МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.
Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

Лабораторная работа №5
по дисциплине: Базы данных
тема: «Организация взаимодействия с базой данных через консольное приложение»

Выполнил: ст. группы ПВ-223
Дмитриев Андрей
Александрович

Проверил:
Панченко Максим Владимирович

Белгород 2024 г.

Вариант 2.

Цель работы: получить навыки подключения к различным системам управления базами данных и взаимодействия с ними. Разработать консольное приложение для взаимодействия с базой данных.

Задание к работе:

1. Подключиться к системе управления базами данных с помощью SQL библиотек выбранного языка программирования.
2. Организовать взаимодействие с базами данных выбранной СУБД (PostgreSQL/MySQL/SQLite).
3. Разработать консольное приложение, в котором производится подключение к базе данных, разработанной на основе предыдущих лабораторных работ, а также обеспечивается выполнение запросов предшествующей лабораторной работы.

Ход работы:

Консольное приложение реализовано на великолепном ЯП Kotlin. Как СУБД используется PostgreSQL, который подключается с помощью стандартной библиотеки JDBC.

Код программы без некоторых запросов:

```
fun main() {
    mainProcess(connectionMode = ConnectionMode.CREATE)
}

fun connect(): Result<Connection> =
    try {
        Class.forName("org.postgresql.Driver")
        Result.success(
            DriverManager.getConnection(
                ConnectionData.URL,
                ConnectionData.USER,
                ConnectionData.PASSWORD
            ).apply { autoCommit = false }
        )
    } catch (e: SQLException) {
        Result.failure(e)
    }

sealed class State {
    class Connection : State()
    class ChangeStore(val conn: java.sql.Connection) : State()
    class Menu(val conn: java.sql.Connection, val storeId: Long) : State()
    class Shopping(val conn: java.sql.Connection, val storeId: Long) :
State()
    class Exit(val conn: java.sql.Connection? = null) : State()
}

enum class ConnectionMode {
    CREATE,
    RECREATE,
    USER_INPUT
}

fun mainProcess(
    connectionMode: ConnectionMode = ConnectionMode.RECREATE,
    scanner: Scanner = Scanner(System.`in`),
) {
```

```
    var state: State = State.Connection()
    while (true) {
        when (state) {
            is State.Connection -> {
                connect().fold(
                    onFailure = { e ->
                        println(e)
                        println("Не удалось выполнить подключение.\nВведите
't', чтобы попробовать снова; 'e' чтобы выйти.")
                        while (true) {
                            val input = scanner.next().lowercase()
                            if (input == "e") {
                                state = State.Exit()
                                break
                            } else if (input == "t") {
                                break
                            } else {
                                println("Неизвестный символ, попробуйте
снова")
                            }
                        }
                    },
                    onSuccess = { conn ->
                        val connection: Connection
                        when (connectionMode) {
                            ConnectionMode.CREATE -> {
                                connection =
conn.executeInitDb().executePrepareDb()
                            }

                            ConnectionMode.RECREATE -> {
                                connection =
conn.executeDeleteDb().executeInitDb().executePrepareDb()
                            }

                            ConnectionMode.USER_INPUT -> {
                                println("Вы желаете продолжить с введёнными
данными? (y/n)")
                                while (true) {
                                    val input = scanner.next().lowercase()
                                    if (input == "y") {
                                        connection =
conn.executeInitDb().executePrepareDb()
                                        break
                                    } else if (input == "n") {
                                        connection =
conn.executeDeleteDb().executeInitDb().executePrepareDb()
                                        break
                                    } else {
                                        println("Неизвестный символ,
попробуйте снова")
                                    }
                                }
                            }
                        }
                        state = State.ChangeStore(connection)
                    }
                )
            }

            is State.ChangeStore -> {
                val possibleIdList = mutableListOf<Long>()
                val castedState = (state as State.ChangeStore)
                castedState.conn.execute("select * from store;") {
                    while (it.next()) {
                        possibleIdList.add(it.getLong(1))
                        println(
```

```
                            StringJoiner("\t")
                                .add("id:" + it.getString(1))
                                .add("адрес:" + it.getString(2))
                        )
                    }
                    println("Введите id магазина:")
                    val storeId = scanner.next().toLong()
                    if (storeId in possibleIdList)
                        state = State.Menu(castedState.conn, storeId)
                    else {
                        println("Магазин с таким id не найден. Попробуйте
снова:")
                        state = State.ChangeStore(castedState.conn)
                    }
                }
            }

            is State.Menu -> {
                val castedState = (state as State.Menu)
                println(
                    """
                    Выберите действие:
                    1. Сделать покупку
                    2. История покупок
                    3. Рейтинг товаров по магазину
                    4. Выход
                    """.trimIndent()
                )

                val input = scanner.next().toInt()
                when (input) {
                    1 -> state = State.Shopping(castedState.conn,
castedState.storeId)
                    2 -> {
                        castedState.conn.execute(
                            """
                            select  t0.${Table.CheckList.ID},
                                    t2.${Table.Product.NAME},
                                    t1.${Table.Purchase.AMOUNT},
                                    t2.${Table.Product.QUANTITY_TO_ASSESS},

t1.${Table.Purchase.AMOUNT}*t3.${Table.Accounting.COST}
                            from ${Table.CheckList.T_NAME} as t0
                            inner join ${Table.Purchase.T_NAME} as t1
                            on
t0.${Table.CheckList.ID}=t1.${Table.Purchase.CHECK_LIST_ID}
                                and
t0.${Table.CheckList.STORE_ID}=${castedState.storeId}
                            inner join ${Table.Product.T_NAME} as t2
                            on
t1.${Table.Purchase.T_NAME}=t2.${Table.Product.ARTICLE}
                            inner join ${Table.Accounting.T_NAME} as t3
                            on
t1.${Table.Purchase.PRODUCT_ARTICLE}=t3.${Table.Accounting.PRODUCT_ARTICLE}
                                and
t3.${Table.Accounting.STORE_ID}=${castedState.storeId}
                            order by t0.${Table.CheckList.ID} asc;
                            """
                        ) {
                            var prevId: Long = -1
                            while (it.next()) {
                                val currId = it.getLong(1)
                                if (prevId != currId) {
                                    println("check id: $currId")
                                    prevId = currId
                                }
                                println(
```

```
                                    StringJoiner("\t")
                                        .add(it.getString(2))
                                        .add(it.getString(3) + "_" +
it.getString(4))
                                        .add("цена: " + it.getString(5) +
"p")
                                )
                            }
                        }
                    }

                    3 -> castedState.conn.execute(
                        """
                        select  t2.${Table.Product.NAME},

round(coalesce(sum(t1.${Table.Purchase.AMOUNT})*100/(select sum(amount) from
${Table.Purchase.T_NAME}),0),2)
                                    as amount_in_percent
                            from check_list as t0
                            inner join ${Table.Purchase.T_NAME} as t1
                            on t0.id=t1.${Table.Purchase.CHECK_LIST_ID}
                                and
t0.${Table.CheckList.STORE_ID}=${castedState.storeId}
                            right join ${Table.Product.T_NAME} as t2
                            on
t1.${Table.Purchase.PRODUCT_ARTICLE}=t2.${Table.Product.ARTICLE}
                            group by t2.${Table.Product.NAME}
                            order by amount_in_percent desc;
                        """
                    ) {
                        while (it.next())
                            println(
                                StringJoiner("\t")
                                    .add(it.getString(1))
                                    .add(it.getString(2)+"%")
                            )
                    }

                    4 -> {
                        println("До встречи!")
                        state = State.Exit()
                    }

                    else -> {
                        println("Неизвестный символ, попробуйте снова")
                        // -> State.Menu
                    }
                }
            }

            is State.Shopping -> {
                val castedState = state as State.Shopping

                println("Ассортимент:")
                val assortment = mutableListOf<Purchase>()
                castedState.conn.execute(
                    """
                        select  t0.${Table.Accounting.PRODUCT_ARTICLE},
                                t1.${Table.Product.NAME},
                                t0.${Table.Accounting.AMOUNT},
                                t0.${Table.Accounting.COST},
                                t1.${Table.Product.QUANTITY_TO_ASSESS}
                            from ${Table.Accounting.T_NAME} as t0
                            inner join ${Table.Product.T_NAME} as t1
                            on
t0.${Table.Accounting.PRODUCT_ARTICLE}=t1.${Table.Product.ARTICLE}
                            where
```

```
${Table.Accounting.STORE_ID}=${castedState.storeId};
                        """
                ) {
                    while (it.next()) {
                        assortment.add(Purchase(it.getLong(1),
it.getDouble(3)))
                        println(
                            StringJoiner("\t")
                                .add(it.getLong(1).toString())
                                .add(it.getString(2))
                                .add(it.getDouble(3).toString())
                                .add(it.getDouble(4).toString() + "p")
                                .add(it.getString(5))
                        )
                    }
                }
                println("Выберите товары для покупки, артикул и количество
через пробел.\nЧтобы завершить нажмите 'e'")
                val purchases = mutableListOf<Purchase>()
                while (true) {
                    val splitInput = scanner.nextLine().split(" ")
                    if (splitInput.size == 1 && splitInput[0] == "e") {
                        castedState.conn.executeBuy(castedState.storeId,
purchases)
                        println("Покупка совершена успешно")
                        state = State.Menu(castedState.conn,
castedState.storeId)
                        break
                    } else if (splitInput.size == 2) {
                        try {
                            val article = splitInput[0].toLong()
                            val amount = splitInput[1].toDouble()

                            val productFromAssortment = assortment.findLast
{ it.article == article }
                            if (productFromAssortment == null) {
                                println("Товар с таким артикулом не найден")
                            } else if (amount >
productFromAssortment.amount) {
                                println("Недостаточно товара на складе")
                            } else {
                                purchases.add(Purchase(article, amount))
                                println("Товар с артиклом $article
добавлен")
                            }
                        } catch (e: NumberFormatException) {
                            println("Некорректный ввод чисел")
                        }
                    } else if (splitInput.isEmpty() || splitInput[0]=="") {
                        // ignore
                    } else {
                        println("Некорректный ввод")
                    }
                }
            }

            is State.Exit -> {
                (state as State.Exit).conn?.close()
                break
            }
        }
    }
}
```

Пример работы:

```
id:1    адрес:LA, 5 Avenue
id:2    адрес:LA, 11 Avenue
id:3    адрес:LA, 12 Avenue
Введите id магазина:
1
Выберите действие:
1. Сделать покупку
2. История покупок
3. Рейтинг товаров по магазину
4. Выход
1
Ассортимент:
1       Колбаса докторская      700.0   100.0р  100g
2       Сырок плавленный        300.0   30.0р   p
3       Молоко, бутылка 1л      300.0   70.0р   p
4       Булочка "Лакомка"       200.0   35.0р   p
5       Стейк, говяжий 100.0    200.0р  p
Выберите товары для покупки, артикул и количество через пробел.
Чтобы завершить нажмите 'e'
1
Некорректный ввод
1 5
Товар с артиклом 1 добавлен
2 3
Товар с артиклом 2 добавлен
e
Покупка совершена успешно
Выберите действие:
1. Сделать покупку
2. История покупок
3. Рейтинг товаров по магазину
4. Выход
2
check id: 1
Колбаса докторская      5.00_100g       цена: 500.00р
Сырок плавленный        3.00_p  цена: 90.00р
Выберите действие:
1. Сделать покупку
2. История покупок
3. Рейтинг товаров по магазину
4. Выход
3
Колбаса докторская      62.50%
Сырок плавленный        37.50%
Молоко, бутылка 1л      0.00%
Булочка "Лакомка"       0.00%
Стейк, говяжий 0.00%
Выберите действие:
1. Сделать покупку
2. История покупок
3. Рейтинг товаров по магазину
4. Выход
```

Остальные вспомогательные функции:

```
object Table {
    object Purchase {
        const val T_NAME = "purchase"

        const val ID = "id"
        const val CHECK_LIST_ID = "check_list_id"
        const val PRODUCT_ARTICLE = "product_article"
        const val AMOUNT = "amount"
    }
```

```
    object CheckList {
        const val T_NAME = "check_list"

        const val ID = "id"
        const val STORE_ID = "store_id"
        const val TIME = "time"
    }

    object Accounting {
        const val T_NAME = "accounting"

        const val STORE_ID = "store_id"
        const val PRODUCT_ARTICLE = "product_article"
        const val COST = "cost"
        const val AMOUNT = "amount"
    }

    object Store {
        const val T_NAME = "store"

        const val ID = "id"
        const val ADDRESS = "address"
    }

    object Product {
        const val T_NAME = "product"

        const val ARTICLE = "article"
        const val NAME = "name"
        const val CATEGORY = "category"
        const val QUANTITY_TO_ASSESS = "quantity_to_assess"
    }
}

enum class Category(val value: String) {
    MEAT("meat"),
    MILK("milk"),
    BAKE("bake")
}

enum class QuantityToAssess(val value: String) {
    PIECE("p"),
    GRAM_100("100g"),
    KILOGRAM_1("1kg")
}

class Purchase(val article: Long, val amount: Double)

object ConnectionData {
    const val URL = "jdbc:postgresql://localhost:5432/postgres"
    const val USER = "postgres"
    const val PASSWORD = "root"
}

private fun Connection._execute(block: (statement: Statement) -> Unit) =
    try {
        apply {
            createStatement().use { statement ->
                block(statement)
                commit()
            }
        }
    } catch (e: Exception) {
        rollback()
        throw e
    }
```

```
fun Connection.execute(query: String, handler: (ResultSet) -> Unit = {}):
Connection =
    _execute { handler(it.executeQuery(query)) }

fun Connection.execute(query: String): Connection =
    _execute { it.execute(query) }

fun Connection.executeDeleteDb(): Connection =
    """
    DROP TABLE IF EXISTS ${Table.Purchase.T_NAME};
    DROP TABLE IF EXISTS ${Table.Accounting.T_NAME};
    DROP TABLE IF EXISTS ${Table.CheckList.T_NAME} CASCADE;
    DROP TABLE IF EXISTS ${Table.Store.T_NAME} CASCADE;
    DROP TABLE IF EXISTS ${Table.Product.T_NAME} CASCADE;
    """.let { execute(it) }

fun Connection.executeInitDb(): Connection =
    """
    -- таблица магазинов
    CREATE TABLE IF NOT EXISTS ${Table.Store.T_NAME} (
        ${Table.Store.ID} BIGSERIAL PRIMARY KEY,
        ${Table.Store.ADDRESS} VARCHAR (100)
    );

    -- таблица продукции, поставляемой в магазины
    CREATE TABLE IF NOT EXISTS ${Table.Product.T_NAME} (
        ${Table.Product.ARTICLE} BIGSERIAL PRIMARY KEY,
        ${Table.Product.NAME} VARCHAR (100) NOT NULL,
        ${Table.Product.CATEGORY} VARCHAR (10),
        ${Table.Product.QUANTITY_TO_ASSESS} VARCHAR (10) NOT NULL
    );

    -- таблица чеков со связью с магазином
    CREATE TABLE IF NOT EXISTS ${Table.CheckList.T_NAME} (
        ${Table.CheckList.ID} BIGSERIAL NOT NULL PRIMARY KEY,
        ${Table.CheckList.STORE_ID} BIGSERIAL REFERENCES store (id)
            ON DELETE SET NULL,     -- если удалить магазин,
                                    -- то чек останется
        ${Table.CheckList.TIME} TIMESTAMP NOT NULL
    );

    -- таблица учёта
    CREATE TABLE IF NOT EXISTS ${Table.Accounting.T_NAME} (
        --${Table.Accounting} BIGSERIAL NOT NULL PRIMARY KEY,
        ${Table.Accounting.STORE_ID} BIGSERIAL NOT NULL REFERENCES store
(id)
            ON DELETE SET NULL,
        ${Table.Accounting.PRODUCT_ARTICLE} BIGSERIAL NOT NULL REFERENCES
product (article)
            ON DELETE SET NULL,
        ${Table.Accounting.COST} NUMERIC(6, 2),
        ${Table.Accounting.AMOUNT} INTEGER NOT NULL default 0,
        PRIMARY KEY (${Table.Accounting.STORE_ID},
${Table.Accounting.PRODUCT_ARTICLE})
    );

    -- покупка является строкой в чеке
    CREATE TABLE IF NOT EXISTS ${Table.Purchase.T_NAME} (
        ${Table.Purchase.T_NAME} BIGSERIAL NOT NULL PRIMARY KEY,
        ${Table.Purchase.CHECK_LIST_ID} BIGSERIAL NOT NULL REFERENCES
check_list (id)
            ON DELETE SET NULL,
        ${Table.Purchase.PRODUCT_ARTICLE} BIGSERIAL NOT NULL REFERENCES
product (article)
            ON DELETE SET NULL,
        ${Table.Purchase.AMOUNT} NUMERIC(6, 2) NOT NULL
```

```
    );
    """.let { execute(it) }

fun Connection.executePrepareDb(): Connection =
    """
    INSERT INTO ${Table.Store.T_NAME}
            (${Table.Store.ID}, ${Table.Store.ADDRESS})
        VALUES
            (1, 'LA, 5 Avenue'),
            (2, 'LA, 11 Avenue'),
            (3, 'LA, 12 Avenue')
        ON CONFLICT (${Table.Store.ID}) DO NOTHING;

    INSERT INTO ${Table.Product.T_NAME}
            (${Table.Product.ARTICLE}, ${Table.Product.NAME},
${Table.Product.CATEGORY}, ${Table.Product.QUANTITY_TO_ASSESS})
        VALUES
            (1, 'Колбаса докторская', 'meat', '100g'),
            (2, 'Сырок плавленный', 'milk', 'p'),
            (3, 'Молоко, бутылка 1л', 'milk', 'p'),
            (4, 'Булочка "Лакомка"', 'bake', 'p'),
            (5, 'Стейк, говяжий', 'meat', 'p')
        ON CONFLICT (${Table.Product.ARTICLE}) DO NOTHING;

    INSERT INTO ${Table.Accounting.T_NAME}
            (${Table.Accounting.STORE_ID},
${Table.Accounting.PRODUCT_ARTICLE}, ${Table.Accounting.COST},
${Table.Accounting.AMOUNT})
        VALUES
            (1, 1, 100, 700),
            (1, 2, 30, 300),
            (1, 3, 70, 300),
            (1, 4, 35, 200),
            (1, 5, 200, 100),
            (2, 1, 190, 700),
            (2, 2, 39, 300),
            (2, 3, 79, 300),
            (2, 4, 39, 200),
            (2, 5, 290, 100)
        ON CONFLICT (${Table.Accounting.STORE_ID},
${Table.Accounting.PRODUCT_ARTICLE}) DO NOTHING;
    """.let { execute(it) }

fun Connection.executeBuy(
    storeId: Long,
    purchases: List<Purchase>,
): Connection {
    try {


        val generatedCheckListId = prepareStatement(
            """
        INSERT INTO ${Table.CheckList.T_NAME}
            (${Table.CheckList.STORE_ID}, ${Table.CheckList.TIME})
        VALUES
            ($storeId, NOW());
        """,
            arrayOf(Table.CheckList.ID)
        ).use { preparedStmt ->
            preparedStmt.executeUpdate()
            preparedStmt
                .generatedKeys
                .apply { next() }
                .getLong(1)
        }

        val queryAddPurchaseBuilder = StringBuilder()
```

```
        purchases.forEach { purchase ->
            queryAddPurchaseBuilder
                .append(
                    """
                INSERT INTO ${Table.Purchase.T_NAME}
                    (${Table.Purchase.CHECK_LIST_ID},
${Table.Purchase.PRODUCT_ARTICLE}, ${Table.Purchase.AMOUNT})
                VALUES
                    ($generatedCheckListId, ${purchase.article},
${purchase.amount});

                """
                )
                .append(
                    """
                UPDATE ${Table.Accounting.T_NAME}
                    SET
${Table.Accounting.AMOUNT}=${Table.Accounting.AMOUNT}-${purchase.amount}
                    WHERE ${Table.Accounting.STORE_ID}=$storeId
                        AND
${Table.Accounting.PRODUCT_ARTICLE}=${purchase.article};

                """
                )
        }

        createStatement().execute(queryAddPurchaseBuilder.toString())

        commit()
    } catch (e: Exception) {
        rollback()
        throw e
    }
    return this
}
```

Вывод: в ходе работы получены навыки подключения к различным системам управления базами данных и взаимодействия с ними. В ходе работы создано консольное приложение для взаимодействия с базой данных.