|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра практической и прикладной информатики (ППИ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «разработка баз данных»

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИКБО-07-21, Хасанбаев И.А.* | (подпись) | |
| Преподаватель | *Чучаева С.М.* | (подпись) | |
| Отчет представлен | «\_\_»\_\_\_\_\_\_2023г. | |  | |

Москва 2023 г.

**Содержание**

[**Часть 1: Создание баз данных.** 3](#_Toc153231670)

[**Часть 2: Процедуры, функции и триггеры** 7](#_Toc153231671)

[**Часть 3: Приложение с графическим интерфейсом** 11](#_Toc153231672)

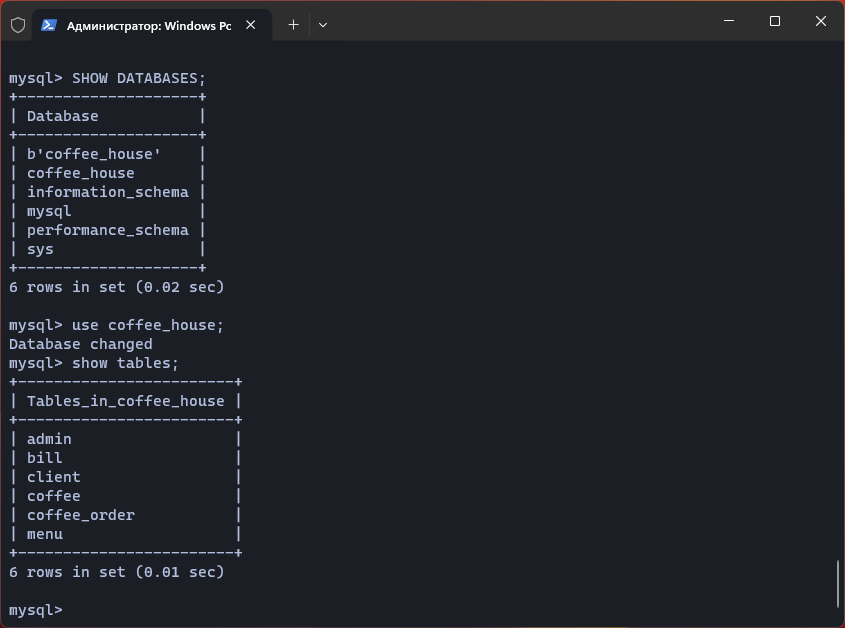
[**Часть 4: Оконные функции** 15](#_Toc153231673)

[**ВЫВОД** 18](#_Toc153231674)

[**ЛИТЕРАТУРНЫЕ ИСПТОЧНИКИ** 19](#_Toc153231675)

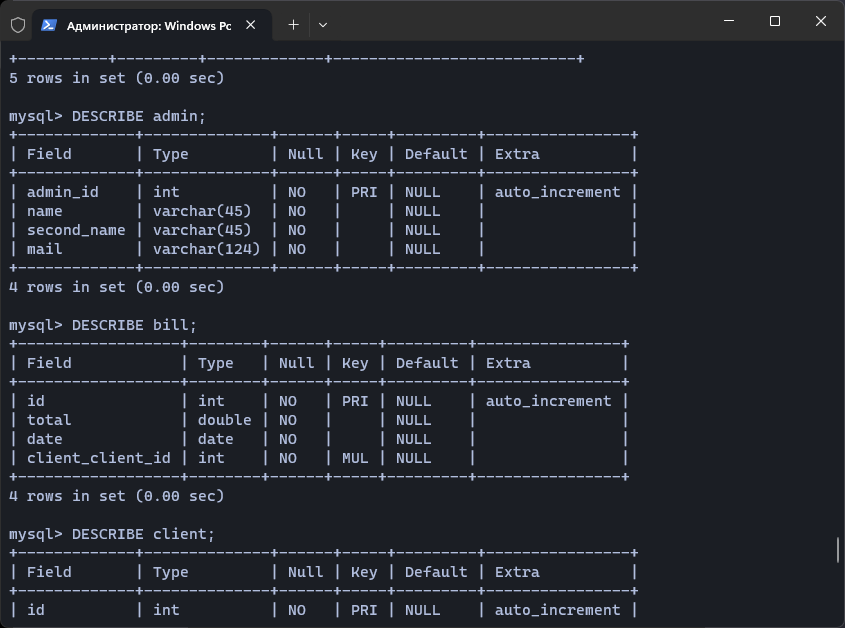
1. **Создание базы данных.**

В ходе выполнения практической работы, была создана база данных, а также таблицы для нее (рисунок 1)

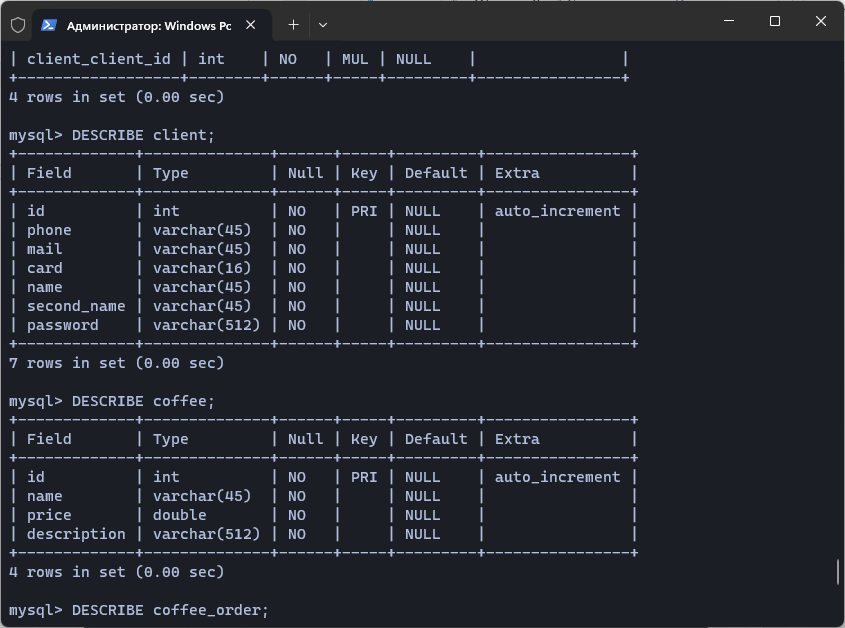


1. созданная база данных и таблицы

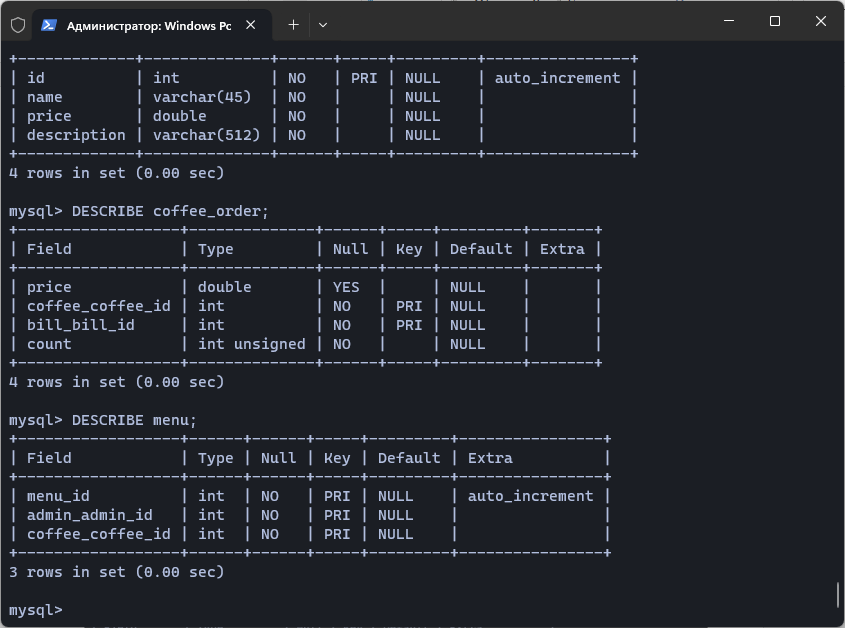
У каждой созданной таблицы, были заданы свои поля (рисунок 2-4).



1. созданные таблицы

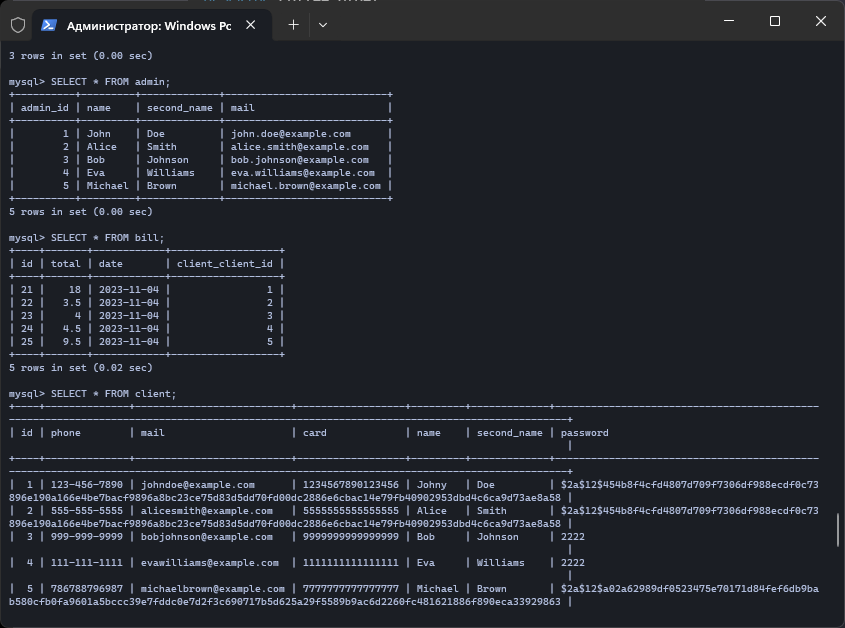


1. созданные таблицы

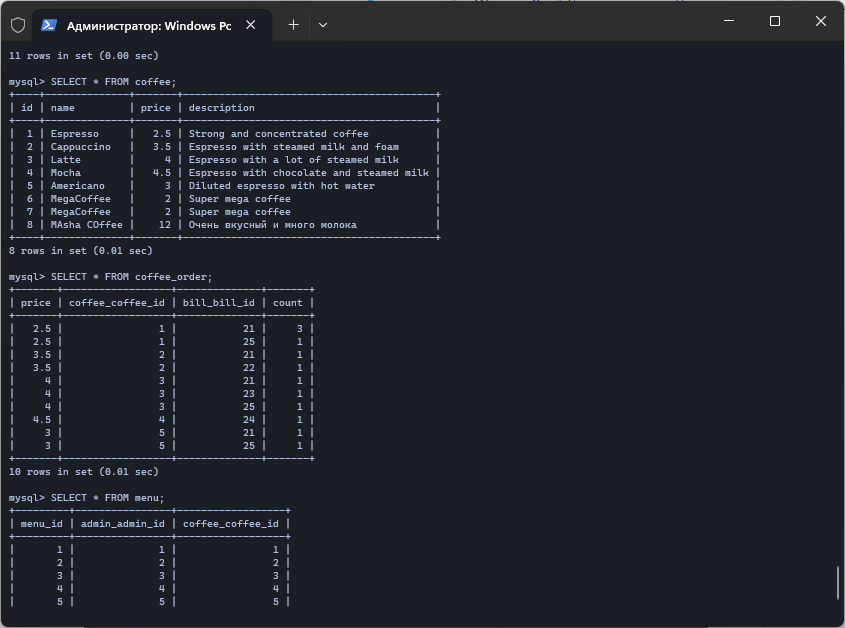


1. созданные таблицы

Далее, в рамках практической работы, каждая таблицы была заполнена данными (рисунок 5-6)

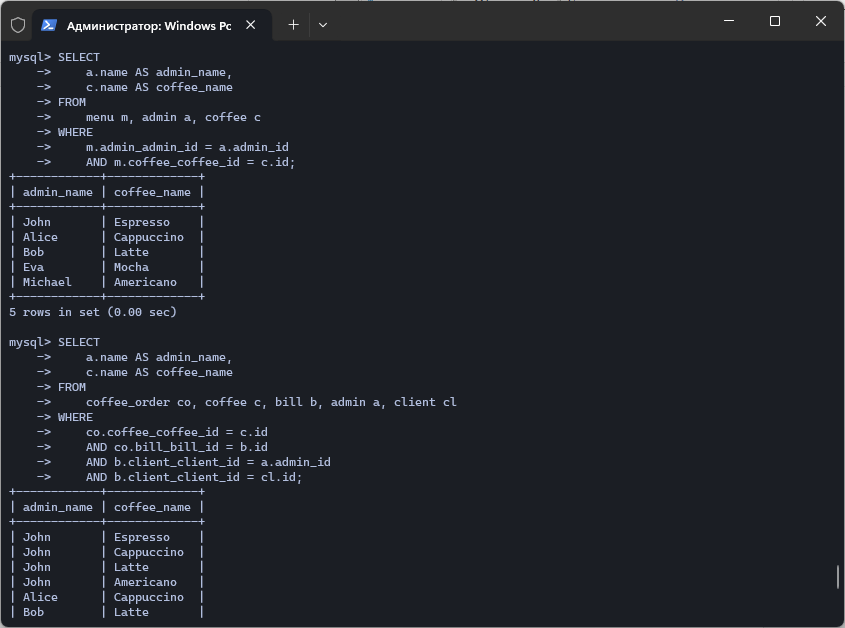


1. заполненные таблицы

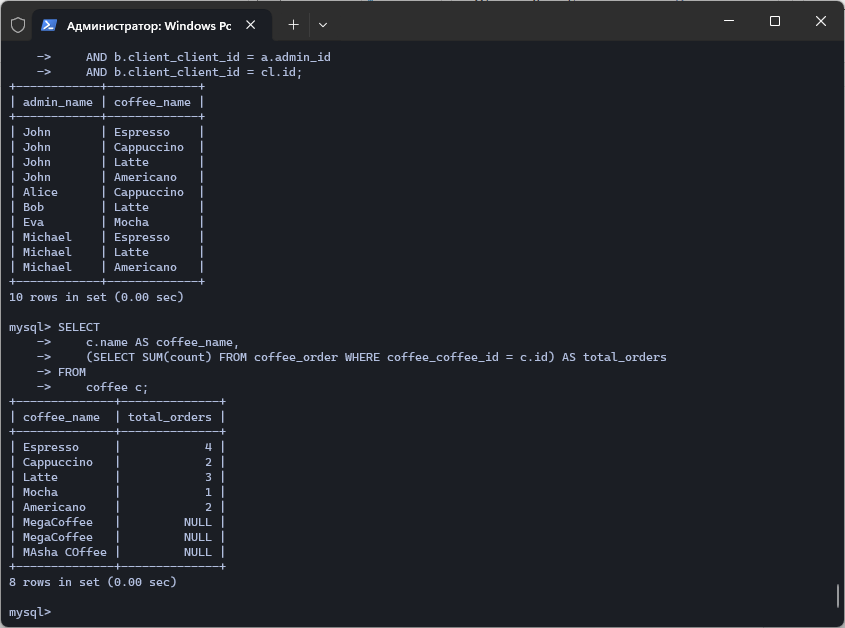


1. заполненные таблицы
2. **Создание запросов**

Далее, было написано три запроса с использованием команды select. Первый запрос выводит имя администратора и кофе, которое он добавил, второй показывает, какой клиент какое кофе заказывал, и третий запрос показывает, сколько раз было заказано каждое кофе (рисунок 7-8)

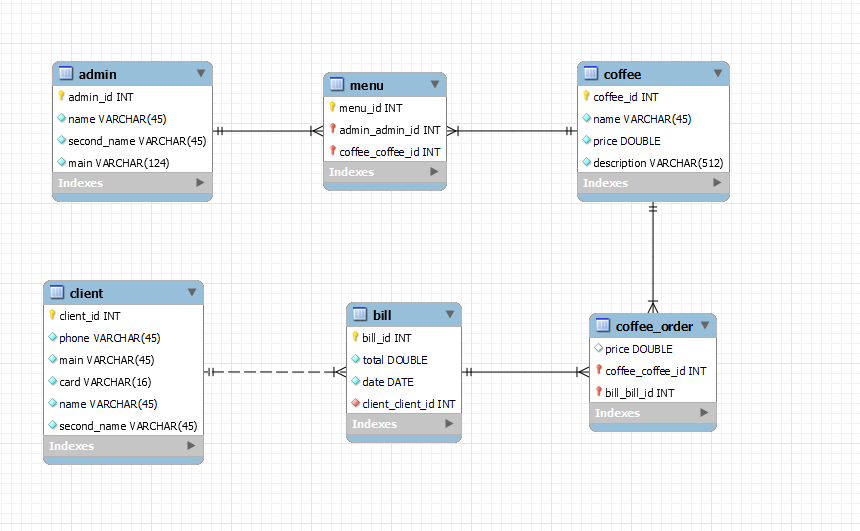


1. выполнение запросов



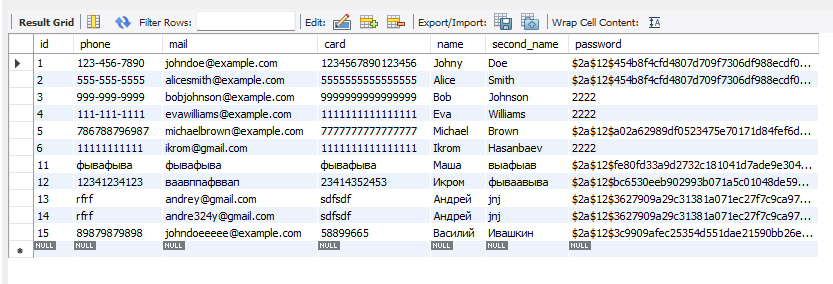
1. выполнение запросов
2. **Процедуры, функции и триггеры**

В ходе выполнения практической работы, была создана схема базы данных (рисунок 9) в mysql workbech.



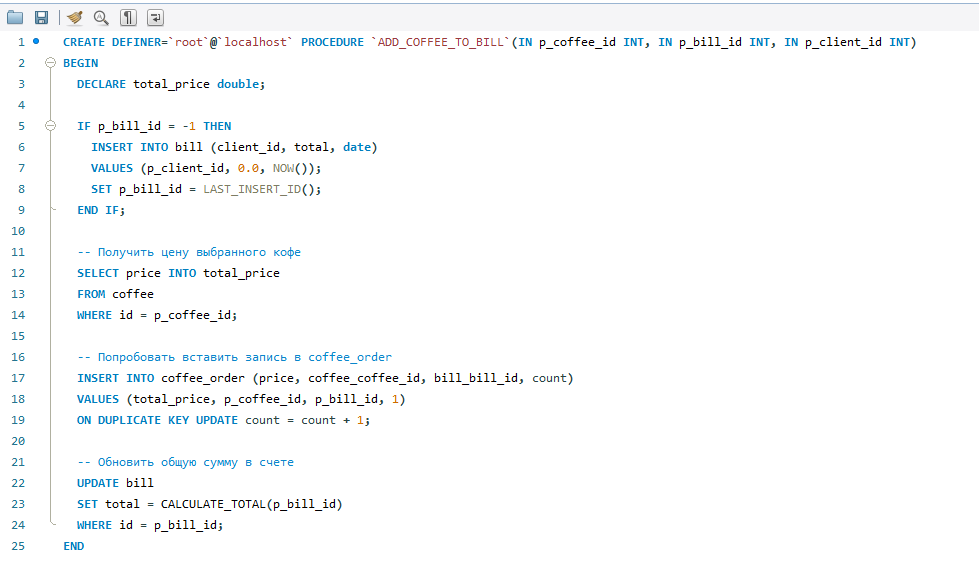
1. схемка базы данных

После этого, было необходимо заполнить базу данных (рисунок 10)

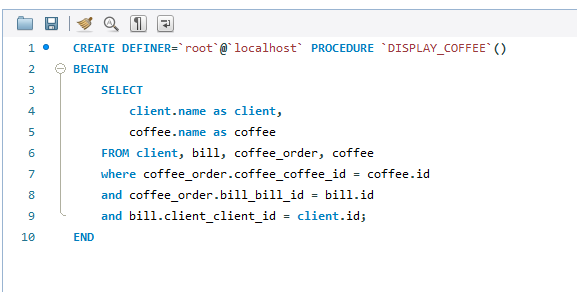


1. заполнение баз данных

Далее, в базе данных было создано две хранимые процедуры. Одна из них, добавляет кофе в заказ (рисунок 11), вторая, отображает заказанные кофе (рисунок 12).

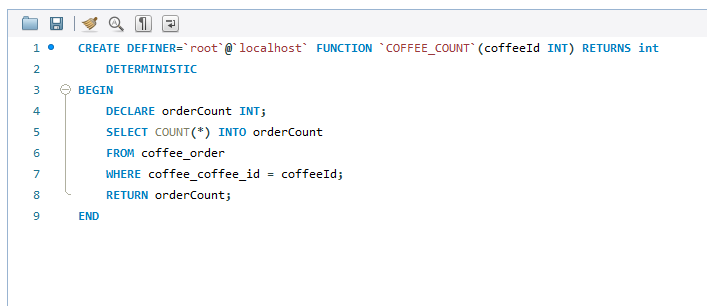


1. добавление кофе в заказ

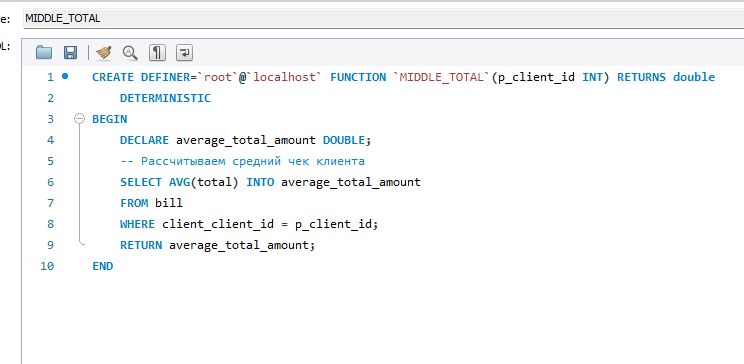


1. отображение заказанных кофе

Так же, были реализованы две функции, одна из которых подсчитывает сколько раз заказывалось кофе (рисунок 13), вторая, считает средний чек клиента (рисунок 14).

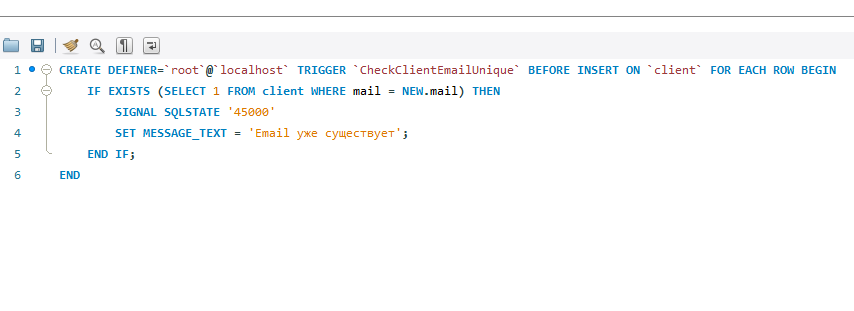


1. подсчет кофе

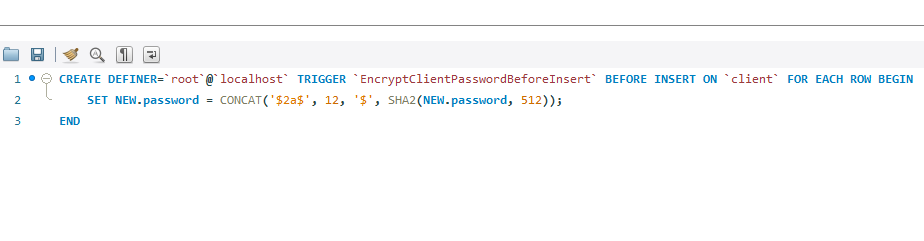


1. средний чек

Далее, было создано два триггера. Один из них, проверяет на уникальность почты пользователя (рисунок 15), второй шифрует введенный пароль (рисунок 16)



1. проверка уникальности почты



1. шифрование пароля
2. **Приложение с графическим интерфейсом**

Созданные триггеры, процедуры и функции были использованы в приложении, написанном на языке Kotlin, с использованием библиотеки exposed. Был написан основной класс для подключения и работой с базой данных (листинг 1)

Листинг 1 – код объекта для работы с базой данных

|  |
| --- |
| object CoffeeHouseDB {  init {  Database.connect(  url = "jdbc:mysql://localhost:3306/coffee\_house",  driver = "com.mysql.cj.jdbc.Driver",  user = "root",  password = "\*\*\*\*\*\*\*\*"  )  }  **fun** **getClientCoffee**(): List<ClientCoffee> {  **return** transaction {  **val** result = mutableListOf<ClientCoffee>()  **val** query = "CALL DISPLAY\_COFFEE()"  exec(query) { rs ->  **while** (rs.next()) {  **val** clientName = rs.getString("client")  **val** coffeeName = rs.getString("coffee")  result.add(ClientCoffee(clientName, coffeeName))  }  }  result  }  }  **fun** **addClient**(client: Client){  transaction {  ClientEntity.insertAndGetId {  it[phone] = client.phone  it[mail] = client.mail  it[card] = client.card  it[name] = client.name  it[secondName] = client.secondName  it[password] = client.password  }  }  }  **fun** **updateClient**(client: Client) {  transaction {  ClientEntity.update({ ClientEntity.id eq client.id }) {  it[phone] = client.phone  it[mail] = client.mail  it[card] = client.card  it[name] = client.name  it[secondName] = client.secondName  it[password] = client.password  }  }  }  **fun** **deleteClient**(clientId: Int) {  transaction {  ClientEntity.deleteWhere { ClientEntity.id eq clientId }  }  }  **fun** **getClients**(): List<Client> {  **val** result = mutableListOf<Client>()  transaction {  ClientEntity.selectAll().forEach {  **var** averageTotal = **0.0**  transaction {  **val** query = "SELECT MIDDLE\_TOTAL(${it[ClientEntity.id].value}) as result"  exec(query) { result ->  **if** (result.next()) {  averageTotal = result.getDouble("result")  }  }  }  result.add(  Client(  it[ClientEntity.id].value,  it[ClientEntity.phone],  it[ClientEntity.mail],  it[ClientEntity.card],  it[ClientEntity.name],  it[ClientEntity.secondName],  it[ClientEntity.password],  averageTotal  )  )  }  }  **return** result  }  **fun** **getCoffee**(): List<Coffee> {  **val** result = mutableListOf<Coffee>()  transaction {  CoffeeEntity.selectAll().forEach {  **var** count = **0**  transaction {  **val** query = "select COFFEE\_COUNT(${it[CoffeeEntity.id].value}) as result"  exec(query) { result ->  **if** (result.next()) {  count = result.getInt("result")  }  }  }  result.add(  Coffee(  it[CoffeeEntity.id].value,  it[CoffeeEntity.name],  it[CoffeeEntity.price],  it[CoffeeEntity.description],  count  )  )  }  }  **return** result  }  **fun** **getAllBillsWithOrders**(toSort: Boolean = **false**, search: String = ""): List<BillWithOrders> {  **return** transaction {  addLogger(StdOutSqlLogger)  **val** baseQuery = BillEntity  .join(ClientEntity, JoinType.INNER, additionalConstraint = { BillEntity.clientId eq ClientEntity.id })  **var** searchQuery =  **if** (search.isNotEmpty()) {  baseQuery.select {  ClientEntity.name eq search  }  } **else** {  baseQuery.selectAll()  }  println(toSort)  **if** (toSort) {  searchQuery = searchQuery.orderBy(BillEntity.total, SortOrder.DESC)  }  searchQuery  .map { row ->  **val** bill = Bill(  id = row[BillEntity.id].value,  total = row[BillEntity.total],  date = row[BillEntity.date],  clientId = row[BillEntity.clientId]  )  **val** client = Client(  id = row[ClientEntity.id].value,  phone = row[ClientEntity.phone],  mail = row[ClientEntity.mail],  card = row[ClientEntity.card],  name = row[ClientEntity.name],  secondName = row[ClientEntity.secondName],  password = row[ClientEntity.password]  )  **val** coffeeList = CoffeeOrderEntity  .join(CoffeeEntity, JoinType.INNER, additionalConstraint = { CoffeeOrderEntity.coffeeId eq CoffeeEntity.id })  .select { CoffeeOrderEntity.billId eq bill.id }  .map {  CoffeeOrder(  price = it[CoffeeOrderEntity.price],  coffeeId = it[CoffeeOrderEntity.coffeeId],  billId = it[CoffeeOrderEntity.billId],  count = it[CoffeeOrderEntity.count],  title = it[CoffeeEntity.name]  )  }  BillWithOrders(bill, client, coffeeList)  }  }  }  **fun** **addCoffeeToBill**(  coffeeId: Int,  billId: Int,  clientId: Int  ) {  **val** query = "CALL ADD\_COFFEE\_TO\_BILL($coffeeId, $billId, $clientId);"  transaction {  exec(query)  }  }  **fun** **getBills**(): List<Bill>{  **val** result = mutableListOf<Bill>()  transaction {  BillEntity.selectAll().forEach {  result.add(  Bill(  it[BillEntity.id].value,  it[BillEntity.total],  it[BillEntity.date],  it[BillEntity.clientId]  )  )  println(it[BillEntity.total])  }  }  **return** result  }  } |

1. **Оконные функции**

Далее были созданы оконные функции (рисунок 17)

Первая функция, с помощью функции COUNT(\*) подчитывает заказы клиентов

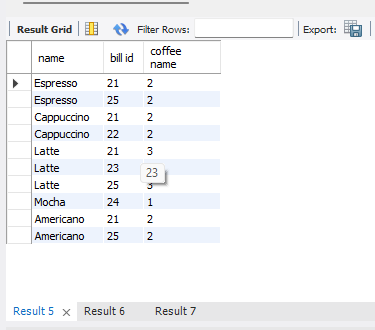
Вторая функция ранжирует заказы клиентов по их счету с помощью функции RANK()

Третья функция отображает последний созданный счет клиента, для этого список счетов клиента сортируется по возрастанию, далее применяется функция FIRST\_VALUE() для получения первого вхождения

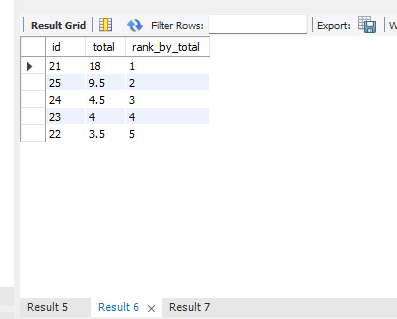


1. оконные функции

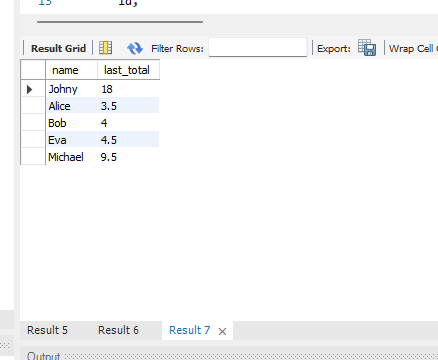
Далее, каждая функция была протестирована (рисунок 18-20)



1. тестирование функций



1. тестирование функций



1. тестирование функций

**ВЫВОД**

В ходе выполнения практических работ, были освоены навыки создание и форматирования баз данных с использованием консоли и приложение с графическим интерфейсом. Так же были освоены и применены навыки для создания хранимых процедур, функций и триггеров. Кроме этого, обретен опыт в создании своих приложений с графическим интерфейсом и подключения их к базам данных. Наконец, применены навыки по написанию оконных функций.

**ЛИТЕРАТУРНЫЕ ИСПТОЧНИКИ**

1. Лекции РТУ МИРЭА «Разработка баз данных» — Богомольная Г.В.
2. Практические занятия РТУ МИРЭА «Разработка баз данных»