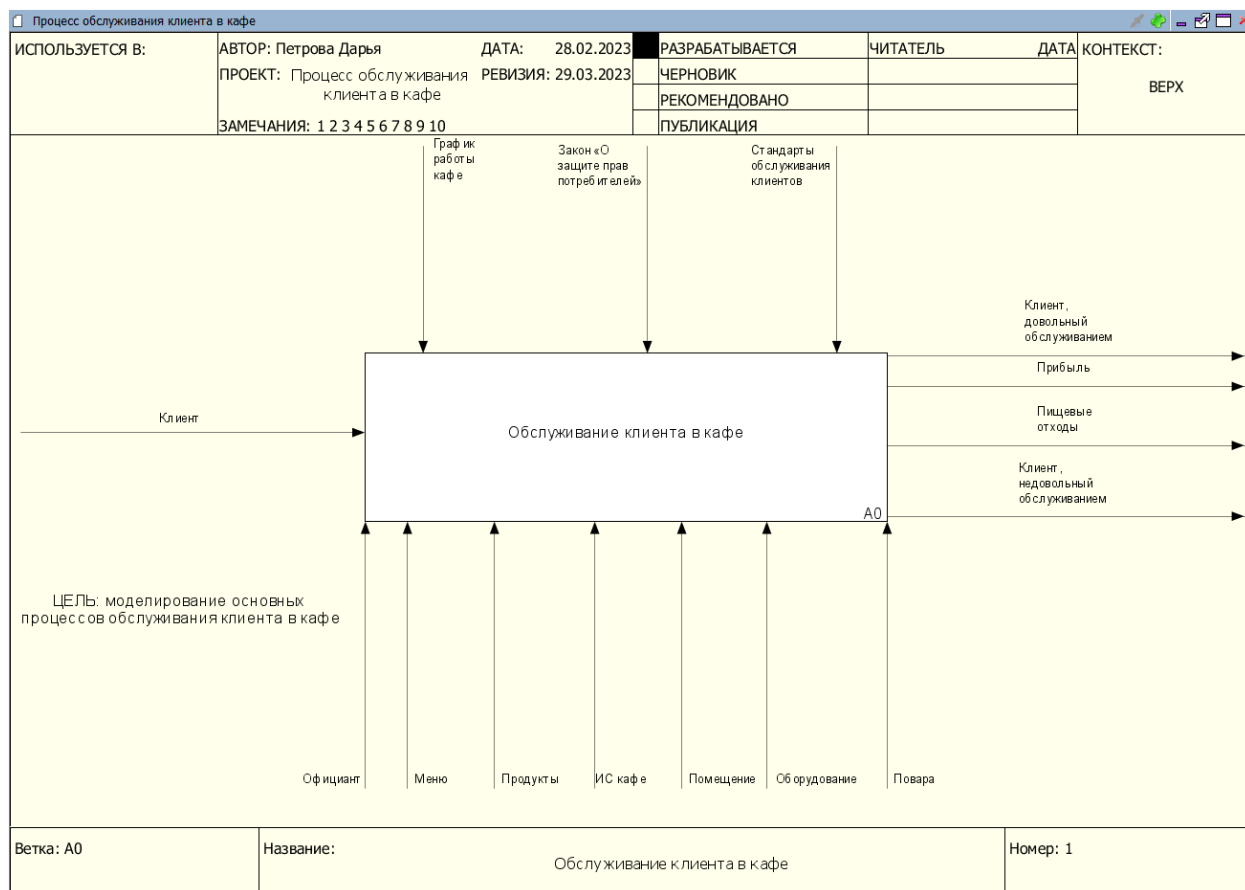


Рисунок 1 – Контекстный уровень

На первом уровне декомпозиции будут использоваться все входящие, выходящие, управляющие потоки и ресурсы. Они будут распределены между блоками (принять заказ, приготовить блюда, подать готовые блюда клиенту и произвести расчёт).

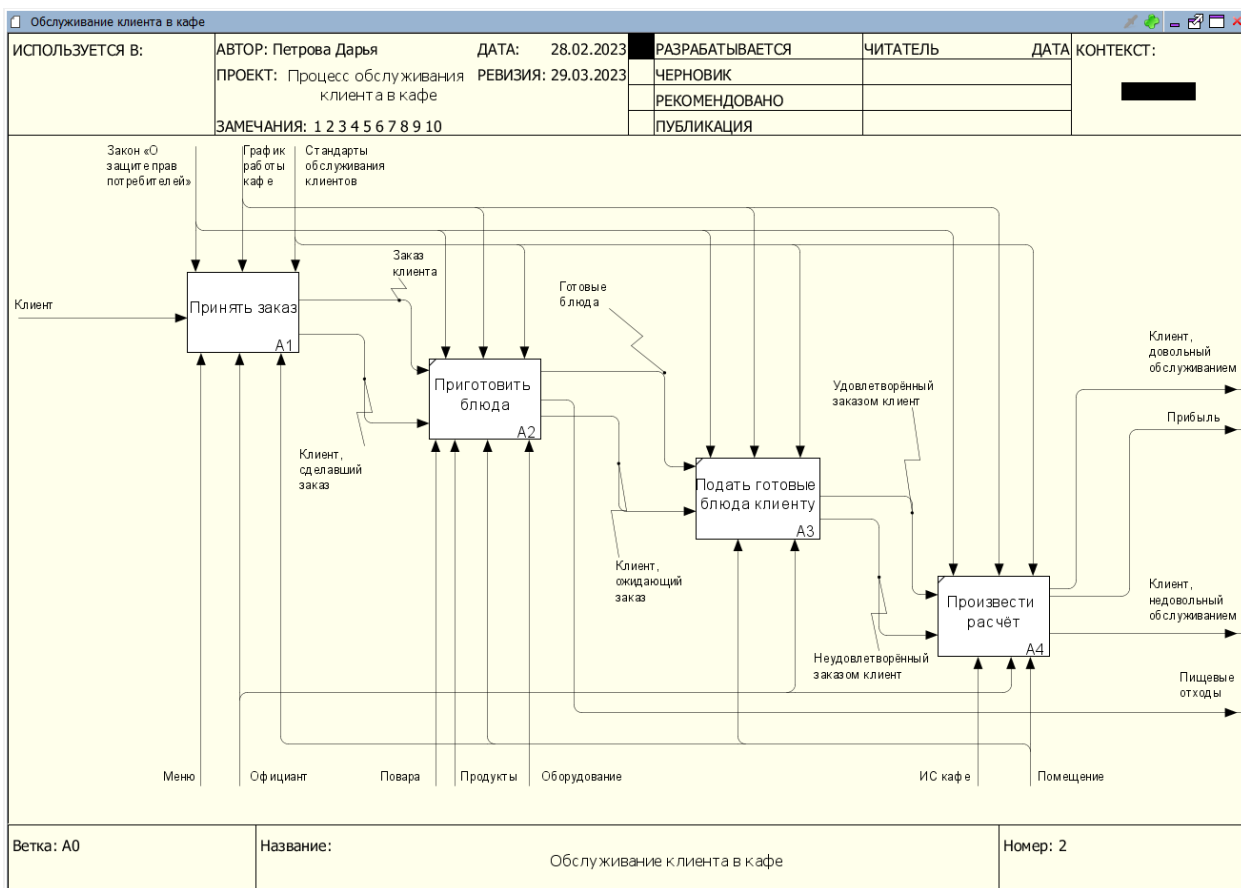


Рисунок 2 – Первый уровень декомпозиции

На второй уровень декомпозиции перейдут уже не все потоки, а только необходимые для блока «принять заказ» с первого уровня. Они также будут разделены между блоками (разместить клиента за столиком, принести клиенту меню, предоставить время на выбор блюд и запомнить заказ клиента).

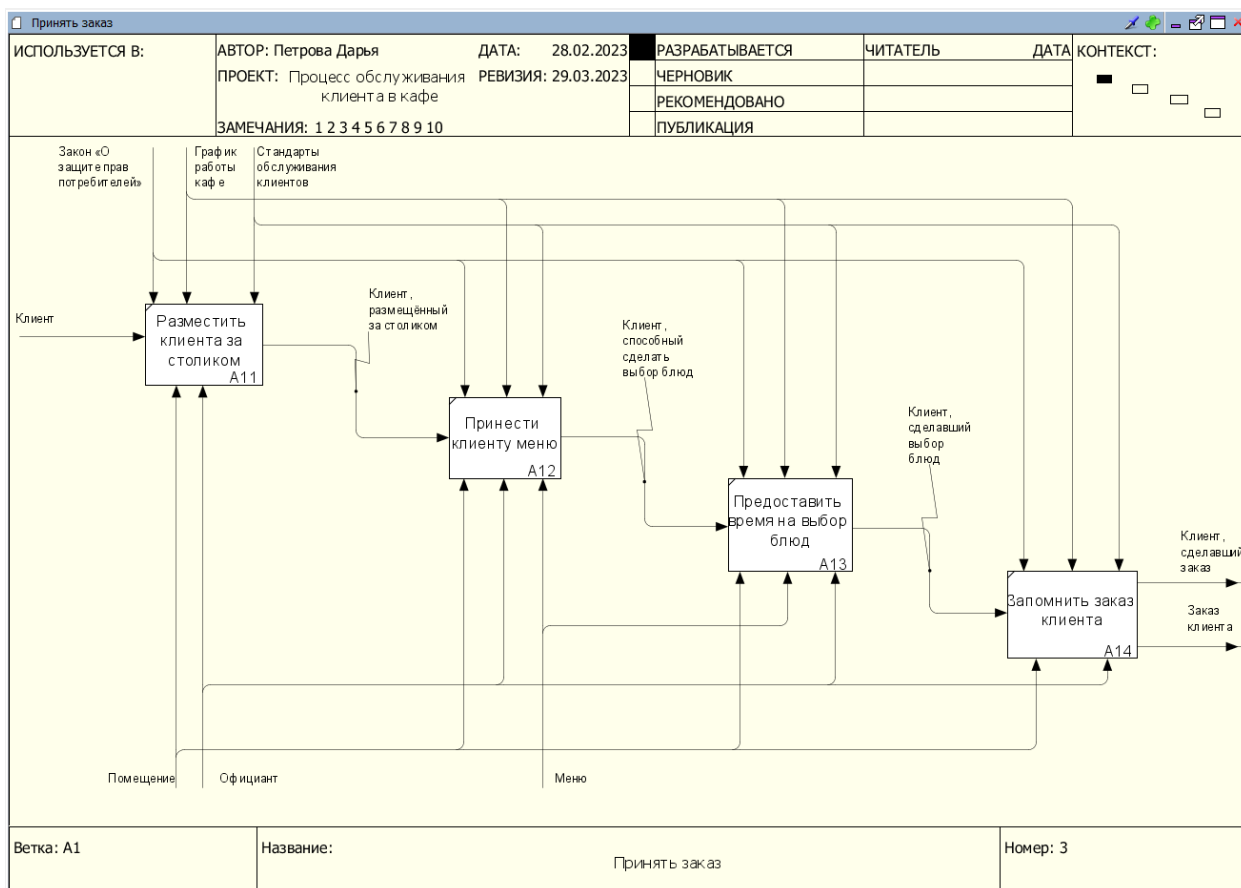


Рисунок 3 – Второй уровень декомпозиции

Модель в нотации DFD

DFD (Data Flow Diagram) - это графическая нотация, которая используется для описания потоков данных через процессы в системе. Эта нотация помогает понять, как данные перемещаются по системе и проходят обработку на каждом этапе.

DFD состоит из блоков, связей и процессов. Блоки представляют собой источники данных, места назначения и промежуточные системные компоненты, где происходит обработка данных. Связи между блоками показывают направление и поток данных между ними. Процессы, в свою очередь, представляют собой операции, которые выполняют различные функции или преобразуют данные.

DFD позволяет описать взаимодействие компонентов системы и сфокусироваться на важных аспектах, таких как ограничение потоков данных, управление изменениями и оптимизация процессов.

Для данной модели блоки и сущности были выбраны такие же, как и на предыдущей диаграмме.

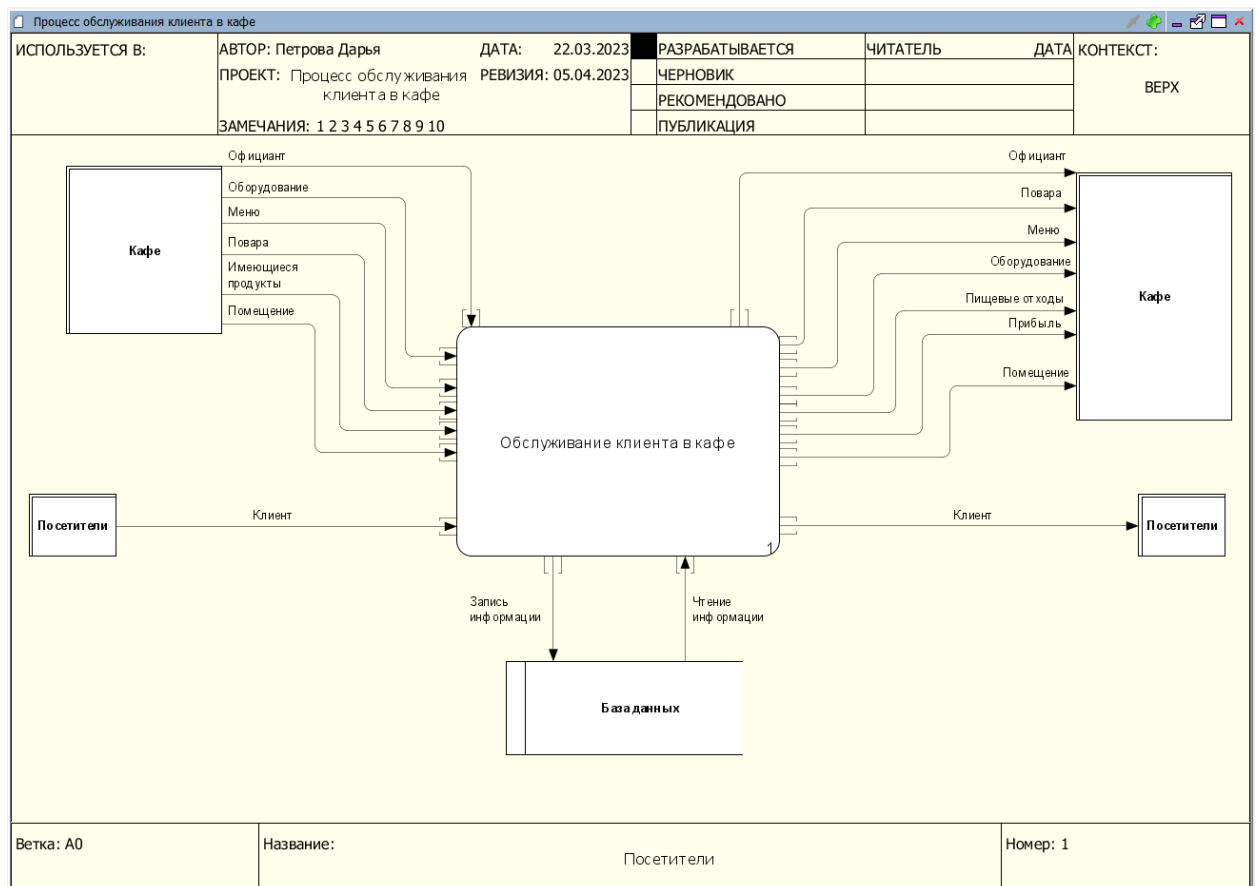


Рисунок 4 – Контекстный уровень

Начиная с первого уровня декомпозиции, «База данных» с первого блока превращается во множество других. Появляются хранилища информации о посетителях, информации о заказах, о сотрудниках, о блюдах и хранилище документов.

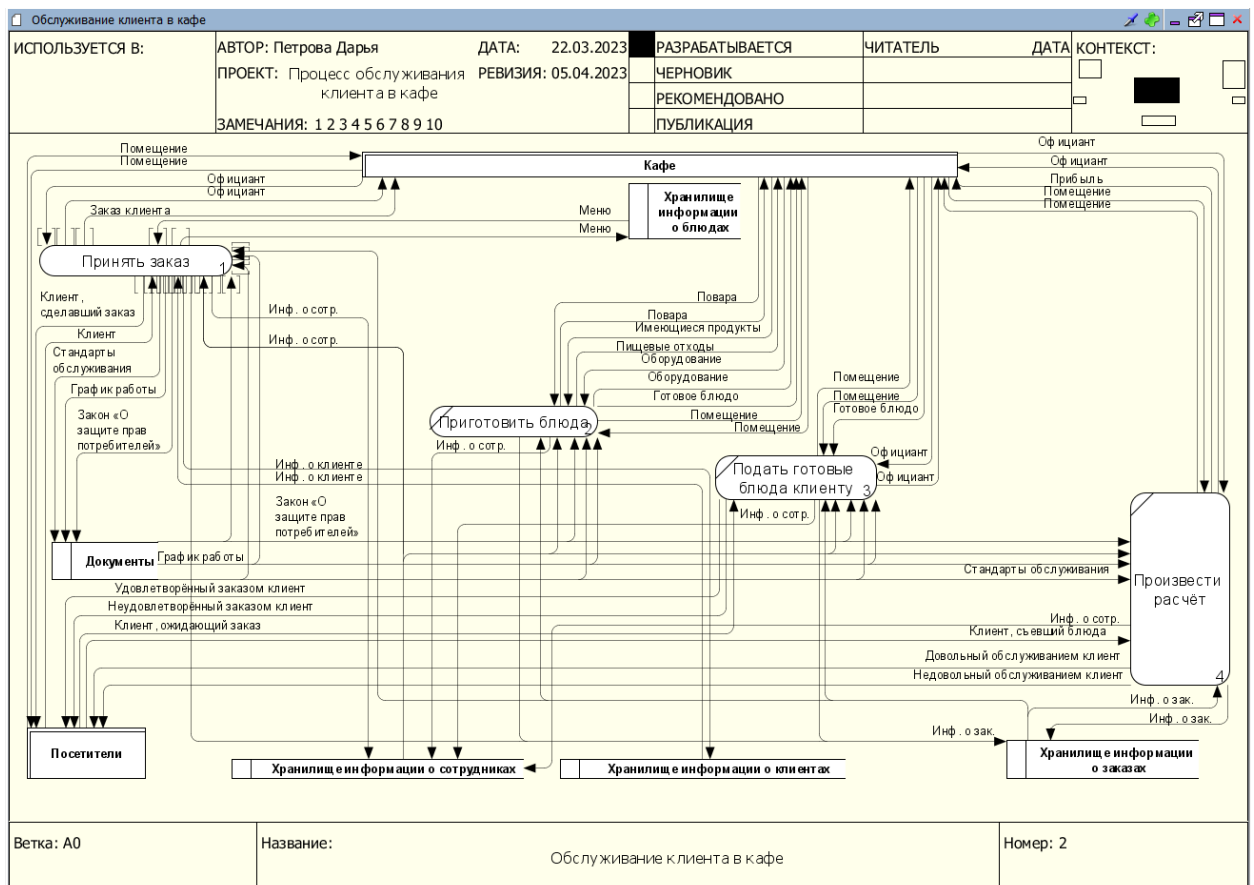


Рисунок 5 – Первый уровень декомпозиции

Второй уровень декомпозиции так же, как и на диаграмме предыдущей нотации, показывает разбиение на блоки процесса «принять заказ».

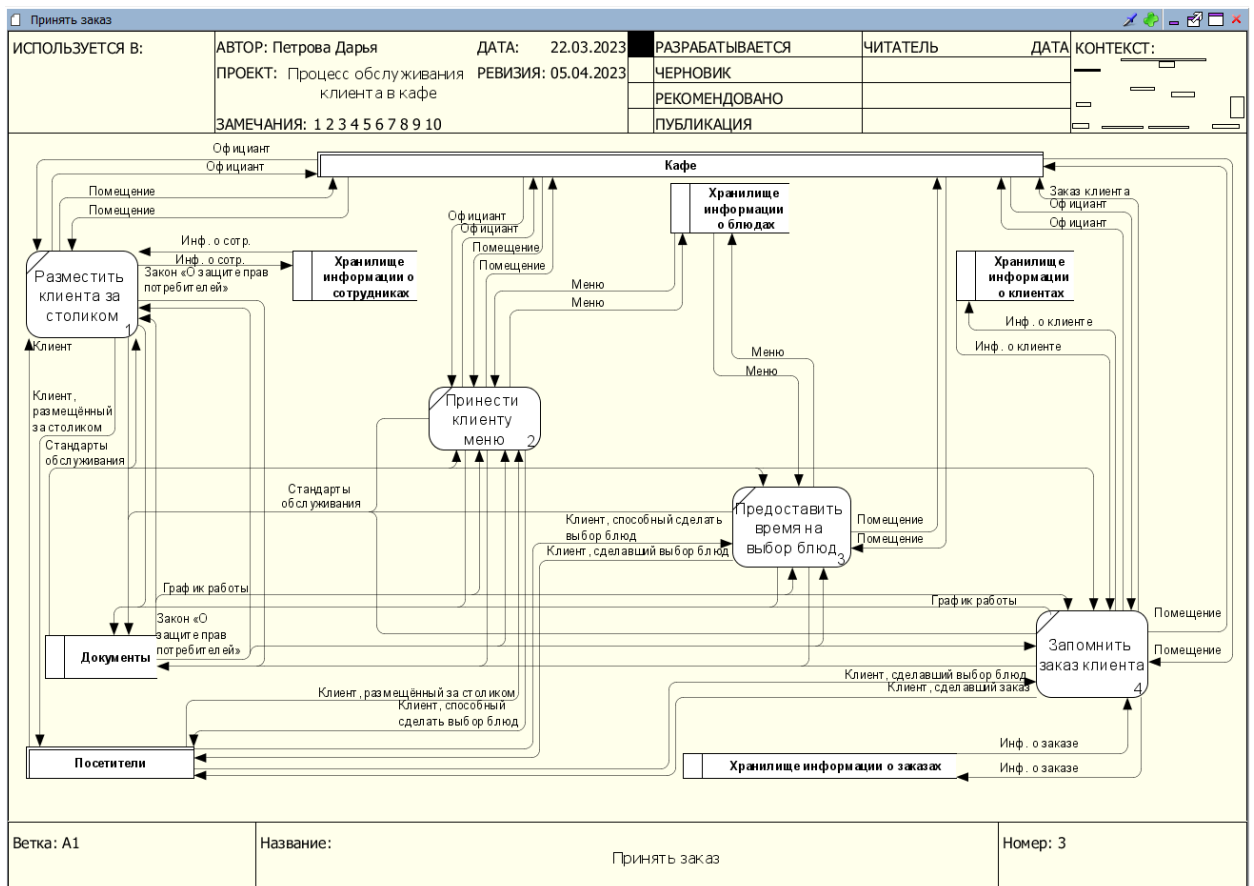


Рисунок 6 – Второй уровень декомпозиции

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ МЕТОД МОДЕЛИРОВАНИЯ

Диаграмма вариантов использования (use case) в нотации UML

UML (Unified Modeling Language) - это язык, который используется для визуального моделирования абстрактных, сложных искусственных систем. UML состоит из нескольких видов диаграмм, включая диаграмму Use Case.

Диаграмма Use Case - это тип диаграммы UML, который позволяет описать, как пользователи взаимодействуют с системой. Она отображает потоки взаимодействия между пользователями и системой и позволяет разрабатывать функциональные требования к системе. Диаграмма состоит из акторов (пользователи) и вариантов использования (функциональные возможности системы).

Акторы - это представители внешней среды, взаимодействующие с системой. Варианты использования - это функции или действия, которые выполняет система.

Диаграмма Use Case помогает определить взаимодействия между пользователями и системой, понять, какую функциональность должна предоставлять система и как отдельные пользователи будут взаимодействовать с системой в своих ролях. Это позволяет лучше понимать потребности пользователей и улучшать процессы внутри системы.

В целом, данная диаграмма является важным инструментом моделирования системы и используется для определения функциональности системы и ее требований к проектированию. Учитывая, что она является частью UML, она позволяет обеспечить общее понимание взаимодействия между пользователями и системой всем участникам проекта.

Для лучшего понимания работы кафе была создана следующая Use Case диаграмма:

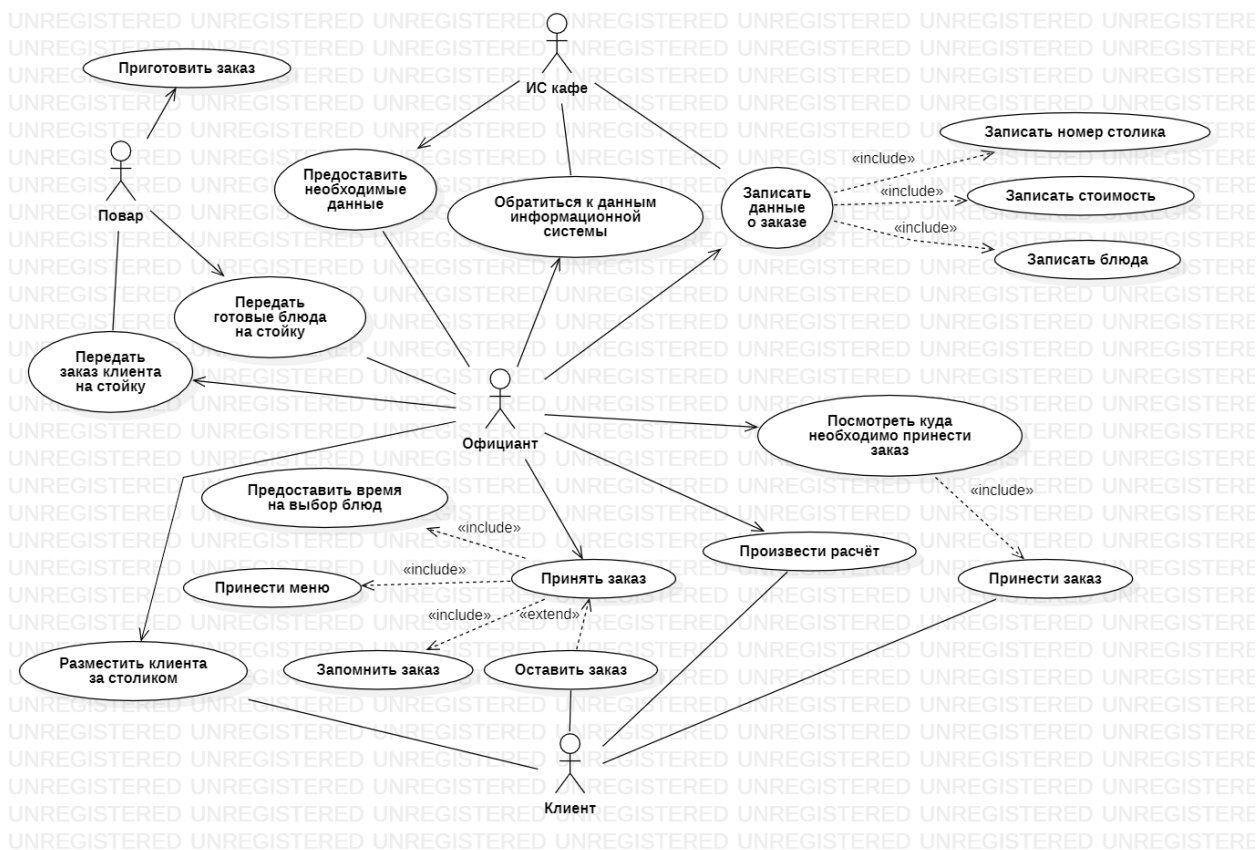


Рисунок 7 – Диаграмма Use Case

Диаграмма классов (class diagram) в нотации UML

Диаграмма классов - это одна из диаграмм UML для визуального представления классов, их связей, методов и атрибутов. Она используется для моделирования основных структур данных в системе.

Диаграмма классов состоит из классов, соединенных линиями связей. Классы могут содержать атрибуты, методы и другие метаданные, а связи могут отображать различные отношения между классами, например, наследование, ассоциацию, агрегацию, композицию и зависимость.

Классы - это шаблоны для создания объектов, содержащие атрибуты (свойства) и методы (операции), которые определяют поведение объектов.

Атрибуты описывают состояние объекта, а методы описывают действия, которые объект может выполнять.

Связи - это отношения между классами, которые описывают взаимодействие между объектами. Например, наследование показывает, что один класс наследует свойства и методы другого класса, а ассоциация описывает, что объекты одного класса связаны с объектами другого класса.

Диаграмма классов помогает визуально представить структуру и зависимости классов и других компонентов системы, что позволяет проще понимать взаимодействие между ними. Она также является основой для разработки кода, так как классы, связи и методы, представленные на диаграмме, служат основой для разработки программного кода.

В целом, диаграмма классов является важным инструментом при проектировании сложных систем и используется для создания модели данных и описания архитектуры приложения. Она также помогает связать технический язык программирования со сценариями продукта, что упрощает коммуникацию между несколькими участниками проекта.

Для процесса обслуживания клиента в кафе была создана диаграмма классов (см. рис. 8), где OrderInfo – информация о заказе, CafeSystem – информационная система кафе, DataBase – база данных, Client – клиент, Employee – работник, Cook – повар, Waiter – официант.

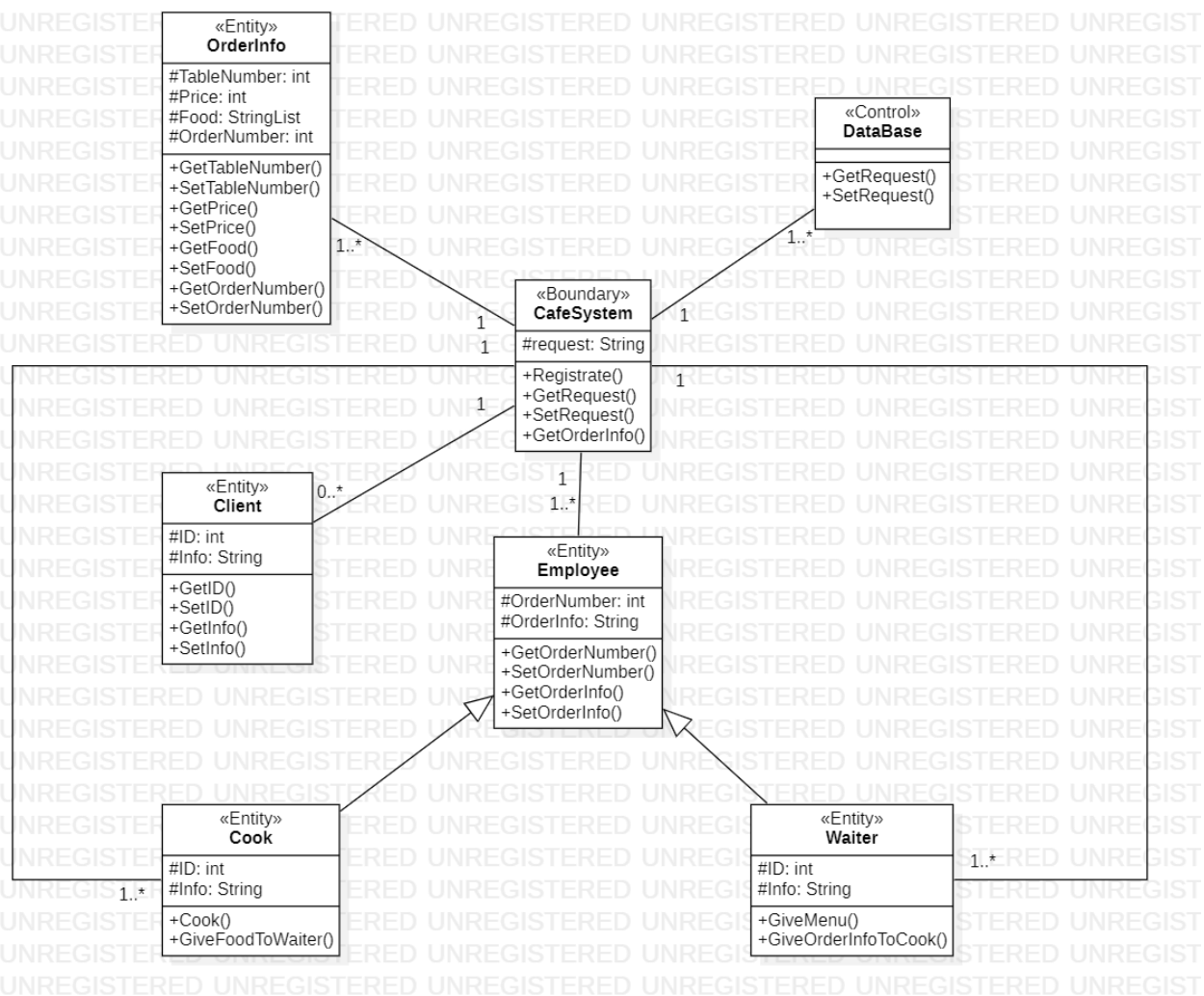


Рисунок 8 – Диаграмма классов

Диаграмма последовательности (sequence diagram) и диаграмма коопераций (colaboration diagram) в нотации UML

Диаграмма последовательности (sequence diagram) - это одна из диаграмм UML, которая используется для визуального отображения взаимодействия объектов в системе во времени. Она показывает, как объекты обмениваются сообщениями между собой и какой порядок сообщений.

Диаграмма последовательности состоит из вертикальных линий (жизненный цикл объектов) и стрелок (сообщения). Каждая вертикальная

линия представляет один объект, а стрелки между объектами показывают сообщения, которые передаются между ними.

На диаграмме последовательности можно отобразить не только обмен сообщениями между объектами, но и условия и операции, которые необходимо выполнить в процессе взаимодействия объектов. Также можно отразить параллельную работу объектов в процессе выполнения функций.

Диаграмма последовательности является важным инструментом при анализе и проектировании сложных систем. Она помогает понять, как объекты взаимодействуют друг с другом и каков порядок выполнения операций, что позволяет оптимизировать процесс разработки и упростить коммуникацию между участниками проекта.

Кроме того, диаграмма последовательности может быть использована для тестирования системы, так как она позволяет убедиться, что система выполняет функции в правильном порядке и с правильными параметрами.

В целом, диаграмма последовательности служит не только для визуализации процессов в системе, но и для понимания взаимодействия между объектами и улучшения процесса проектирования.

Для процесса обслуживания клиента в кафе была построена диаграмма последовательности для лучшего понимания взаимодействий клиента, работников кафе, их связь через систему и базу данных:

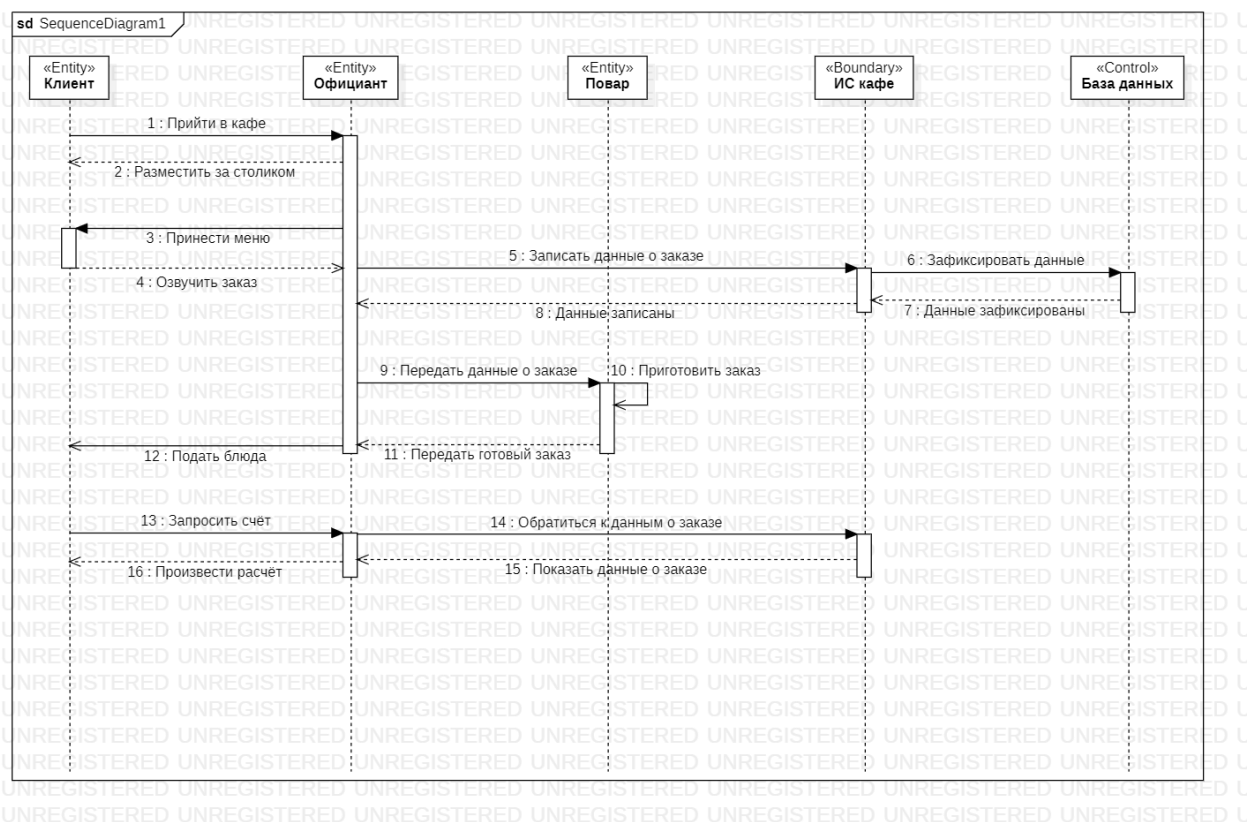


Рисунок 9 – Диаграмма последовательности

На диаграммах последовательности и коопераций, которые также являются диаграммами языка UML, можно использовать сущности (entity), границы (boundary) и элементы управления (control) для моделирования взаимодействия между объектами системы.

Как и на диаграмме классов, в диаграмме коопераций Entity Classes (Классы сущностей) являются объектами, задача которых состоит в представлении реальных объектов или их абстракций, взаимодействующих в системе. Границы (Boundary Classes) отвечают за обработку входящих и исходящих данных в системе, которые поступают от или передаются другим системам или пользователям. Следовательно, они помогают определить, каким образом система взаимодействует с внешним миром. Элементы управления (Control Classes) определяют логику, которая управляет тем, как система взаимодействует со своими сущностями и границами.

Диаграмма кооперации (collaboration diagram или communication diagram) - это одна из диаграмм UML, используемая для визуального отображения совместной работы или взаимодействия объектов в системе. Она показывает, как объекты взаимодействуют друг с другом, какие сообщения и каким порядком они обмениваются в процессе работы.

Диаграмма кооперации состоит из объектов, связей и сообщений, которые передаются между объектами. Объекты могут быть представлены в виде прямоугольника с названием класса, а сообщения - в виде стрелок между объектами. В диаграмме могут использоваться различные виды связей для показа отношений между объектами, включая ассоциации, агрегации, композиции и наследования.

Диаграмма кооперации является полезным инструментом для понимания совместной работы объектов и их влияния на другие объекты в системе. Она может использоваться для моделирования процессов совместной работы, обеспечения более эффективной коммуникации между различными участниками проекта и оптимизации процесса проектирования.

Диаграмма кооперации также может быть использована для определения структуры системы и выявления аспектов, требующих оптимизации. Она позволяет проанализировать взаимодействие объектов в системе и улучшить их взаимодействие и функциональность.

В целом, диаграмма кооперации предоставляет визуальное представление совместной работы объектов в системе и является важным инструментом анализа и проектирования сложных систем. Она помогает лучше понимать, как объекты взаимодействуют друг с другом и каких методов и процедур может не хватать для их более эффективной совместной работы.

Для процесса обслуживания клиента в кафе также была составлена диаграмма коопераций:

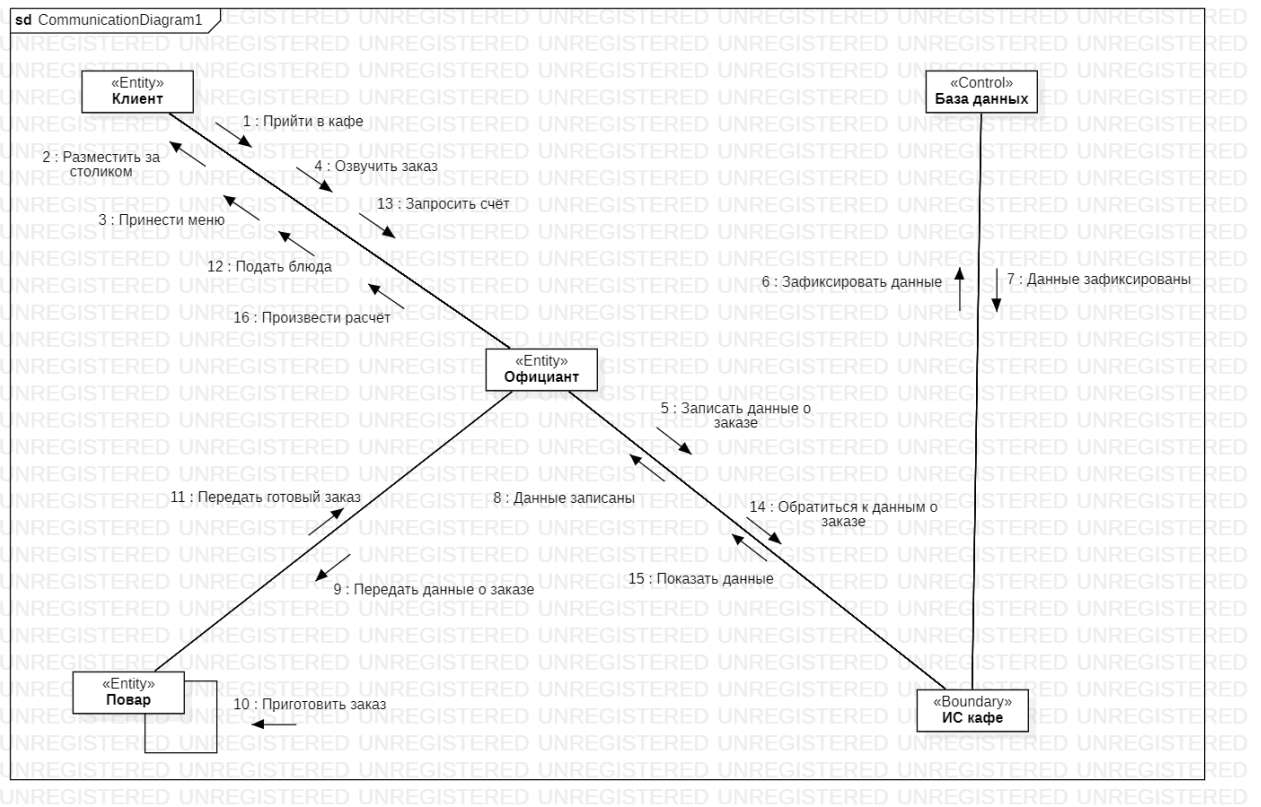


Рисунок 10 – Диаграмма коопераций

ИНФОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Разработка логической модели базы данных

Логическая модель базы данных представляет собой схему данных, которая описывает структуру информации, хранимой в базе данных. Она является вторым этапом в процессе проектирования базы данных после создания концептуальной модели, которая определяет сущности и их связи.

Логическая модель базы данных не зависит от конкретной СУБД и физической организации данных, а скорее фокусируется на том, как данные связаны друг с другом. Она включает в себя описание таблиц, полей, связей между таблицами, правил для данных, ограничений и дополнительных атрибутов.

Один из важнейших аспектов логической модели - это то, что она должна быть нормализована. Нормализация - это процесс разбиения таблиц на более мелкие, чтобы уменьшить дублирование данных и обеспечить более эффективный доступ к данным. Есть несколько форм нормализации, которые определяют, как таблицы должны быть разбиты, включая первую, вторую и третью нормальные формы.

Еще одним важным элементом логической модели являются индексы. Индекс позволяет быстро найти запись в таблице или наборе таблиц. Индекс обычно создается на поле, которое используется часто для поиска данных. Например, если в базе данных есть таблица клиентов, на которой вы хотите быстро найти всех клиентов с определенным именем, вы можете создать индекс на поле с именем клиента.

Логическая модель базы данных является основой для физической модели базы данных.

Для базы данных кафе была разработана следующая логическая модель:

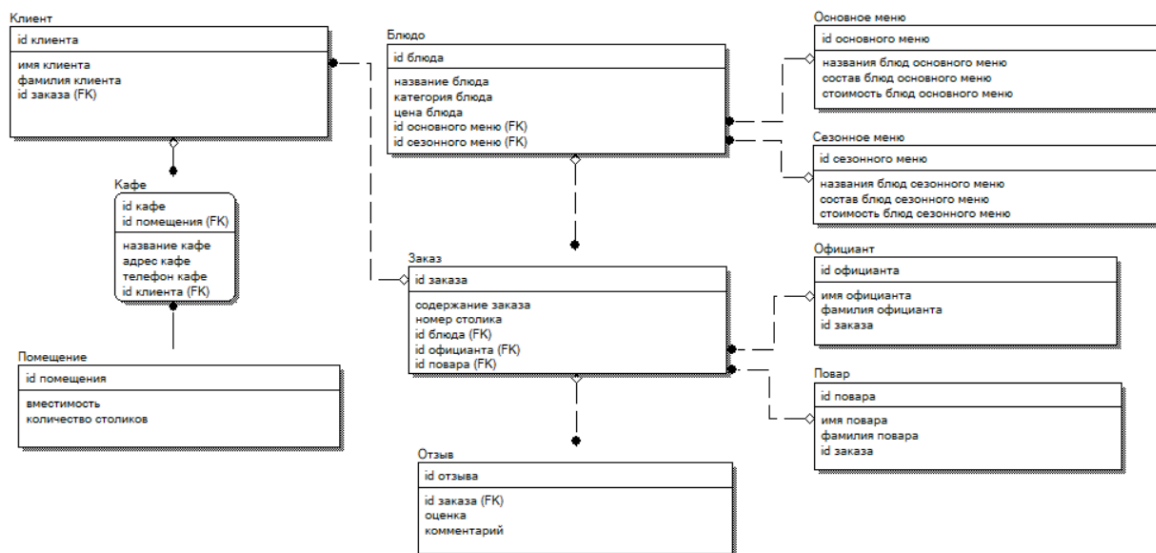


Рисунок 11 – Логическая модель

Разработка модели базы данных в нотации Питера Чена

Модель базы данных в нотации Питера Чена, также известная как модель "сущность-связь", является одной из самых распространенных моделей баз данных. Она была разработана в 1970-х годах и основана на представлении данных в виде сущностей и их связей.

Основу модели представляют сущности, которые представляют объекты или понятия, обычно связанные с бизнес-процессом или предметной областью. Сущности представляются в виде прямоугольников, внутри которых указывается название сущности.

Связи между сущностями обозначаются линиями, которые соединяют прямоугольники. У связей есть свои характеристики, которые определяют, каким образом связаны сущности. Например, у связи может быть атрибут, который определяет кратность связи (один к одному, один ко многим, многие к многим).

На диаграмме Питера Чена также можно отображать атрибуты, которые описывают характеристики сущности. Атрибуты можно представить в виде эллипсов, которые подключаются к сущности линиями со стрелками.

Атрибуты могут иметь различные типы данных (строки, числа, даты и т.д.), а также ограничения, которые определяют допустимые значения для атрибутов.

Одним из преимуществ модели в нотации Питера Чена является ее простота и наглядность, что делает ее удобной для визуализации сложных отношений между сущностями и их связей. Она также легко изменяема, что позволяет быстро обновлять модели при изменениях в бизнес-процессах или предметных областях.

Для обозначения связей между сущностями, созданными для процесса обслуживания клиента в кафе, была разработана модель базы данных в нотации Питера Чена.

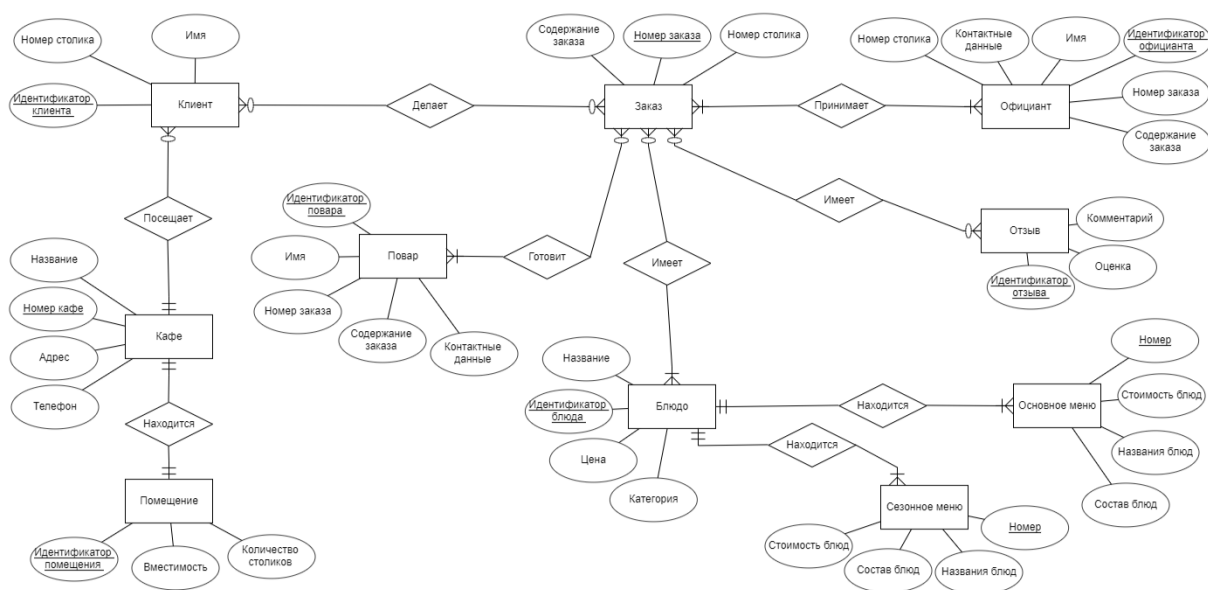


Рисунок 12 – Модель базы данных по нотации Питера Чена

Разработка даталогической (физической) модели базы данных

Физическая модель базы данных - это модель, которая описывает, как данные будут храниться на физическом уровне в базе данных. Она включает в

себя определение таблиц, столбцов, индексов, ограничений, ключей и других физических аспектов базы данных.

Когда это уже определено в логической модели базы данных, физическая модель базы данных определяется на основе недостающих физических характеристик, таких как размеры столбцов, типы данных, индексы и т. д.

Основная цель физической модели базы данных - это определить, как данные будут сохраняться на жестком диске, каким образом они будут извлекаться и какие будут использоваться алгоритмы для повышения эффективности работы с базой данных.

Для базы данных кафе была разработана следующая физическая модель:

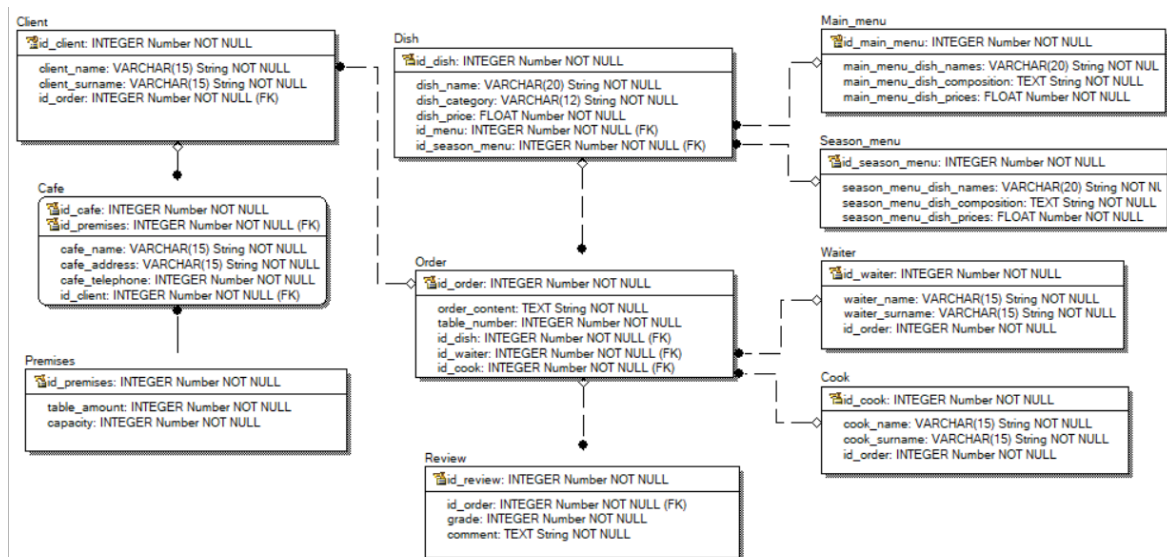


Рисунок 13 – Физическая модель

ПРАКТИЧЕСКАЯ РАБОТА №1

Создадим БД safe и выберем её для работы (Рис. 14).

```
mysql> create database cafe;  
Query OK, 1 row affected (0.01 sec)  
  
mysql> use cafe;  
Database changed
```

Рисунок 14 – Создание базы данных «safe»

Создадим 3 таблицы: order_of_client, user_data и cafe (Рис. 15).

```
mysql> create table order_of_client (id_order int, content text, table_number int);  
Query OK, 0 rows affected (0.11 sec)  
  
mysql> create table user_data (id_user int, name text, surname text);  
Query OK, 0 rows affected (0.05 sec)  
  
mysql> create table cafe (id_cafe int, name text, address text, telephone int);  
Query OK, 0 rows affected (0.06 sec)
```

Рисунок 15 – Создание таблиц

Смотрим все имеющиеся базы данных с помощью команды show databases и смотрим список таблиц базы данных cafe (Рис. 16).

```
mysql> show databases;
+-----+
| Database |
+-----+
| cafe     |
| information_schema |
| mysql    |
| performance_schema |
| sys      |
+-----+
5 rows in set (0.05 sec)

mysql> use cafe
Database changed
mysql> show tables
-> ;
+-----+
| Tables_in_cafe |
+-----+
| cafe           |
| order_of_client |
| user_data      |
+-----+
3 rows in set (0.04 sec)
```

Рисунок 16 – Просмотр всех БД и таблиц БД cafe

В ответе видим названия наших трех таблиц. Теперь посмотрим описание столбцов, например, таблицы cafe (Рис. 17). Удалим эту таблицу и снова посмотрим список всех таблиц в нашей БД.


```
mysql> describe cafe;
+-----+-----+-----+-----+-----+-----+
| Field      | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id_cafe    | int  | YES  |     | NULL    |       |
| name       | text | YES  |     | NULL    |       |
| address    | text | YES  |     | NULL    |       |
| telephone | int  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql> drop table cafe;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_cafe |
+-----+
| order_of_client |
| user_data       |
+-----+
2 rows in set (0.00 sec)
```

Рисунок 17 – Описание столбцов таблицы cafe и удаление таблицы

Наша таблица действительно удалена. Теперь удалим и саму БД cafe и убедимся в этом, сделав запрос на все имеющиеся БД (Рис. 18).

```
mysql> drop database cafe;
Query OK, 2 rows affected (0.07 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql         |
| performance_schema |
| sys          |
+-----+
4 rows in set (0.00 sec)
```

Рисунок 18 – Удаление БД

Снова создадим БД, усовершенствуем таблицы для нашего кафе и снова создадим три таблицы. Убедимся в том, что они созданы, с помощью команды show tables (Рис. 19).

```
mysql> show tables;
+-----+
| Tables_in_cafe |
+-----+
| client          |
| order_of_client |
| user_data       |
+-----+
3 rows in set (0.00 sec)
```

Рисунок 19 – Создание новых таблиц

И, наконец, посмотрим структуру наших таблиц (Рис. 20).

```
mysql> describe user_data;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id_user | int           | NO   | PRI | NULL    | auto_increment |
| name    | varchar(15)   | NO   |     | NULL    |                |
| surname | varchar(15)   | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.05 sec)

mysql> describe client;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id_client  | int           | NO   | PRI | NULL    | auto_increment |
| birth_date | varchar(10)   | NO   |     | NULL    |                |
| id_order   | int           | NO   | MUL | NULL    |                |
| id_user    | int           | NO   | MUL | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> describe order_of_client;
+-----+-----+-----+-----+-----+-----+
| Field          | Type | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id_order       | int  | NO   | PRI | NULL    | auto_increment |
| content        | text | NO   |     | NULL    |                |
| table_number   | int  | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

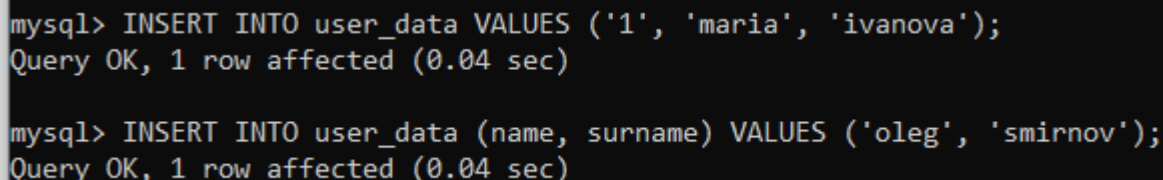
Рисунок 20 – Структура созданных таблиц

Теперь нам необходимо внести данные в наши таблицы. Для этого используется оператор INSERT. Попробуем внести в нашу таблицу user_data следующие значения:

```
INSERT INTO user_data VALUES ('1', 'maria', 'ivanova');
```

И попробуем второй вариант для внесения данных в некоторые поля таблицы. В нашей таблице все поля обязательны для заполнения, но наше первое поле имеет ключевое слово — `AUTO_INCREMENT` (т.е. оно заполняется автоматически), поэтому мы можем пропустить этот столбец (Рис. 21):

```
INSERT INTO user_data (name, surname) VALUES ('oleg', 'smirnov');
```

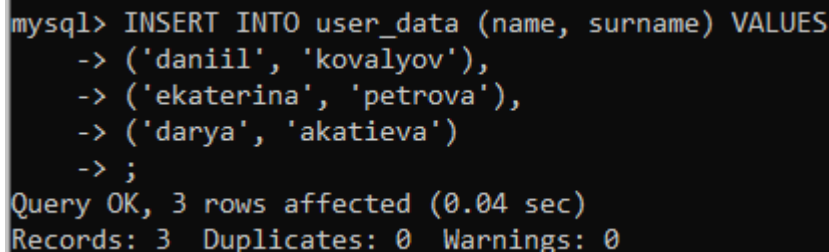


```
mysql> INSERT INTO user_data VALUES ('1', 'maria', 'ivanova');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO user_data (name, surname) VALUES ('oleg', 'smirnov');
Query OK, 1 row affected (0.04 sec)
```

Рисунок 21 – Внесение данных в таблицу user_data

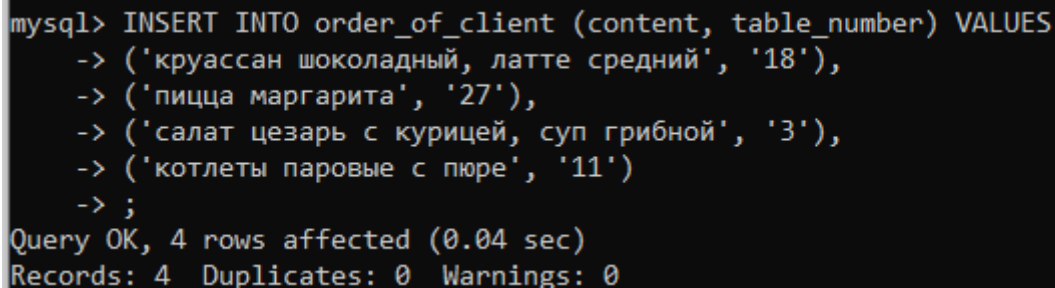
Внесем информацию еще о нескольких пользователях. Чтобы добавить сразу несколько строк, просто перечислим скобки со значениями через запятую (Рис. 22).



```
mysql> INSERT INTO user_data (name, surname) VALUES
-> ('daniil', 'kovalyov'),
-> ('ekaterina', 'petrova'),
-> ('darya', 'akatieva')
-> ;
Query OK, 3 rows affected (0.04 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

Рисунок 22 – Внесение данных в таблицу user_data

Внесем данные во вторую таблицу — `order_of_client` (Рис. 23).



```
mysql> INSERT INTO order_of_client (content, table_number) VALUES
-> ('круассан шоколадный, латте средний', '18'),
-> ('пицца маргарита', '27'),
-> ('салат цезарь с курицей, суп грибной', '3'),
-> ('котлеты паровые с пюре', '11')
-> ;
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

Рисунок 23 – Внесение данных в таблицу order_of_client

И теперь внесем данные в третью таблицу — client (Рис. 24). Надо помнить, что значения в поле id_user и id_order должны присутствовать в таблицах user_data (данные пользователей) и order_of_client (заказ клиента) соответственно.

```
mysql> INSERT INTO client (birth_date, id_order, id_user) VALUES
-> ('10.10.1990', '1', '2'),
-> ('02.04.2001', '2', '3'),
-> ('05.09.2003', '3', '4'),
-> ('25.06.1987', '4', '1')
-> ;
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

Рисунок 24 – Внесение данных в таблицу client

Теперь попробуем внести еще одну тему, но с id_user, которого в таблице user_data нет (т.к. мы внесли в таблицу user_data только 5 пользователей, то id=6 не существует). Сервер выдает ошибку и говорит, что не может внести такую строку, т.к. в поле, являющемся внешним ключом, стоит значение, отсутствующее в связанной таблице user_data (Рис. 25).

```
mysql> INSERT INTO client (birth_date, id_order, id_user) VALUES ('13.08.2000', '4', '6');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`cafe`.`client`
, CONSTRAINT `client_ibfk_2` FOREIGN KEY (`id_user`) REFERENCES `user_data` (`id_user`))
mysql>
```

Рисунок 25 – Ошибочное внесение данных в таблицу client

Итак, в нашей БД forum есть таблицы: user_data (данные пользователей), order_of_client (заказ) и client (клиент). И мы хотим посмотреть, какие данные в них содержатся. Для этого в SQL существует оператор SELECT. Посмотрим все столбцы из таблицы user_data (Рис. 26).

```
mysql> SELECT * FROM user_data;
+-----+-----+-----+
| id_user | name   | surname |
+-----+-----+-----+
|      1 | maria  | ivanova |
|      2 | oleg   | smirnov |
|      3 | daniil | kovalyov |
|      4 | ekaterina | petrova |
|      5 | darya  | akatieva |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

Рисунок 26 – Просмотр всех столбцов таблицы user_data

Добавим оставшиеся таблицы в нашу БД (Рис. 27-28).

```

mysql> create table cafe (
  -> id_cafe int (10) AUTO_INCREMENT,
  -> name varchar(15) NOT NULL,
  -> address text NOT NULL,
  -> telephone int (11) NOT NULL,
  -> PRIMARY KEY (id_cafe)
  -> );
Query OK, 0 rows affected, 2 warnings (0.02 sec)

mysql> create table review (
  -> id_review int (10) AUTO_INCREMENT,
  -> grade int (1) NOT NULL,
  -> comment text NOT NULL,
  -> id_order int (10) NOT NULL,
  -> id_client int (10) NOT NULL,
  -> PRIMARY KEY (id_review),
  -> FOREIGN KEY (id_order) REFERENCES order_of_client (id_order),
  -> FOREIGN KEY (id_client) REFERENCES client (id_client)
  -> );
Query OK, 0 rows affected, 4 warnings (0.03 sec)

mysql> create table dish (
  -> id_dish int (10) AUTO_INCREMENT,
  -> name varchar(30) NOT NULL,
  -> category varchar(20) NOT NULL,
  -> price float NOT NULL,
  -> id_order int (10) NOT NULL,
  -> PRIMARY KEY (id_dish),
  -> FOREIGN KEY (id_order) REFERENCES order_of_client (id_order)
  -> );
Query OK, 0 rows affected, 2 warnings (0.03 sec)

mysql> create table waiter (
  -> id_waiter int (10) AUTO_INCREMENT,
  -> rating float NOT NULL,
  -> id_order int (10) NOT NULL,
  -> id_user int (10) NOT NULL,
  -> PRIMARY KEY (id_waiter),
  -> FOREIGN KEY (id_user) REFERENCES user_data (id_user),
  -> FOREIGN KEY (id_order) REFERENCES order_of_client (id_order)
  -> );
Query OK, 0 rows affected, 3 warnings (0.03 sec)

mysql> create table cook (
  -> id_cook int (10) AUTO_INCREMENT,
  -> specialization varchar(15) NOT NULL,
  -> id_user int (10) NOT NULL,
  -> PRIMARY KEY (id_cook),
  -> FOREIGN KEY (id_user) REFERENCES user_data (id_user)
  -> );
Query OK, 0 rows affected, 2 warnings (0.03 sec)

```

Рисунок 27 – Добавление оставшихся таблиц в БД

```
mysql> create table cook_has_order (  
-> id_cook int (10) NOT NULL,  
-> id_order int (10) NOT NULL,  
-> FOREIGN KEY (id_cook) REFERENCES cook (id_cook),  
-> FOREIGN KEY (id_order) REFERENCES order_of_client (id_order)  
-> );  
Query OK, 0 rows affected, 2 warnings (0.10 sec)  
  
mysql> create table cafe_has_client (  
-> id_cafe int (10) NOT NULL,  
-> id_client int (10) NOT NULL,  
-> FOREIGN KEY (id_cafe) REFERENCES cafe (id_cafe),  
-> FOREIGN KEY (id_client) REFERENCES client (id_client)  
-> );  
Query OK, 0 rows affected, 2 warnings (0.08 sec)
```

Рисунок 28 – Добавление оставшихся таблиц в БД

ПРАКТИЧЕСКАЯ РАБОТА №2

В конце предыдущей работы мы увидели все данные, которые были внесены в таблицу `user_data`. Но предположим, что нужно посмотреть только столбец `id_user`. Для этого в запросе укажем имя этого столбца:

`SELECT id_user FROM user_data;` (Рис. 29)

```
mysql> SELECT id_user FROM user_data;
+-----+
| id_user |
+-----+
|      1 |
|      2 |
|      3 |
|      4 |
|      5 |
+-----+
5 rows in set (0.00 sec)
```

Рисунок 29 – Просмотр столбца `id_user` таблицы `user_data`

Если нужно посмотреть, например, имена и фамилии наших пользователей, то перечислим интересующие столбцы через запятую:

`SELECT name, surname FROM user_data;` (Рис. 30)

```
mysql> SELECT name, surname FROM user_data;
+-----+-----+
| name   | surname |
+-----+-----+
| maria  | ivanova |
| oleg   | smirnov |
| daniil | kovalyov |
| ekaterina | petrova |
| darya  | akatieva |
+-----+-----+
5 rows in set (0.00 sec)
```

Рисунок 30 – Просмотр столбцов `name, surname` таблицы `user_data`

Аналогично, можно посмотреть, какие данные содержат и другие наши таблицы. Давайте посмотрим, какие существуют заказы:

SELECT * FROM order_of_client; (Рис. 31)

```
mysql> SELECT * FROM order_of_client;
+-----+-----+-----+
| id_order | content                                | table_number |
+-----+-----+-----+
| 1 | круассан шоколадный, латте средний | 18 |
| 2 | пицца маргарита                     | 27 |
| 3 | салат цезарь с курицей, суп грибной | 3 |
| 4 | котлеты паровые с пюре              | 11 |
+-----+-----+-----+
4 rows in set (0.04 sec)
```

Рисунок 31 – Просмотр всех столбцов таблицы order_of_client

Сейчас в таблице всего 4 заказа, а если их будет 100? Хотелось бы, чтобы они выводились, например, по алфавиту. Для этого в SQL существует ключевое слово ORDER BY после которого указывается имя столбца, по которому будет происходить сортировка. Синтаксис следующий:

SELECT имя_столбца FROM имя_таблицы ORDER BY
имя_столбца_сортировки; (Рис. 32)

```
mysql> SELECT * FROM order_of_client ORDER BY content;
+-----+-----+-----+
| id_order | content                                | table_number |
+-----+-----+-----+
| 4 | котлеты паровые с пюре              | 11 |
| 1 | круассан шоколадный, латте средний | 18 |
| 2 | пицца маргарита                     | 27 |
| 3 | салат цезарь с курицей, суп грибной | 3 |
+-----+-----+-----+
4 rows in set (0.04 sec)
```

Рисунок 32 – Сортировка по возрастанию по столбцу content

По умолчанию сортировка идет по возрастанию, но это можно изменить, добавив ключевое слово DESC (Рис. 33).

```
mysql> SELECT * FROM order_of_client ORDER BY content DESC;
```

id_order	content	table_number
3	салат цезарь с курицей, суп грибной	3
2	пицца маргарита	27
1	круассан шоколадный, латте средний	18
4	котлеты паровые с пюре	11

```
4 rows in set (0.00 sec)
```

Рисунок 33 – Сортировка по убыванию по столбцу content

Сортировку можно производить сразу по нескольким столбцам. Например, следующий запрос отсортирует данные по столбцу content, и если в этом столбце будет несколько одинаковых строк, то в столбце table_number будет осуществлена сортировка по убыванию (Рис. 34).

```
mysql> SELECT * FROM order_of_client ORDER BY content DESC, table_number DESC;
```

id_order	content	table_number
3	салат цезарь с курицей, суп грибной	3
2	пицца маргарита	27
1	круассан шоколадный, латте средний	18
4	котлеты паровые с пюре	11

```
4 rows in set (0.00 sec)
```

Рисунок 34 – Сортировка по нескольким столбцам

Очень часто бывает, что все информация из таблицы не нужна. Например, необходимо узнать, какие блюда находятся в заказе с id=4. Для этого в SQL есть ключевое слово WHERE, синтаксис у такого запроса следующий:

SELECT имя_столбца FROM имя_таблицы WHERE условие;

Для нашего примера условием является идентификатор заказа, т.е. нужны только те строки, в столбце id_order которых стоит 4:

SELECT * FROM order_of_client WHERE id_order=4; (Рис. 35)

```
mysql> SELECT * FROM order_of_client WHERE id_order=4;
+-----+-----+-----+
| id_order | content                | table_number |
+-----+-----+-----+
|         4 | котлеты паровые с пюре |          11 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Рисунок 35 – Выбор строк с id_order=4

Или нужно узнать, под каким столиком заказ «пицца маргарита» (Рис. 36).

```
mysql> SELECT * FROM order_of_client WHERE content='пицца маргарита';
+-----+-----+-----+
| id_order | content                | table_number |
+-----+-----+-----+
|         2 | пицца маргарита       |          27 |
+-----+-----+-----+
1 row in set (0.04 sec)
```

Рисунок 36 – Выбор строк с content= 'пицца маргарита'

Поиск с использованием метасимволов может осуществляться только в текстовых полях. Самый распространенный метасимвол — %. Он означает любые символы. Например, если нам надо найти слова, начинающиеся с букв «кр», то напомним LIKE 'кр%', а если хотим найти слова, которые содержат символы «са», то напомним LIKE '%са%'. Пример показан на Рисунке 37.

```
mysql> SELECT * FROM order_of_client WHERE content LIKE '%са%';
+-----+-----+-----+
| id_order | content                | table_number |
+-----+-----+-----+
|         1 | круассан шоколадный, латте средний |          18 |
|         3 | салат цезарь с курицей, суп грибной |           3 |
+-----+-----+-----+
2 rows in set (0.04 sec)
```

Рисунок 37 – Выбор строк, которые в столбце content содержат символы «са»

Еще один часто используемый метасимвол — _. В отличие от %, который обозначает несколько или ни одного символа, нижнее подчеркивание обозначает ровно один символ. Пример представлен на Рисунке 38.

```
mysql> SELECT * FROM order_of_client WHERE content LIKE '_алат%';
+-----+-----+-----+
| id_order | content | table_number |
+-----+-----+-----+
| 3 | салат цезарь с курицей, суп грибной | 3 |
+-----+-----+-----+
1 row in set (0.03 sec)
```

Рисунок 38 – Выбор строк, которые в столбце content содержат символы «алат» с одним символом перед этим

Предположим, что нашему кафе нужны роли. Для этого в таблицу user_data надо добавить столбец с ролью пользователя. Для добавления столбцов в таблицу используется оператор ALTER TABLE — ADD COLUMN. Добавим столбец role в таблицу user_data:

```
ALTER TABLE user_data ADD COLUMN role varchar(20);
```

Столбец появился в конце таблицы (Рис. 39).

```
mysql> describe user_data;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id_user | int | NO | PRI | NULL | auto_increment |
| name | varchar(15) | NO | | NULL | |
| surname | varchar(15) | NO | | NULL | |
| role | varchar(20) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.03 sec)
```

Рисунок 39 – Добавление столбца «role»

Для того, чтобы указать местоположение столбца используются ключевые слова: FIRST — новый столбец будет первым, и AFTER — указывает после какого столбца поместить новый. Давайте добавим два столбца в таблицу client: один — orders_num — количество сделанных заказов, а другой — status — статус клиента. Оба столбца вставим после поля birth_date (Рис. 40):

```
ALTER TABLE client ADD COLUMN orders_num int(10) AFTER
birth_date,
```

ADD COLUMN status varchar(20) AFTER orders_num;

```
mysql> describe client;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| id_client  | int           | NO   | PRI | NULL    | auto_increment |
| birth_date | varchar(10)   | NO   |     | NULL    |                 |
| orders_num | int           | YES  |     | NULL    |                 |
| status     | varchar(20)   | YES  |     | NULL    |                 |
| id_order   | int           | NO   | MUL | NULL    |                 |
| id_user    | int           | NO   | MUL | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Рисунок 40 – Добавление столбцов «orders_num», «status»

Теперь надо назначить статус «премиум» какому-нибудь клиенту. Для обновления уже существующих данных служит оператор UPDATE. Изменять данные можно и сразу в нескольких строках, и во всей таблице. Например, решили давать статус в зависимости от количества сделанных клиентом заказов. Давайте в нашу таблицу сначала внесем значения столбца orders_num (Рис. 41).

```
mysql> UPDATE client SET orders_num=10 WHERE id_client=1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE client SET orders_num=45 WHERE id_client=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE client SET orders_num=52 WHERE id_client=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> UPDATE client SET orders_num=68 WHERE id_client=4;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM client;
+-----+-----+-----+-----+-----+-----+
| id_client | birth_date | orders_num | status | id_order | id_user |
+-----+-----+-----+-----+-----+-----+
| 1 | 10.10.1990 | 10 | NULL | 1 | 2 |
| 2 | 02.04.2001 | 45 | NULL | 2 | 3 |
| 3 | 05.09.2003 | 52 | NULL | 3 | 4 |
| 4 | 25.06.1987 | 68 | NULL | 4 | 1 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

Рисунок 41 – Обновление значений столбца «orders_num»

А теперь зададим статус «премиум» тем, у кого количество заказов больше 50 (Рис. 42).

```
mysql> UPDATE client SET status='премиум' WHERE orders_num>50;
Query OK, 2 rows affected (0.01 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT * FROM client;
+-----+-----+-----+-----+-----+-----+
| id_client | birth_date | orders_num | status | id_order | id_user |
+-----+-----+-----+-----+-----+-----+
| 1 | 10.10.1990 | 10 | NULL | 1 | 2 |
| 2 | 02.04.2001 | 45 | NULL | 2 | 3 |
| 3 | 05.09.2003 | 52 | премиум | 3 | 4 |
| 4 | 25.06.1987 | 68 | премиум | 4 | 1 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Рисунок 42 – Обновление значений столбца «status»

Данные изменились в двух строках, согласно заданному условию. Понятно, что если в запросе опустить условие, то данные будут обновлены во всех строках таблицы.

Предположим, что не нравится название Статус у нашего столбца, надо переименовать столбец в Текущий статус — current_status. Для изменения имени существующего столбца используется оператор CHANGE. Давайте поменяем status на current_status (Рис. 43).

```
mysql> ALTER TABLE client CHANGE status current_status varchar(20);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM client;
+-----+-----+-----+-----+-----+-----+
| id_client | birth_date | orders_num | current_status | id_order | id_user |
+-----+-----+-----+-----+-----+-----+
| 1 | 10.10.1990 | 10 | NULL | 1 | 2 |
| 2 | 02.04.2001 | 45 | NULL | 2 | 3 |
| 3 | 05.09.2003 | 52 | премиум | 3 | 4 |
| 4 | 25.06.1987 | 68 | премиум | 4 | 1 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Рисунок 43 – Замена имени столбца «status» на «current_status»

Рассмотрим оператор DELETE, который позволяет удалять строки из таблицы. Давайте из таблицы отзывов удалим те отзывы, в которых оценка «3» (Рис. 44).

```
mysql> SELECT * FROM review;
+-----+-----+-----+-----+-----+
| id_review | grade | comment                                | id_order | id_client |
+-----+-----+-----+-----+-----+
| 1         | 4     | вкусно, но долго готовили             | 1        | 1         |
| 2         | 3     | очень долгое ожидание заказа          | 2        | 2         |
| 3         | 5     | отличное обслуживание!               | 3        | 3         |
| 4         | 3     | принесли не те блюда, ждал свои полчаса | 4        | 4         |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DELETE FROM review WHERE grade='3';
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM review;
+-----+-----+-----+-----+-----+
| id_review | grade | comment                                | id_order | id_client |
+-----+-----+-----+-----+-----+
| 1         | 4     | вкусно, но долго готовили             | 1        | 1         |
| 3         | 5     | отличное обслуживание!               | 3        | 3         |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Рисунок 44 – Удаление строк по условию

ПРАКТИЧЕСКАЯ РАБОТА №3

Добавим в базу данных две процедуры и протестируем их (Рис. 45-46).

Первая процедура выводит сообщение о том, имеет ли клиент статус «Премиум», в зависимости от общего количества сделанных им заказов.

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `status_check`(IN orders_num INT, OUT status_info TEXT)
2 BEGIN
3     IF orders_num > 50 THEN
4         SET status_info = "клиент имеет статус премиум";
5     ELSE
6         SET status_info = "клиент не имеет статус премиум";
7     END IF;
8 END
```

Рисунок 45 – Первая процедура

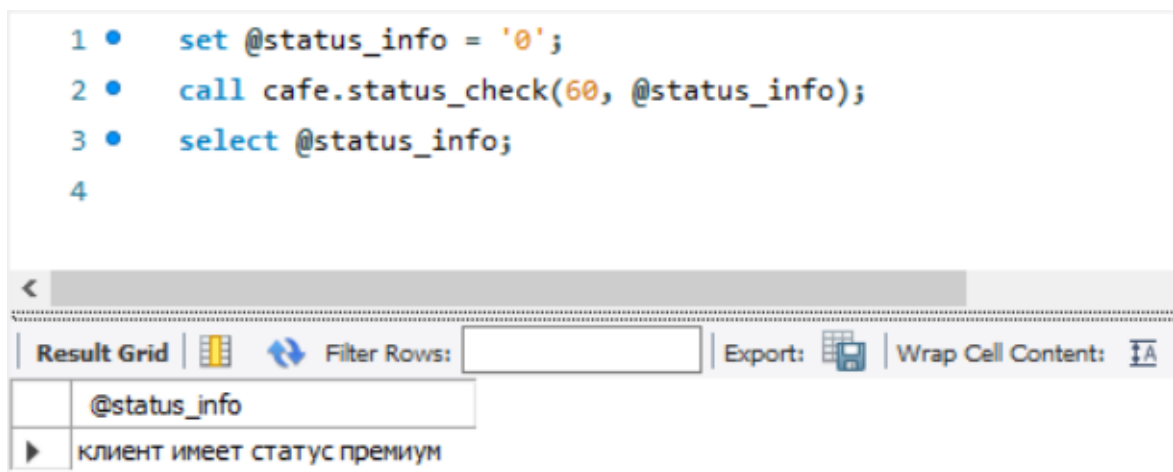


Рисунок 46 – Тестирование первой процедуры

Вторая процедура считает стоимость заказа, учитывая статус клиента. Если клиент «Премиум», то к сумме применяется скидка 20%.

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `price_counting`(IN dish_num INT, IN client_status BOOLEAN, OUT price INT)
2 BEGIN
3     IF client_status = 0 THEN
4         SET price = dish_num*300;
5     ELSE
6         SET price = dish_num*300*0.8;
7     END IF;
8 END
```

Рисунок 47 – Вторая процедура

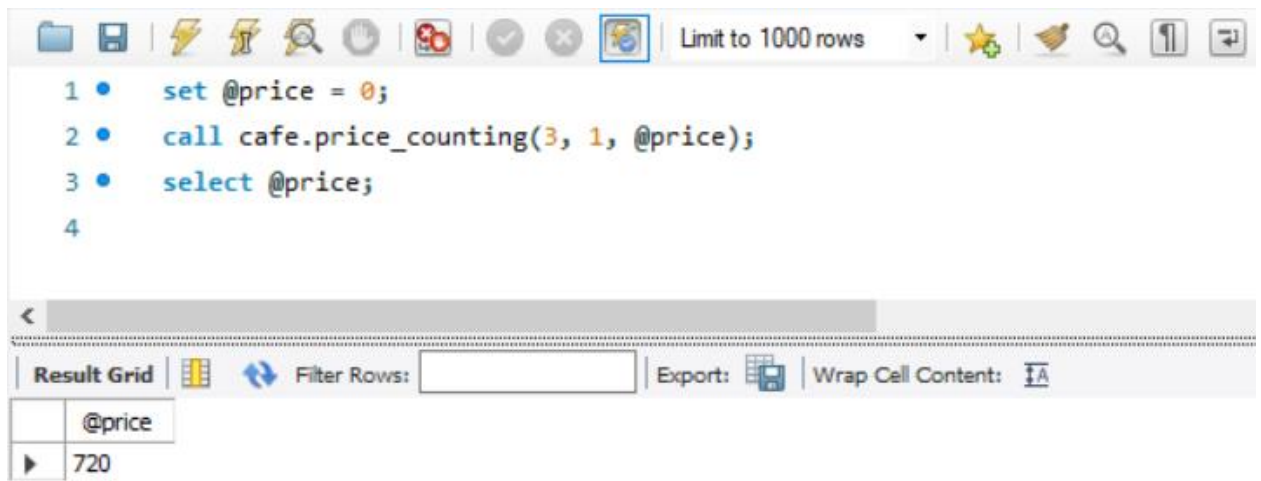


Рисунок 48 – Тестирование второй процедуры

Теперь добавим в базу данных две функции и протестируем их (Рис. 49-50).

Первая функция считает, сколько клиенту лет, по его дате рождения.

```

1 • CREATE DEFINER=`root`@`localhost` FUNCTION `years_counting`(date1 date) RETURNS int
2     DETERMINISTIC
3     BEGIN
4     DECLARE date2 date;
5     SELECT current_date() INTO date2;
6     RETURN year(date2)-year(date1);
7     END

```

Рисунок 49 – Первая функция

```

1 • select id_client, birth_date, years_counting(birth_date) as 'years' from client;
2

```

	id_client	birth_date	years
▶	1	1990-10-10	33
	2	2001-04-02	22
	3	2003-09-05	20
	4	1987-06-25	36

Рисунок 50 – Тестирование первой функции

Вторая функция считает скидку у клиента в зависимости от сделанных заказов.

```
1 • CREATE DEFINER='root'@'localhost' FUNCTION `discount_counting`(orders_num INT) RETURNS text CHARSET utf8mb4
2     DETERMINISTIC
3 BEGIN
4     IF orders_num < 5 THEN
5         RETURN "скидки нет";
6     ELSEIF orders_num >= 5 AND orders_num < 15 THEN
7         RETURN "скидка составляет 5%";
8     ELSEIF orders_num >= 15 AND orders_num < 30 THEN
9         RETURN "скидка составляет 7%";
10    ELSEIF orders_num >= 30 AND orders_num <= 50 THEN
11        RETURN "скидка составляет 10%";
12    ELSEIF orders_num > 50 THEN
13        RETURN "скидка составляет 20%";
14    END IF;
15 END
```

Рисунок 51 – Вторая функция

```
1 • select id_client, orders_num, discount_counting(orders_num) as 'discount_info' from client;
2
```

	id_client	orders_num	discount_info
1	1	10	скидка составляет 5%
2	2	45	скидка составляет 10%
3	3	52	скидка составляет 20%
4	4	68	скидка составляет 20%

Рисунок 52 – Тестирование второй функции

Теперь добавим триггеры в базу данных.

Первый триггер не даст добавить отзыв на заказ с оценкой, не входящей в интервал от 0 до 5 (Рис. 53).

```
mysql> DELIMITER //
mysql> CREATE TRIGGER grade_control
-> BEFORE INSERT ON review
-> FOR EACH ROW
-> BEGIN
-> IF NEW.grade < 0 OR NEW.grade > 5 THEN
-> SIGNAL SQLSTATE '45000'
-> SET MESSAGE_TEXT = 'ваша оценка должна быть от 0 до 5';
-> END IF;
-> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> INSERT INTO review VALUES ('2', '8', 'отлично!', '2', '2');
ERROR 1644 (45000): ваша оценка должна быть от 0 до 5
mysql> SELECT * FROM review;
+-----+-----+-----+-----+-----+
| id_review | grade | comment                | id_order | id_client |
+-----+-----+-----+-----+-----+
|         1 |     4 | вкусно, но долго готовили |         1 |         1 |
|         3 |     5 | отличное обслуживание!   |         3 |         3 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Рисунок 53 – Первый триггер

Второй триггер не даст добавить клиента со статусом "премиум", если количество заказов этого клиента меньше 50 (Рис. 54).

```
mysql> CREATE TRIGGER status_control
-> BEFORE INSERT ON client
-> FOR EACH ROW
-> BEGIN
-> IF NEW.orders_num <= 50 AND NEW.current_status='премиум' THEN
-> SIGNAL SQLSTATE '45000'
-> SET MESSAGE_TEXT = 'для получения статуса премиум клиенту необходимо сделать более 50 заказов';
-> END IF;
-> END //
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO client VALUES ('5', '2002-05-10', '15', 'премиум', '5', '5')//
ERROR 1644 (45000): для получения статуса премиум клиенту необходимо сделать более 50 заказов
```

Рисунок 54 – Второй триггер

Третий триггер удаляет данные о поваре из таблицы cook_has_order перед удалением повара из таблицы cook (Рис. 55).

```

mysql> SELECT * FROM cook;
+-----+-----+-----+
| id_cook | specialization | id_user |
+-----+-----+-----+
|      1 | кондитер      |      6 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM cook_has_order;
+-----+-----+
| cook_id_cook | order_id_order |
+-----+-----+
|           1 |           2 |
+-----+-----+
1 row in set (0.00 sec)

mysql> DELIMITER //
mysql> CREATE TRIGGER before_cookhasorder_delete
-> BEFORE DELETE ON cook
-> FOR EACH ROW
-> BEGIN
-> DELETE FROM cook_has_order WHERE cook_id_cook = old.id_cook;
-> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELETE FROM cook WHERE id_cook = '1';
-> //
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM cook;
-> //
Empty set (0.00 sec)

mysql> SELECT * FROM cook_has_order//
Empty set (0.00 sec)

```

Рисунок 55 – Третий триггер

Четвёртый триггер удаляет данные из таблиц cook, waiter, client перед удалением из user_data (Рис. 56-57).

```
mysql> SELECT * FROM user_data;
```

id_user	name	surname	role
1	maria	ivanova	NULL
2	oleg	smirnov	NULL
3	daniil	kovalyov	NULL
4	ekaterina	petrova	NULL
5	darya	akatieva	NULL
6	ivan	markov	повар
7	sergey	fonin	официант
8	alina	timofeeva	клиент

```
8 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM client;
```

id_client	birth_date	orders_num	current_status	id_order	id_user
1	1990-10-10	10	NULL	1	2
2	2001-04-02	45	NULL	2	3
3	2003-09-05	52	премиум	3	4
4	1987-06-25	68	премиум	4	1
5	2000-07-07	57	премиум	5	8

```
5 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM waiter;
```

id_waiter	rating	id_order	id_user
1	4.5	3	7

```
1 row in set (0.00 sec)
```



```
mysql> SELECT * FROM cook;
```

id_cook	specialization	id_user
1	кондитер	6

```
1 row in set (0.00 sec)
```

Рисунок 56 – Четвёртый триггер (ч. 1)

```
mysql> DELETE FROM user_data WHERE id_user = '6';
Query OK, 1 row affected (0.01 sec)

mysql> DELETE FROM user_data WHERE id_user = '7';
Query OK, 1 row affected (0.01 sec)

mysql> DELETE FROM user_data WHERE id_user = '8';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM user_data;
+-----+-----+-----+-----+
| id_user | name   | surname | role |
+-----+-----+-----+-----+
| 1      | maria  | ivanova | NULL |
| 2      | oleg   | smirnov | NULL |
| 3      | daniil | kovalyov | NULL |
| 4      | ekaterina | petrova | NULL |
| 5      | darya  | akatieva | NULL |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM client;
+-----+-----+-----+-----+-----+-----+
| id_client | birth_date | orders_num | current_status | id_order | id_user |
+-----+-----+-----+-----+-----+-----+
| 1        | 1990-10-10 | 10        | NULL          | 1        | 2       |
| 2        | 2001-04-02 | 45        | NULL          | 2        | 3       |
| 3        | 2003-09-05 | 52        | премиум      | 3        | 4       |
| 4        | 1987-06-25 | 68        | премиум      | 4        | 1       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM waiter;
Empty set (0.00 sec)

mysql> SELECT * FROM cook;
Empty set (0.00 sec)
```

Рисунок 57 – Четвёртый триггер (ч. 2)

Пятый триггер обновляет статус повара в таблице cook, если в таблицу cook_has_order добавляется запись о заказе для этого повара (Рис. 58-59).

```
mysql> ALTER TABLE cook ADD COLUMN cook_status varchar(20);
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM cook;
+-----+-----+-----+-----+
| id_cook | specialization | id_user | cook_status |
+-----+-----+-----+-----+
| 1      | кондитер      | 6      | NULL       |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Рисунок 58 – Пятый триггер (ч. 1)

```

mysql> DELIMITER //
mysql> CREATE TRIGGER cook_status_update
-> BEFORE INSERT ON cook_has_order
-> FOR EACH ROW
-> BEGIN
-> UPDATE cook
-> SET cook_status = 'занят заказом'
-> WHERE id_cook = NEW.cook_id_cook;
-> END //
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO cook_has_order VALUES ('1', '5');
-> //
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM cook;
-> //
+-----+-----+-----+-----+
| id_cook | specialization | id_user | cook_status |
+-----+-----+-----+-----+
|      1 | кондитер      |      6 | занят заказом |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

Рисунок 59 – Пятый триггер (ч. 2)

ПРАКТИЧЕСКАЯ РАБОТА №4

Оконные функции в MySQL — это специальный тип функции, который позволяет выполнять агрегатные и аналитические операции над группами строк, которые определены внутри отдельного окна (или оконного фрейма). Оконные функции представляют собой удобный инструмент по работы с данными и предоставляют более гибкий способ обращения с ними, чем традиционные агрегатные функции за счёт того, что они могут учитывать порядок сортировки данных и разбивать их на группы без фактической группировки.

Рассмотрим оконную функцию OVER(). С помощью этой функции найдём среднее значение количества заказов среди всех клиентов (Рис. 60).

```
mysql> SELECT
-> id_client,
-> birth_date,
-> orders_num,
-> current_status,
-> id_order,
-> id_user,
-> AVG(orders_num) OVER () AS 'Avg'
-> FROM client;
```

id_client	birth_date	orders_num	current_status	id_order	id_user	Avg
1	1990-10-10	10	NULL	1	2	43.7500
2	2001-04-02	45	NULL	2	3	43.7500
3	2003-09-05	52	премиум	3	4	43.7500
4	1987-06-25	68	премиум	4	1	43.7500

4 rows in set (0.01 sec)

Рисунок 60 – Функция OVER()

Применим инструкцию PARTITION BY(), чтобы получить таблицу, сгруппированную по столбцу current_status (Рис. 61). Теперь для каждой из двух групп (NULL и премиум) рассчитывается своё среднее значение.


```
mysql> SELECT
-> current_status,
-> id_client,
-> birth_date,
-> orders_num,
-> id_order,
-> id_user,
-> AVG(orders_num) OVER (PARTITION BY current_status) AS 'Avg'
-> FROM client;
```

current_status	id_client	birth_date	orders_num	id_order	id_user	Avg
NULL	1	1990-10-10	10	1	2	27.5000
NULL	2	2001-04-02	45	2	3	27.5000
премиум	3	2003-09-05	52	3	4	60.0000
премиум	4	1987-06-25	68	4	1	60.0000

4 rows in set (0.01 sec)

Рисунок 61 – Функция PARTITION BY()

В рамках оконной функции ORDER BY выполняет сортировку, но имеет несколько иное назначение. Для более наглядной демонстрации заменим функцию AVG на SUM и попробуем применить ORDER BY для разделов, определённых ранее (Рис. 62). При помощи ORDER BY мы указали, что хотим видеть не просто сумму всех значений в окне, а для каждого значения количества заказов сумму с предыдущими согласно сортировке. Таким образом, последнее значение столбца Sum является итоговым значением суммы количества заказов в данном разделе, а каждое предшествующее ему – значение данной строки и предыдущих, относящихся к этому разделу.

```
mysql> SELECT
-> current_status,
-> id_client,
-> birth_date,
-> orders_num,
-> id_order,
-> id_user,
-> SUM(orders_num) OVER (PARTITION BY current_status ORDER BY id_client) AS 'Sum'
-> FROM client;
```

current_status	id_client	birth_date	orders_num	id_order	id_user	Sum
NULL	1	1990-10-10	10	1	2	10
NULL	2	2001-04-02	45	2	3	55
премиум	3	2003-09-05	52	3	4	52
премиум	4	1987-06-25	68	4	1	120

4 rows in set (0.01 sec)

Рисунок 62 – Функция ORDER BY

Инструкция ROWS позволяет ограничить строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей.

Инструкция RANGE же, в свою очередь, работает не со строками, а с диапазоном строк в инструкции ORDER BY. То есть в RANGE одной строкой могут считаться несколько реальных строк, одинаковых по рангу.

Рассмотрим пример применения нескольких из описанных выше инструкций (Рис. 63). В данном случае сумма будет рассчитываться по текущей и следующей ячейке в окне. Последнее значение окна будет иметь то же значение, что и столбец количества заказов, так как больше не с чем складывать.

```
mysql> SELECT
-> current_status,
-> id_client,
-> birth_date,
-> orders_num,
-> id_order,
-> id_user,
-> SUM(orders_num) OVER (PARTITION BY current_status ORDER BY id_client ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS 'Sum'
-> FROM client;
```

current_status	id_client	birth_date	orders_num	id_order	id_user	Sum
NULL	1	1990-10-10	10	1	2	55
NULL	2	2001-04-02	45	2	3	45
премиум	3	2003-09-05	52	3	4	120
премиум	4	1987-06-25	68	4	1	68

4 rows in set (0.00 sec)

Рисунок 63 – Применение инструкции ROWS с инструкциями CURRENT ROW и FOLLOWING

Агрегатные оконные функции

Агрегатными функциями называются функции, которые выполняют арифметические вычисления на наборе данных и возвращают итоговое значение.

COUNT(). Результатом использования функции в данном случае является id заказа, содержание заказа, номер столика и количество заказов, которые сделаны на каждый столик (Рис. 64).

```
mysql> SELECT
-> id_order,
-> content,
-> table_number,
-> COUNT(*) OVER (PARTITION BY table_number) AS table_orders_count
-> FROM order_of_client;
```

id_order	content	table_number	table_orders_count
3	салат цезарь с курицей, суп грибной	3	1
4	котлеты паровые с пюре	11	2
6	коктейль молочный клубничный	11	2
5	пицца пепперони	12	1
1	круассан шоколадный, латте средний	18	1
2	пицца маргарита	27	2
7	сок апельсиновый, пирог осетинский с сыром и зеленью	27	2

7 rows in set (0.00 sec)

Рисунок 64 – Функция COUNT()

SUM(). Посчитаем общую сумму блюд в каждой категории (Рис. 65).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> SUM(price) OVER (PARTITION BY category) AS price_category
-> FROM dish;
```

id_dish	name	category	price	price_category
6	котлеты паровые с пюре	вторые блюда	350	350
1	круассан шоколадный	выпечка	200	200
10	пирог с сыром и зеленью	закуски	400	400
2	латте средний	напитки	200	750
8	коктейль молочный клубничный	напитки	300	750
9	сок апельсиновый	напитки	250	750
3	пицца маргарита	пицца	500	1000
7	пицца пепперони	пицца	500	1000
4	салат цезарь с курицей	салаты	350	350
5	суп грибной	супы	300	300

10 rows in set (0.01 sec)

Рисунок 65 – Функция SUM()

AVG(). Посчитаем среднее значение суммы всех блюд в каждой категории (Рис. 66).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> AVG(price) OVER (PARTITION BY category) AS avg_price_category
-> FROM dish;
```

id_dish	name	category	price	avg_price_category
6	котлеты паровые с пюре	вторые блюда	350	350
1	круассан шоколадный	выпечка	200	200
10	пирог с сыром и зеленью	закуски	400	400
2	латте средний	напитки	200	250
8	коктейль молочный клубничный	напитки	300	250
9	сок апельсиновый	напитки	250	250
3	пицца маргарита	пицца	500	500
7	пицца пепперони	пицца	500	500
4	салат цезарь с курицей	салаты	350	350
5	суп грибной	супы	300	300

10 rows in set (0.00 sec)

Рисунок 66 – Функция AVG()

MIN()). Посчитаем минимальную цену блюда в каждой категории (Рис. 67).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> MIN(price) OVER (PARTITION BY category) AS min_price_category
-> FROM dish;
```

id_dish	name	category	price	min_price_category
6	котлеты паровые с пюре	вторые блюда	350	350
1	круассан шоколадный	выпечка	200	200
10	пирог с сыром и зеленью	закуски	400	400
2	латте средний	напитки	200	200
8	коктейль молочный клубничный	напитки	300	200
9	сок апельсиновый	напитки	250	200
3	пицца маргарита	пицца	500	500
7	пицца пепперони	пицца	500	500
4	салат цезарь с курицей	салаты	350	350
5	суп грибной	супы	300	300

10 rows in set (0.01 sec)

Рисунок 67 – Функция MIN()

MAX()). Посчитаем максимальную цену блюда в каждой категории (Рис. 68).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> MAX(price) OVER (PARTITION BY category) AS max_price_category
-> FROM dish;
```

id_dish	name	category	price	max_price_category
6	котлеты паровые с пюре	вторые блюда	350	350
1	круассан шоколадный	выпечка	200	200
10	пирог с сыром и зеленью	закуски	400	400
2	латте средний	напитки	200	300
8	коктейль молочный клубничный	напитки	300	300
9	сок апельсиновый	напитки	250	300
3	пицца маргарита	пицца	500	500
7	пицца пепперони	пицца	500	500
4	салат цезарь с курицей	салаты	350	350
5	суп грибной	супы	300	300

10 rows in set (0.00 sec)

Рисунок 68 – Функция MAX()

Ранжирующие функции

Ранжирующие функции — это функции, которые определяют ранг для каждой строки в окне.

ROW_NUMBER(). В данном примере функция возвращает номер строки, который присвоен блюду в его категории (Рис. 69).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> ROW_NUMBER() OVER (PARTITION BY category ORDER BY id_dish) AS rownum
-> FROM dish;
```

id_dish	name	category	price	rownum
6	котлеты паровые с пюре	вторые блюда	350	1
1	круассан шоколадный	выпечка	200	1
10	пирог с сыром и зеленью	закуски	400	1
2	латте средний	напитки	200	1
8	коктейль молочный клубничный	напитки	300	2
9	сок апельсиновый	напитки	250	3
3	пицца маргарита	пицца	500	1
7	пицца пепперони	пицца	500	2
4	салат цезарь с курицей	салаты	350	1
5	суп грибной	супы	300	1

10 rows in set (0.01 sec)

Рисунок 69 – Функция ROW_NUMBER()

RANK(). Функция возвращает ранг каждого блюда в своей категории по величине его цены (Рис. 70).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> RANK() OVER (PARTITION BY category ORDER BY price DESC) AS pricerank
-> FROM dish;
```

id_dish	name	category	price	pricerank
6	котлеты паровые с пюре	вторые блюда	350	1
1	круассан шоколадный	выпечка	200	1
10	пирог с сыром и зеленью	закуски	400	1
8	коктейль молочный клубничный	напитки	300	1
9	сок апельсиновый	напитки	250	2
2	латте средний	напитки	200	3
3	пицца маргарита	пицца	500	1
7	пицца пепперони	пицца	500	1
4	салат цезарь с курицей	салаты	350	1
5	суп грибной	супы	300	1

10 rows in set (0.01 sec)

Рисунок 70 – Функция RANK()

DENSE_RANK(). Функция определяет ранг блюда по его цене независимо от категории блюда (Рис. 71).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> DENSE_RANK() OVER (ORDER BY price DESC) AS denserank
-> FROM dish;
```

id_dish	name	category	price	denserank
3	пицца маргарита	пицца	500	1
7	пицца пепперони	пицца	500	1
10	пирог с сыром и зеленью	закуски	400	2
4	салат цезарь с курицей	салаты	350	3
6	котлеты паровые с пюре	вторые блюда	350	3
5	суп грибной	супы	300	4
8	коктейль молочный клубничный	напитки	300	4
9	сок апельсиновый	напитки	250	5
1	круассан шоколадный	выпечка	200	6
2	латте средний	напитки	200	6

10 rows in set (0.00 sec)

Рисунок 71 – Функция DENSE_RANK()

NTILE(*). В результате запроса мы получаем отсортированную таблицу, которая разделена на 4 примерно равные ценовые группы блюд (Рис. 72).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> NTILE(4) OVER (ORDER BY price) AS pricegroup
-> FROM dish;
```

id_dish	name	category	price	pricegroup
1	круассан шоколадный	выпечка	200	1
2	латте средний	напитки	200	1
9	сок апельсиновый	напитки	250	1
5	суп грибной	супы	300	2
8	коктейль молочный клубничный	напитки	300	2
4	салат цезарь с курицей	салаты	350	2
6	котлеты паровые с пюре	вторые блюда	350	3
10	пирог с сыром и зеленью	закуски	400	3
3	пицца маргарита	пицца	500	4
7	пицца пепперони	пицца	500	4

10 rows in set (0.00 sec)

Рисунок 72 – Функция NTILE(*)

Функции смещения

Функции смещения — это функции, которые позволяют перемещаться и обращаться к разным строкам в окне относительно текущей строки, а также обращаться к значениям в начале или в конце окна.

LAG(*) и LEAD(*). С помощью функции LAG(*) получаем предыдущее блюдо в этом же заказе (Рис. 73).

```
mysql> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> id_order,
-> LAG(name) OVER (PARTITION BY id_order ORDER BY id_dish) AS previousdish
-> FROM dish;
```

id_dish	name	category	price	id_order	previousdish
1	круассан шоколадный	выпечка	200	1	NULL
2	латте средний	напитки	200	1	круассан шоколадный
3	пицца маргарита	пицца	500	2	NULL
4	салат цезарь с курицей	салаты	350	3	NULL
5	суп грибной	супы	300	3	салат цезарь с курицей
6	котлеты паровые с пюре	вторые блюда	350	4	NULL
7	пицца пепперони	пицца	500	5	NULL
8	коктейль молочный клубничный	напитки	300	6	NULL
9	сок апельсиновый	напитки	250	7	NULL
10	пирог с сыром и зеленью	закуски	400	7	сок апельсиновый

10 rows in set (0.01 sec)

Рисунок 73 – Функция LAG(*)

FIRST_VALUE() и LAST_VALUE(). В данном примере функция FIRST_VALUE() возвращает название первого блюда в каждой категории (Рис. 74).

```
mysql> WITH firstDish AS (
-> SELECT
-> id_dish,
-> name,
-> category,
-> price,
-> FIRST_VALUE(name) OVER (PARTITION BY category ORDER BY id_dish) AS firstInCategory
-> FROM dish
-> )
-> SELECT
-> id_dish,
-> name,
-> category,
-> price
-> FROM firstDish
-> WHERE firstInCategory = name;
```

id_dish	name	category	price
6	котлеты паровые с пюре	вторые блюда	350
1	круассан шоколадный	выпечка	200
10	пирог с сыром и зеленью	закуски	400
2	латте средний	напитки	200
3	пицца маргарита	пицца	500
4	салат цезарь с курицей	салаты	350
5	суп грибной	супы	300

7 rows in set (0.02 sec)

Рисунок 74 – Функция FIRST_VALUE()

ЗАКЛЮЧЕНИЕ

В результате проведенного исследования было выполнено описание предметной области работы кафе. Для дальнейшей разработки были использованы различные методы моделирования, такие как структурно-функциональный метод, объектно-ориентированный метод, инфологическое проектирование базы данных. После этого была разработана логическая и физическая модели базы данных.

Также получены навыки работы с MySQL, создана база данных кафе, в которой используются процедуры, функции и триггеры, а также изучены оконные функции, их виды и способы применения. В ходе изучения языка SQL были освоены ключевые аспекты и принципы работы с базами данных. Изучение SQL дало полное представление о структуре баз данных, их ключевых компонентах и взаимосвязях между таблицами, что расширило возможности для формирования сложных запросов и получения необходимой информации.