Файл: /home/sergey/Documents/configurate/FracSin.py (относительно: FracSin.py)

```
import numpy as np
import matplotlib.pyplot as plt
# Параметры
levels = 7
# Амплитуды: A_k = A0 * (0.3)^k
A0 = 1.0
decay = 0.32
# Частоты: f_k = f0 * 8^k
f0 = 1.0
# Генерируем массивы амплитуд и частот
A = [A0 * (decay ** k) for k in range(levels)]
f = [f0 * (8 ** k) for k in range(levels)]
# Параметрическая кривая: x(t) = t, y(t) = Σ A_k * sin(2π f_k t + φ_k)  (но это не даст обвивания)
# Для обвивания нужно на каждом уровне смещать по нормали, но при 7 уровнях это крайне тяжело вычислить.
# Поэтому я предлагаю упрощённую модель: кривая задаётся как (X(t), Y(t)) = (t, F(t)), где F(t) = Σ A_k sin(2π f_k t)
# Это даст сильную осцилляцию, но без самопересечений. Для самопересечений нужно параметрическое обвивание.
# Для демонстрации сложности построим график функции y = Σ A_k sin(2π f_k t) (уровни накладываются аддитивно)
# Это не совсем то, что мы рисовали раньше, но зато вычислимо для 7 уровней.
t = np.linspace(0, 2*np.pi, 50000)
y = np.zeros_like(t)
for k in range(levels):
    y += A[k] * np.sin(f[k] * t)
plt.figure(figsize=(14, 6))
plt.plot(t, y, 'b-', lw=0.3)
plt.title('Сумма синусоид: 7 уровней, частота растёт в 8 раз, амплитуда умножается на {decay}')
plt.xlabel('t')
plt.ylabel('y')
plt.grid(True, alpha=0.2)
plt.show()
# Теперь увеличенный фрагмент, чтобы увидеть мелкие колебания
t_fragment = np.linspace(0, 0.05, 20000)
y_fragment = np.zeros_like(t_fragment)
for k in range(levels):
    y_fragment += A[k] * np.sin(f[k] * t_fragment)
plt.figure(figsize=(14, 6))
plt.plot(t_fragment, y_fragment, 'b-', lw=0.5)
plt.title('Фрагмент [0, 0.05] — видны высокочастотные осцилляции')
plt.xlabel('t')
plt.ylabel('y')
plt.grid(True, alpha=0.2)
plt.show()
```

Файл: /home/sergey/Documents/configurate/base_manager.py (относительно: base_mana

```
# base_manager.py
from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QTableWidget, QTableWidgetItem,
                             QMessageBox, QHeaderView)
from PyQt5.QtCore import Qt
from database import DatabaseService

class BaseEntityManager(QWidget):
    """
    Базовый класс для виджетов управления справочниками.
    Предоставляет таблицу и виртуальные методы для CRUD.
    """
    def __init__(self, parent=None, db: DatabaseService = None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.entity_name = ""  # должно быть переопределено
        self.table_headers = []  # заголовки колонок
        self.key_column = "id"  # первичный ключ
        self.data = []  # кэш данных
        layout = QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        # Кнопки CRUD (будут подключены в подклассе)
        self.crud_buttons = None  # будет установлен в подклассе
        # Таблица
        self.table = QTableWidget()
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        self.table.setSelectionMode(QTableWidget.SingleSelection)
        self.table.horizontalHeader().setStretchLastSection(True)
        self.table.verticalHeader().setVisible(False)
        self.table.itemDoubleClicked.connect(self.on_item_double_clicked)
        layout.addWidget(self.table)
        self._setup_table()
    def _setup_table(self):
        if self.table_headers:
            self.table.setColumnCount(len(self.table_headers))
            self.table.setHorizontalHeaderLabels(self.table_headers)
    def set_crud_buttons(self, crud_widget):
        """Привязать виджет с кнопками и подключить сигналы."""
        self.crud_buttons = crud_widget
        self.crud_buttons.add_clicked.connect(self.add_item)
        self.crud_buttons.edit_clicked.connect(self.edit_item)
        self.crud_buttons.delete_clicked.connect(self.delete_item)
    def load_data(self):
        """Загрузить данные из БД и обновить таблицу. Переопределяется в подклассах."""
        pass
    def refresh_table(self):
        """Обновить отображение таблицы на основе self.data."""
        self.table.setRowCount(0)
        if not self.data:
            return
        self.table.setRowCount(len(self.data))
        for row_idx, row_data in enumerate(self.data):
            for col_idx, col_name in enumerate(self.table_headers):
                value = row_data.get(col_name, "")
                item = QTableWidgetItem(str(value))
                item.setData(Qt.UserRole, row_data.get(self.key_column))  # сохраняем id
                self.table.setItem(row_idx, col_idx, item)
    def get_selected_id(self):
        """Возвращает id выбранной строки или None."""
        current_row = self.table.currentRow()
        if current_row >= 0:
            item = self.table.item(current_row, 0)
            if item:
                return item.data(Qt.UserRole)
        return None
    def add_item(self):
        """Добавление нового элемента. Переопределяется."""
        pass
    def edit_item(self):
        """Редактирование выбранного элемента. Переопределяется."""
        pass
    def delete_item(self):
        """Удаление выбранного элемента. Переопределяется."""
        pass
    def on_item_double_clicked(self, item):
        """По двойному клику вызываем редактирование."""
        self.edit_item()
```

Файл: /home/sergey/Documents/configurate/composition_manager.py (относительно: com

```
# composition_manager.py
from PyQt5.QtWidgets import QMessageBox
```

```
from database import DatabaseService

class CompositionManager:
    def __init__(self, db=None, parent_widget=None):
        self.db = db or DatabaseService()
        self.parent = parent_widget
    def get_entity_cname(self, entity_id: int) -> str:
        """Возвращает системное имя (cname) сущности, приводит к нижнему регистру."""
        res = self.db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (entity_id,))
        if not res:
            raise ValueError(f"Сущность с id {entity_id} не найдена")
        return res[0][0].lower()
    def ensure_ext_table_and_field(self, child_entity_id, link_field_name):
        # Получаем cname дочерней сущности
        child_cname = self.get_entity_cname(child_entity_id)
        ext_table = f"ext.ext_{child_cname}"
        # Проверяем существование таблицы
        check_tab = self.db.execute_query("SELECT to_regclass(%s)", (ext_table,))
        if not check_tab or not check_tab[0][0]:
            sql_create = f"""
                CREATE TABLE IF NOT EXISTS {ext_table} (
                    id SERIAL PRIMARY KEY,
                    versionid INTEGER NOT NULL REFERENCES data.entity_versions(id) ON DELETE CASCADE
            """
            self.db.execute_query(sql_create, fetch=False)
        # Проверяем наличие поля в meta.fields
        check_field = self.db.execute_query(
            "SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s",
            (child_entity_id, link_field_name)
        )
        if not check_field:
            ins = self.db.execute_query(
                "INSERT INTO meta.fields (entitytypeid, cfieldname, cfieldtype, calias, lisindexed, status_id)
                VALUES (%s, %s, %s, %s, %s, %s) RETURNING id",
                (child_entity_id, link_field_name, f"FK: {link_field_name}")
            )
            if ins:
                field_id = ins[0][0]
                self.db.execute_procedure("meta.p_field_sync", (field_id,))
        return True
    def add_composition(self, parent_entityid, child_entityid, role_id, link_field, sortorder=10):
        if not self.ensure_ext_table_and_field(child_entityid, link_field):
            if self.parent:
                QMessageBox.warning(self.parent, "Ошибка", "Не удалось подготовить ext-таблицу/поле")
            return False
        check = self.db.execute_query(
            """SELECT id FROM meta.entity_composition
            WHERE parent_entityid=%s AND child_entityid=%s AND role_id=%s""",
            (parent_entityid, child_entityid, role_id)
        )
        if check:
            if self.parent:
                QMessageBox.warning(self.parent, "Ошибка", "Такая связь уже существует")
            return False
        sql = """INSERT INTO meta.entity_composition
            (parent_entityid, child_entityid, role_id, c_link_fieldname, isortorder)
            VALUES (%s, %s, %s, %s, %s)"""
        self.db.execute_query(sql, (parent_entityid, child_entityid, role_id, link_field, sortorder), fetch=False)
        return True
    def update_composition(self, comp_id, parent_entityid, child_entityid, role_id, link_field, sortorder=10):
        if not self.ensure_ext_table_and_field(child_entityid, link_field):
            if self.parent:
                QMessageBox.warning(self.parent, "Ошибка", "Не удалось подготовить ext-таблицу/поле")
            return False
        check = self.db.execute_query(
            """SELECT id FROM meta.entity_composition
            WHERE parent_entityid=%s AND child_entityid=%s AND role_id=%s AND id!=%s""",
            (parent_entityid, child_entityid, role_id, comp_id)
        )
        if check:
            if self.parent:
                QMessageBox.warning(self.parent, "Ошибка", "Такая связь уже существует")
            return False
        sql = """UPDATE meta.entity_composition SET
                parent_entityid=%s, child_entityid=%s, role_id=%s,
                c_link_fieldname=%s, isortorder=%s
            WHERE id=%s"""
self.db.execute_query(sql, (parent_entityid, child_entityid, role_id, link_field, sortorder, comp_id), fetch=False)
        return True
    def delete_composition(self, comp_id):
        self.db.execute_query("DELETE FROM meta.entity_composition WHERE id=%s", (comp_id,), fetch=False)
```

Файл: /home/sergey/Documents/configurate/create_import_structure.py (относительно: cr

```
# create_import_structure.py
import os
ROOT = "/home/sergey/Documents/configurate"
STRUCTURE = {
    "mozart_import/graph": [
        "__init__.py",
        "graphics_items.py",
        "graphics_view.py",
        "relation_scene.py",
        "link_line.py",
    ],
    "mozart_import/wizards": [
        "__init__.py",
        "source_page.py",
        "schema_page.py",
        "relation_page.py",
        "mapping_detail_page.py",
        "execute_page.py",
    ],
    "mozart_import/core": [
        "__init__.py",
        "models.py",
        "source_analyzer.py",
        "mapping_manager.py",
    ],
    "mozart_import/utils": [
        "__init__.py",
        "field_creator.py",
    ],
    "mozart_import/connectors": [],  # уже есть, оставляем
}
def create_structure():
    for dirpath, files in STRUCTURE.items():
        full_path = os.path.join(ROOT, dirpath)
        os.makedirs(full_path, exist_ok=True)
        for filename in files:
            file_path = os.path.join(full_path, filename)
            if not os.path.exists(file_path):
                with open(file_path, "w", encoding='utf-8') as f:
                    f.write("# -*- coding: utf-8 -*-\n")
                print(f"Created: {file_path}")
            else:
                print(f"Exists: {file_path}")
if __name__ == "__main__":
    create_structure()
    print("Done.")
```

Файл: /home/sergey/Documents/configurate/crud_buttons.py (относительно: crud_buttons

```
# crud_buttons.py
# crud_buttons.py
from PyQt5.QtWidgets import QWidget, QHBoxLayout, QPushButton
from PyQt5.QtCore import pyqtSignal
from lang.local_translator import LocalTranslator
class CrudButtons(QWidget):
    add_clicked = pyqtSignal()
```

```
    edit_clicked = pyqtSignal()
    delete_clicked = pyqtSignal()
    def __init__(self, parent=None, with_add=True, with_edit=True, with_delete=True):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.with_add = with_add
        self.with_edit = with_edit
        self.with_delete = with_delete
        layout = QHBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        if with_add:
            self.btn_add = QPushButton()
            self.btn_add.clicked.connect(self.add_clicked.emit)
            layout.addWidget(self.btn_add)
        if with_edit:
            self.btn_edit = QPushButton()
            self.btn_edit.clicked.connect(self.edit_clicked.emit)
            layout.addWidget(self.btn_edit)
        if with_delete:
            self.btn_del = QPushButton()
            self.btn_del.clicked.connect(self.delete_clicked.emit)
            layout.addWidget(self.btn_del)
        layout.addStretch()
        self.retranslate_ui()
    def retranslate_ui(self):
        if self.with_add and hasattr(self, 'btn_add'):
            self.btn_add.setText(self.translator.tr('btn_add'))
        if self.with_edit and hasattr(self, 'btn_edit'):
            self.btn_edit.setText(self.translator.tr('btn_edit'))
        if self.with_delete and hasattr(self, 'btn_del'):
            self.btn_del.setText(self.translator.tr('btn_delete'))
```

Файл: /home/sergey/Documents/configurate/database.py (относительно: database.py)

```
#database.py
import psycopg2
from ide_core import goProject

class DatabaseService:
    """Единый класс для взаимодействия с БД, обёртка над goProject."""
    def __init__(self):
        self._conn = goProject.db_conn
    def execute_query(self, sql, params=None, fetch=True):
        """Выполнить SQL-запрос (SELECT или модифицирующий)."""
        if fetch:
            return goProject.f_Execute(sql, params)
        else:
            try:
                with self._conn.cursor() as cur:
                    cur.execute(sql, params)
                    return True
            except Exception as e:
                print(f"DB Error: {e}")
                return False
    def execute_procedure(self, proc_name, params=None):
        if params:
            placeholders = ', '.join(['%s'] * len(params))
            sql = f"CALL {proc_name}({placeholders})"
        else:
            sql = f"CALL {proc_name}()"
        return self.execute_query(sql, params, fetch=False)
    def get_connection(self):
        return self._conn
    def refresh_connection(self):
        self._conn = goProject.db_conn
```

Файл: /home/sergey/Documents/configurate/ide_core.py (относительно: ide_core.py)

```
# ide_core.py
import psycopg2
import sys
import os

class GoProject:
    def __init__(self):
        # --- Закладка 1: БАЗА ДАННЫХ (Layer 1) ---
        self.db_host = "localhost"
        self.db_port = "5432"
        self.db_name = "mozart_erp"
        self.db_user = "postgres"
        self.db_pass = ""
        self.db_conn = None  # Активное соединение psycopg2
        self.current_user_id = None  # ID текущего пользователя (конфигуратора)
        # --- Закладка 2: СРЕДНИЙ СЛОЙ (Layer 2) ---
        self.l2_path = os.getcwd() + "/layer2_server"
        self.l2_port = 8000
        self.l2_process = None
        # --- Закладка 3: КЛИЕНТ (Layer 3) ---
        self.l3_path = os.getcwd() + "/layer3_client"
        self.l3_starter = "starter.py"
        self.l3_process = None
    def p_Connect(self):
        """Установка физического соединения с PostgreSQL.
        Проверяет, что пользователь имеет роль конфигуратора.
        """
        try:
            # Закрываем старое соединение, если оно было
            if self.db_conn:
                self.db_conn.close()
            self.db_conn = psycopg2.connect(
                host=self.db_host,
                port=self.db_port,
                database=self.db_name,
                user=self.db_user,
                password=self.db_pass,
                connect_timeout=5
            )
            self.db_conn.autocommit = True
            # Проверяем, является ли текущий пользователь конфигуратором
            with self.db_conn.cursor() as cur:
                # Сначала получаем ID пользователя по логину
                cur.execute("SELECT id FROM auth.users WHERE login = %s", (self.db_user,))
                row = cur.fetchone()
                if not row:
                    self.db_conn.close()
                    self.db_conn = None
                    return False, f"Пользователь {self.db_user} не найден в системе."
                user_id = row[0]
                # Проверяем наличие роли с типом 'CONFIG'
                cur.execute("""
                    SELECT 1 u.id
                    FROM auth.users u
                    JOIN auth.userrolecontext urc ON u.id = urc.userid
                    JOIN auth.roles r ON urc.roleid = r.id
                    JOIN auth.role_types rt ON r.role_type_id = rt.id
                    WHERE u.id = %s AND rt.callas = 'CONFIG'
                """, (user_id,))
                if not cur.fetchone():
                    self.db_conn.close()
                    self.db_conn = None
```

```
            return False, "У вас нет прав конфигуратора. Доступ запрещён."
        self.current_user_id = user_id
        return True, "Соединение со Слоном установлено успешно. Добро пожаловать, конфигуратор!"
    except Exception as e:
        self.db_conn = None
        return False, f"Ошибка подключения: {str(e)}"
def Execute(self, sql, params=None):
    """ Выполнение SQL-запроса (SELECT).
    Возвращает коллекцию строк (IoResult). [0.3]
    """
    if not self.db_conn:
        return []
    try:
        with self.db_conn.cursor() as cur:
            cur.execute(sql, params)
            if cur.description:  # Если запрос возвращает данные (SELECT)
                return cur.fetchall()
            return []
    except Exception as e:
        print(f"SQL Error: {e}")
        return []
def SaveProject(self, file_path):
    """ Сохранение манифеста проекта в .mzt файл (JSON) """
    data = {
        "db": {
            "host": self.db_host, "port": self.db_port,
            "name": self.db_name, "user": self.db_user, "pass": self.db_pass
        },
        "l2": {"path": self.l2_path, "port": self.l2_port},
        "l3": {"path": self.l3_path, "starter": self.l3_starter}
    }
    with open(file_path, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=4, ensure_ascii=False)
def LoadProject(self, file_path):
    """ Загрузка параметров проекта из .mzt файла """
    if not os.path.exists(file_path):
        return False, "Файл проекта не найден."
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        self.db_host = data['db'].get('host', 'localhost')
        self.db_port = data['db'].get('port', '5432')
        self.db_name = data['db'].get('name', '')
        self.db_user = data['db'].get('user', 'postgres')
        self.db_pass = data['db'].get('pass', '')
        self.l2_path = data['l2'].get('path', '')
        self.l2_port = data['l2'].get('port', '8000')
        self.l3_path = data['l3'].get('path', '')
        self.l3_starter = data['l3'].get('starter', 'starter.py')
        return True, "Проект загружен."
    except Exception as e:
        return False, f"Ошибка чтения файла: {str(e)}"

# Создаем единственный экземпляр объекта управления проектом
goProject = GoProject()
```

Файл: /home/sergey/Documents/configurate/ide_db_work.py (относительно: ide_db_work.p

```
# ide_db_work.py
# ide_db_work.py
# ide_db_work.py
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QTabWidget, QPushButton
from PyQt5.QtCore import Qt
from database.database import DatabaseService
from tabs.tab_balance_units_tab, tab_users_roles_tab, EntitiesTab,
                DocumentsTab, DataViewTab, RelationsTab
from lang.local_translator import LocalTranslator

class DBWorkDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('window_title_db_work'))
        self.resize(1300, 800)
        self.db = DatabaseService()
        layout = QVBoxLayout(self)
        self.tabs = QTabWidget()
        self.tabs.addTab(BalanceUnitsTab(self, db=self.db), self.translator.tr('tab_balance_units'))
        self.tabs.addTab(UsersRolesTab(self, db=self.db), self.translator.tr('tab_users_roles'))
        self.tabs.addTab(EntitiesTab(self, db=self.db), self.translator.tr('tab_entities'))
        self.tabs.addTab(DocumentsTab(self, db=self.db), self.translator.tr('tab_documents'))
        self.tabs.addTab(DataViewTab(self, db=self.db), self.translator.tr('tab_data_view'))
        self.tabs.addTab(RelationsTab(self, db=self.db), self.translator.tr('tab_relations'))
        layout.addWidget(self.tabs)
        self.btn_close = QPushButton(self.translator.tr('btn_close'))
        self.btn_close.clicked.connect(self.accept)
        layout.addWidget(self.btn_close, alignment=Qt.AlignRight)
        # Применяем текущий язык ко всем элементам
        self.retranslate_ui()
    def retranslate_ui(self):
        self.setWindowTitle(self.translator.tr('window_title_db_work'))
        self.tabs.setTabText(0, self.translator.tr('tab_balance_units'))
        self.tabs.setTabText(1, self.translator.tr('tab_users_roles'))
        self.tabs.setTabText(2, self.translator.tr('tab_entities'))
        self.tabs.setTabText(3, self.translator.tr('tab_documents'))
        self.tabs.setTabText(4, self.translator.tr('tab_data_view'))
        self.tabs.setTabText(5, self.translator.tr('tab_relations'))
        self.btn_close.setText(self.translator.tr('btn_close'))
        # Обновляем содержимое вкладок, если они имеют метод retranslate_ui
        for i in range(self.tabs.count()):
            widget = self.tabs.widget(i)
            if hasattr(widget, 'retranslate_ui'):
                widget.retranslate_ui()
```

Файл: /home/sergey/Documents/configurate/ide_field_edit.py (относительно: ide_field_edit

```
#ide_field_edit.py
import sys
import re
from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QFormLayout, QLineEdit,
                QComboBox, QCheckBox, QPushButton, QHBoxLayout, QLabel,
                QTextEdit)
from PyQt5.QtCore import Qt
from ide_core.project import goProject
from widgets.reference_editor import ReferenceEditor

class FieldEditDialog(QDialog):
    def __init__(self, field_data=None, status_id=None, read_only_mode=False, db=None):
        super().__init__()
        self.setWindowTitle("Свойства поля (meta.fields)")
        self.setFixedSize(500, 550)
        # Данные: [Имя, ТипКод, Алиас, Описание, Len, Dec, Idx, RefID]
        if not field_data:
            self.data = ["new_field", "C", "Новое поле", "", 255, 0, False, None]
        else:
            self.data = field_data
        self.status_id = status_id
        self.read_only_mode = read_only_mode
        self.db = db or DatabaseService()
        self.init_ui()
```

```
    def init_ui(self):
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(str(self.data[0]))
        self.edit_name.textChanged.connect(self.validate_field_name)
        form.addRow("Имя поля (SQL):", self.edit_name)
        self.combo_type = QComboBox()
        types = goProject.f.execute("SELECT cname, ccode FROM meta.field_types ORDER BY cname")
        for name, code in types:
            self.combo_type.addItem(name, code)
        idx = self.combo_type.findData(str(self.data[1]))
        if idx >= 0: self.combo_type.setCurrentIndex(idx)
        self.combo_type.currentIndexChanged.connect(self.p_CheckType)
        form.addRow("Тип данных:", self.combo_type)
        self.edit_alias = QLineEdit(str(self.data[2]))
        form.addRow("Бизнес-алиас:", self.edit_alias)
        self.edit_comment = QTextEdit()
        self.edit_comment.setMaximumHeight(100)
        form.addRow("Описание:", self.edit_comment)
        # Ссылка на другую сущность (ReferenceEditor)
        self.ref_editor = ReferenceEditor(
            table_name='meta.entitytypes',
            display_field='cname',
            db=self.db
        )
        if len(self.data) > 7 and self.data[7] and str(self.data[7]).isdigit():
            ref_id = int(self.data[7])
            res = self.db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (ref_id,))
            if res:
                self.ref_editor.set_value(ref_id, res[0][0])
        form.addRow("Ссылается на (Ref):", self.ref_editor)
        self.edit_len = QLineEdit(str(self.data[4]))
        form.addRow("Длина (Len):", self.edit_len)
        self.edit_dec = QLineEdit(str(self.data[5]))
        form.addRow("Точность (Dec):", self.edit_dec)
        self.chk_idx = QCheckBox("Создать индекс в БД")
        val_idx = self.data[6]
        self.chk_idx.setChecked(str(val_idx).lower() == 'true' or val_idx is True)
        form.addRow(self.chk_idx)
        # Статус
        self.combo_status = QComboBox()
        statuses = self.db.execute_query("SELECT id, name FROM meta.object_statuses ORDER BY id")
        for sid, sname in statuses:
            self.combo_status.addItem(sname, sid)
        if self.status_id:
            idx_status = self.combo_status.findData(self.status_id)
            if idx_status >= 0:
                self.combo_status.setCurrentIndex(idx_status)
        form.addRow("Статус:", self.combo_status)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton("Принять")
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton("Отмена")
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
        self.p_CheckType()
        # Блокировка для PROD
        if self.read_only_mode or self.status_id == 2:  # PROD
            self.edit_name.setEnabled(False)
            self.combo_type.setEnabled(False)
            self.edit_alias.setEnabled(False)
            self.edit_len.setEnabled(False)
            self.edit_dec.setEnabled(False)
            self.ref_editor.setReadOnly(True)
            self.edit_comment.setEnabled(True)
            self.chk_idx.setEnabled(True)
            self.combo_status.setEnabled(True)
    def validate_field_name(self, text):
        if not text:
            self.edit_name.setStyleSheet("")
            return
        if re.match(r'^[a-zA-Z][a-zA-Z0-9_]*$', text):
            self.edit_name.setStyleSheet("")
        else:
            self.edit_name.setStyleSheet("border: 1px solid red;")
    def p_CheckType(self):
        t = self.combo_type.currentData()
        self.edit_len.setEnabled(t in ('C', 'N'))
        self.edit_dec.setEnabled(t == 'N')
        self.ref_editor.setEnabled(t == 'R')
        self.chk_idx.setEnabled(t not in ('G', 'V', 'F', 'M'))
    def get_data(self):
        return [
            self.edit_name.text(),
            self.combo_type.currentData(),
            self.edit_alias.text(),
            self.edit_comment.toPlainText(),
            int(self.edit_len.text() if self.edit_len.text().isdigit() else 0),
            int(self.edit_dec.text() if self.edit_dec.text().isdigit() else 0),
            self.chk_idx.isChecked(),
            self.ref_editor.get_value(),
            self.combo_status.currentData()
        ]
```

Файл: /home/sergey/Documents/configurate/ide_main.py (относительно: ide_main.py)

```
# ide_main.py
import sys
import os
from PyQt5.QtWidgets import (QApplication, QMainWindow, QAction, QFileDialog,
                             QMessageBox, QStatusBar, QWidget, QVBoxLayout,
                             QHBoxLayout, QFormLayout, QLineEdit, QPushButton,
                             QFrame, QLabel, QGroupBox, QSplitter, QDialog)
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont
from ide_core import goProject
from ide_db_work import DBWorkDialog
from lang.local_translator import LocalTranslator
from database import DatabaseService

class MozartIDE(QMainWindow):
    def __init__(self):
        super().__init__()
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('window_title_main'))
        self.resize(1100, 700)
        # Словарь для хранения действий меню
        self.actions = {}
        # Центральный виджет
        central = QWidget()
        self.setCentralWidget(central)
        main_layout = QHBoxLayout(central)
        main_layout.setContentsMargins(5, 5, 5, 5)
        # Сплиттер
        splitter = QSplitter(Qt.Horizontal)
        main_layout.addWidget(splitter)
        # Левая панель (настройки подключения)
        left = self._create_left_panel()
```

```
    splitter.addWidget(left)
    # Правая панель (кнопки управления)
    right = self._create_right_panel()
    splitter.addWidget(right)
    splitter.setSizes([450, 650])
    # Статус бар
    self.statusBar = QStatusBar()
    self.setStatusBar(self.statusBar)
    self.statusBar.showMessage(self.translator.tr('status_ready'))
    # Меню
    self._init_menu()
    # Переменные состояния
    self.current_project_file = None
    self.buttons_enabled = False
    self.db_work_dialog = None # ссылка на диалог, если открыт
    self.retranslate_ui()
def _init_menu(self):
    menubar = self.menuBar()
    # Меню "Файл"
    file_menu = menubar.addMenu(self.translator.tr('menu_file'))
    self.actions['file_menu'] = file_menu
    new_act = QAction(self.translator.tr('action_new_project'), self)
    new_act.triggered.connect(self.new_project)
    file_menu.addAction(new_act)
    self.actions['new_project'] = new_act
    open_act = QAction(self.translator.tr('action_open_project'), self)
    open_act.triggered.connect(self.open_project)
    file_menu.addAction(open_act)
    self.actions['open_project'] = open_act
    save_act = QAction(self.translator.tr('action_save_project'), self)
    save_act.triggered.connect(self.save_project)
    file_menu.addAction(save_act)
    self.actions['save_project'] = save_act
    file_menu.addSeparator()
    exit_act = QAction(self.translator.tr('action_exit'), self)
    exit_act.triggered.connect(self.close)
    file_menu.addAction(exit_act)
    self.actions['exit'] = exit_act
    # Меню "Язык"
    lang_menu = menubar.addMenu(self.translator.tr('menu_language'))
    self.actions['lang_menu'] = lang_menu
    languages = [
        ('ru', 'Русский'),
        ('en', 'English'),
        ('de', 'Deutsch'),
        ('tt', 'Татарча')
    ]
    for code, name in languages:
        action = QAction(name, self)
        action.setData(code)
        action.triggered.connect(lambda checked, lang=code: self.change_language(lang))
        lang_menu.addAction(action)
    # Меню "Помощь"
    help_menu = menubar.addMenu(self.translator.tr('menu_help'))
    self.actions['help_menu'] = help_menu
    about_act = QAction(self.translator.tr('action_about'), self)
    about_act.triggered.connect(self.about)
    help_menu.addAction(about_act)
    self.actions['about'] = about_act
def change_language(self, lang_code):
    self.translator.set_language(lang_code)
    self.retranslate_ui()
    if self.db_work_dialog and self.db_work_dialog.isVisible():
        self.db_work_dialog.retranslate_ui()
def retranslate_ui(self):
    # Главное окно
    self.setWindowTitle(self.translator.tr('window_title_main'))
    # Левая панель
    self.file_group.setTitle(self.translator.tr('group_project_file'))
    self.conn_group.setTitle(self.translator.tr('group_db_connection'))
    self.btn_open.setText(self.translator.tr('btn_open_project'))
    self.btn_new.setText(self.translator.tr('btn_new_project'))
    self.btn_save.setText(self.translator.tr('btn_save_project'))
    self.btn_connect.setText(self.translator.tr('btn_connect'))
    if self.project_db_conn:
        self.lbl_conn_status.setText(self.translator.tr('status_connected'))
    else:
        self.lbl_conn_status.setText(self.translator.tr('status_disconnected'))
    # Правая панель
    self.db_group.setTitle(self.translator.tr('group_db_work'))
    self.srv_group.setTitle(self.translator.tr('group_server'))
    self.cliente_group.setTitle(self.translator.tr('group_client'))
    self.btn_db_work.setText(self.translator.tr('btn_db_work'))
    self.btn_srv_start.setText(self.translator.tr('btn_start_server'))
    self.btn_srv_stop.setText(self.translator.tr('btn_stop_server'))
    self.btn_client_start.setText(self.translator.tr('btn_start_client'))
    # Меню (QMenu использует setTitle, QAction — setText)
    self.actions['file_menu'].setTitle(self.translator.tr('menu_file'))
    self.actions['new_project'].setText(self.translator.tr('action_new_project'))
    self.actions['open_project'].setText(self.translator.tr('action_open_project'))
    self.actions['save_project'].setText(self.translator.tr('action_save_project'))
    self.actions['exit'].setText(self.translator.tr('action_exit'))
    self.actions['lang_menu'].setTitle(self.translator.tr('menu_language'))
    self.actions['help_menu'].setTitle(self.translator.tr('menu_help'))
    self.actions['about'].setText(self.translator.tr('action_about'))
    # Статус бар
    self.statusBar.showMessage(self.translator.tr('status_ready'))
def _create_left_panel(self):
    panel = QWidget()
    layout = QVBoxLayout(panel)
    layout.setAlignment(Qt.AlignTop)
    # Группа "Файл проекта"
    self.file_group = QGroupBox(self.translator.tr('group_project_file'))
    file_layout = QVBoxLayout()
    self.btn_open = QPushButton()
    self.btn_open.clicked.connect(self.open_project)
    self.btn_new = QPushButton()
    self.btn_new.clicked.connect(self.new_project)
    self.btn_save = QPushButton()
    self.btn_save.clicked.connect(self.save_project)
    file_layout.addWidget(self.btn_open)
    file_layout.addWidget(self.btn_new)
    file_layout.addWidget(self.btn_save)
    self.file_group.setLayout(file_layout)
    layout.addWidget(self.file_group)
    # Разделитель
    line = QFrame()
    line.setFrameShape(QFrame.HLine)
    line.setFrameShadow(QFrame.Sunken)
    layout.addWidget(line)
    # Группа "Подключение к базе данных"
    self.conn_group = QGroupBox(self.translator.tr('group_db_connection'))
    form = QFormLayout()
    self.edit_host = QLineEdit()
    self.edit_port = QLineEdit("5432")
    self.edit_db = QLineEdit()
    self.edit_user = QLineEdit()
    self.edit_pass = QLineEdit()
    self.edit_pass.setEchoMode(QLineEdit.Password)
    form.addRow(self.translator.tr('field_host'), self.edit_host)
    form.addRow(self.translator.tr('field_port'), self.edit_port)
    form.addRow(self.translator.tr('field_db'), self.edit_db)
    form.addRow(self.translator.tr('field_user'), self.edit_user)
```

```
        form.addRow(self.translator.tr('field_password'), self.edit_pass)
        self.btn_connect = QPushButton(self.translator.tr('btn_connect'))
        self.btn_connect.clicked.connect(self.connect_to_db)
        self.btn_connect.setEnabled(False)
        form.addRow(self.btn_connect)
        self.conn_group.setLayout(form)
        layout.addWidget(self.conn_group)
        # Информация о статусе подключения
        self.lbl_conn_status = QLabel(self.translator.tr('status_disconnected'))
        self.lbl_conn_status.setStyleSheet("color: red;")
        layout.addWidget(self.lbl_conn_status)
        layout.addStretch()
        return panel
    def _create_right_panel(self):
        panel = QWidget()
        layout = QVBoxLayout(panel)
        layout.setAlignment(Qt.AlignTop)
        # Группа "Работа с БД"
        self.db_group = QGroupBox(self.translator.tr('group_db_work'))
        db_layout = QVBoxLayout()
        self.btn_db_work = QPushButton(self.translator.tr('btn_db_work'))
        self.btn_db_work.clicked.connect(self.open_db_work)
        self.btn_db_work.setEnabled(False)
        db_layout.addWidget(self.btn_db_work)
        self.db_group.setLayout(db_layout)
        layout.addWidget(self.db_group)
        # Группа "Сервер приложений"
        self.srv_group = QGroupBox(self.translator.tr('group_server'))
        srv_layout = QVBoxLayout()
        self.btn_srv_start = QPushButton(self.translator.tr('btn_start_server'))
        self.btn_srv_stop = QPushButton(self.translator.tr('btn_stop_server'))
        self.btn_srv_start.setEnabled(False)
        self.btn_srv_stop.setEnabled(False)
        srv_layout.addWidget(self.btn_srv_start)
        srv_layout.addWidget(self.btn_srv_stop)
        self.srv_group.setLayout(srv_layout)
        layout.addWidget(self.srv_group)
        # Группа "Клиент"
        self.client_group = QGroupBox(self.translator.tr('group_client'))
        client_layout = QVBoxLayout()
        self.btn_client_start = QPushButton(self.translator.tr('btn_start_client'))
        self.btn_client_start.setEnabled(False)
        client_layout.addWidget(self.btn_client_start)
        self.client_group.setLayout(client_layout)
        layout.addWidget(self.client_group)
        layout.addStretch()
        return panel
    # --- Методы работы с проектом ---
    def new_project(self):
        self.current_project_file = None
        self.clear_connection_fields()
        self.btn_connect.setEnabled(True)
        self.statusBar.showMessage(self.translator.tr('status_new_project'))
    def open_project(self):
        path, _ = QFileDialog.getOpenFileName(
            self, self.translator.tr('dialog_open_project'), "",
            "Mozart Project (*.mzt);;All files (*)"
        )
        if not path:
            return
        ok, msg = goProject.p_LoadProject(path)
        if ok:
            self.current_project_file = path
            self.update_connection_fields()
            self.btn_connect.setEnabled(True)
            self.statusBar.showMessage(self.translator.tr('status_project_loaded') + f" {os.path.basename(path)}")
        else:
            QMessageBox.critical(self, self.translator.tr('error_loading'), msg)
    def save_project(self):
        if not self.current_project_file:
            path, _ = QFileDialog.getSaveFileName(
                self, self.translator.tr('dialog_save_project'), "",
                "Mozart Project (*.mzt)"
            )
            if path:
                if not path.lower().endswith(".mzt"):
                    path += ".mzt"
                self.current_project_file = path
        if self.current_project_file:
            goProject.p_SaveProject(self.current_project_file)
            self.statusBar.showMessage(self.translator.tr('status_project_saved') + f"
{os.path.basename(self.current_project_file)}")
    # --- Работа с подключением ---
    def _clear_connection_fields(self):
        self.edit_host.clear()
        self.edit_port.setText("5432")
        self.edit_db.clear()
        self.edit_user.clear()
        self.edit_pass.clear()
    def _update_connection_fields(self):
        self.edit_host.setText(goProject.db_host)
        self.edit_port.setText(str(goProject.db_port))
        self.edit_db.setText(goProject.db_name)
        self.edit_user.setText(goProject.db_user)
        self.edit_pass.setText(goProject.db_pass)
    def connect_to_db(self):
        goProject.db_host = self.edit_host.text().strip()
        goProject.db_port = self.edit_port.text().strip()
        goProject.db_name = self.edit_db.text().strip()
        goProject.db_user = self.edit_user.text().strip()
        goProject.db_pass = self.edit_pass.text()
        ok, msg = goProject.p_Connect()
        if ok:
            # Синхронизируем локальную базу переводов с PostgreSQL
            db_service = DatabaseSetupDialog(self)  # используем тот же класс, что и в других частях
            db_service.p_SyncTranslations(goProject.db_service)
            # Принудительно устанавливаем язык (например, русский)
            self.translator.set_language('ru')
            self.retranslate_ui()
            self.lbl_conn_status.setText(self.translator.tr('status_connected'))
            self.lbl_conn_status.setStyleSheet("color: green;")
            self.statusBar.showMessage(self.translator.tr('status_connected'))
            self.btn_db_work.setEnabled(True)
            self.btn_srv_start.setEnabled(True)
            self.btn_client_start.setEnabled(True)
        else:
            self.lbl_conn_status.setText(self.translator.tr('status_disconnected'))
            self.lbl_conn_status.setStyleSheet("color: red;")
            QMessageBox.warning(self, self.translator.tr('error'), msg)
            self.btn_db_work.setEnabled(False)
            self.btn_srv_start.setEnabled(False)
            self.btn_client_start.setEnabled(False)
    def open_db_work(self):
        if self.db_work_dialog is None or not self.db_work_dialog.isVisible():
            self.db_work_dialog = DBWorkDialog(self)
        self.db_work_dialog.show()
        self.db_work_dialog.raise_()
        self.db_work_dialog.activateWindow()
    def about(self):
        QMessageBox.about(self, self.translator.tr('action_about'),
                          "MOZART ERP Configurator\nВерсия 0.1\nРазработано на PyQt5")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MozartIDE()
    window.show()
    sys.exit(app.exec_())
```

Файл: /home/sergey/Documents/configurate/lang/local_translator.py (относительно: lang/l

```
# lang/local_translator.py
import sqlite3
import threading
import os
import sys
class LocalTranslator:
    _instance = None
    _lock = threading.Lock()
    def __new__(cls):
        if cls._instance is None:
            with cls._lock:
                if cls._instance is None:
                    cls._instance = super().__new__(cls)
                    cls._instance._initialized = False
        return cls._instance
    def __init__(self):
        if self._initialized:
            return
        project_dir = os.path.dirname(os.path.abspath(sys.argv[0]))
        lang_dir = os.path.join(project_dir, 'lang_data')
        os.makedirs(lang_dir, exist_ok=True)
        self.db_path = os.path.join(lang_dir, 'lang.db')
        self._init_db()
        self.current_language = 'ru'
        self._initialized = True
    def _init_db(self):
        conn = sqlite3.connect(self.db_path)
        cur = conn.cursor()
        cur.execute(
            '''CREATE TABLE IF NOT EXISTS translations (
                label_key TEXT NOT NULL,
                language TEXT NOT NULL,
                translation TEXT NOT NULL,
                PRIMARY KEY (label_key, language)
            )'''
        )
        cur.execute("SELECT COUNT(*) FROM translations WHERE language='en'")
        if cur.fetchone()[0] == 0:
            self._load_fallback(conn)
        conn.commit()
        conn.close()
    def _get_fallback(self):
        return {
            'ru': {
                'window_title_main': 'MOZART ERP - Конфигуратор',
                'menu_file': 'Файл',
                'menu_language': 'Язык',
                'menu_help': 'Помощь',
                'action_new_project': 'Новый проект...',
                'action_open_project': 'Открыть проект...',
                'action_save_project': 'Сохранить параметры',
                'action_exit': 'Выход',
                'action_about': 'О программе',
                'group_project_file': 'Файл проекта',
                'group_db_connection': 'Подключение к базе данных',
                'group_db_work': 'Работа с БД',
                'group_server': 'Сервер приложений',
                'group_client': 'Клиент',
                'btn_open_project': 'Открыть проект...',
                'btn_new_project': 'Новый проект',
                'btn_save_project': 'Сохранить параметры',
                'btn_connect': 'Подключиться',
                'btn_db_work': 'Работа с БД',
                'btn_start_server': 'Запустить сервер',
                'btn_stop_server': 'Остановить сервер',
                'btn_start_client': 'Запустить клиент',
                'btn_close': 'Закрыть',
                'btn_add': 'Добавить',
                'btn_edit': 'Редактировать',
                'btn_delete': 'Удалить',
                'btn_ok': 'OK',
                'btn_cancel': 'Отмена',
                'status_ready': 'Готов. Откройте или создайте проект.',
                'status_connected': 'Подключение к БД успешно',
                'status_disconnected': 'Не подключено',
                'field_host': 'Хост',
                'field_port': 'Порт',
                'field_db': 'База данных',
                'field_user': 'Пользователь',
                'field_password': 'Пароль',
                'field_name': 'Имя',
                'field_alias': 'Алиас',
                'field_id': 'ID',
                'field_active': 'Активна',
                'error': 'Ошибка',
                'info': 'Информация',
                'warning': 'Предупреждение',
                'dialog_load_project': 'Открыть загрузки',
                'dialog_open_project': 'Открыть проект',
                'dialog_save_project': 'Сохранить проект',
                'status_new_project': 'Новый проект создан. Заполните параметры БД и подключитесь.',
                'status_project_loaded': 'Проект загружен.',
                'status_project_saved': 'Проект сохранен.',
                'title_edit_balance_unit': 'Редактирование балансовой единицы',
                'label_is_base_unit': 'Является балансовой единицей',
                'field_parent': 'Родитель',
                'tab_as_parent': 'Как родитель',
                'tab_as_child': 'Как ребенок',
                'field_detail': 'Деталь',
                'field_master': 'Мастер',
                'field_role': 'Роль',
                'field_link_field': 'Поле связи',
                'field_order': 'Порядок',
                'title_add_visibility': 'Добавить видимость БЕ',
                'field_balance_unit': 'Балансовая единица',
                'field_policy': 'Политика',
                'policy_individual': 'Индивидуальная',
                'policy_all_visible': 'Все видны',
                'error_visibility_exists': 'Такая запись уже существует',
                'warning_select_entity': 'Выберите сущность',
                'warning_select_field': 'Выберите поле',
                'error_prod_field_delete': 'Поля в продуктивном статусе нельзя удалить',
                'confirm_delete_fields': 'Удалить выбранные поля (%d шт.)?',
                'confirm_delete_records': 'Удалить выбранные записи (%d шт.)?',
                'tab_users': 'Пользователи',
                'tab_roles': 'Роли',
                'tab_context': 'Контекст',
                'tab_users_roles': 'Пользователи и роли',
                'title_user_edit': 'Редактирование пользователя',
                'title_role_edit': 'Редактирование роли',
                'title_context_assign': 'Назначение контекста',
                'field_login': 'Логин',
                'field_password_hash': 'Хеш пароля',
                'field_person_instance': 'Экземпляр персоны',
                'label_active': 'Активен',
                'label_interface_role': 'Интерфейсная роль',
                'field_parent_role': 'Родительская роль',
                'field_interface_path': 'Путь к интерфейсу',
                'field_comment': 'Комментарий',
                'field_user': 'Пользователь',
                'field_balance_unit': 'Балансовая единица',
                'warning_select_user': 'Выберите пользователя',
                'warning_select_role': 'Выберите роль',
                'warning_select_context': 'Выберите запись контекста',
                'warning_context_required': 'Необходимо наличие пользователей, ролей и БЕ',
                'confirm_delete': 'Подтверждение удаления',
                'confirm_delete_user': 'Удалить пользователя?',
                'confirm_delete_role': 'Удалить роль?',
                'confirm_delete_context': 'Удалить контекст?',
                'field_interface': 'Интерфейсная',
                'warning_select_record': 'Выберите запись',
                # Новые ключи для entities и вкладок
                'tab_types': 'Типы',
                'title_entity_type_edit': 'Редактирование типа сущности',
                'field_system_name': 'Системное имя',
                'field_description': 'Описание',
                'field_entity_class': 'Класс сущности',
                'field_baseclass_legacy': 'Базовый класс (устар.)',
                'label_dependent': 'Зависимая',
                'label_hierarchical': 'Иерархическая',
                'label_versioned': 'Версионная',
                'field_security_strategy': 'Стратегия безопасности',
```

```
            'field_status': 'Статус',
            'tab_fields': 'Поля',
            'tab_bu_visibility': 'Видимость БЕ',
            'tab_composition': 'Подчиненные таблицы',
            'tab_routes': 'Маршруты',
            'field_type': 'Тип',
            'field_decimals': 'Точность',
            'field_index': 'Индекс',
            'field_ref_type': 'Ссылка на тип',
            'routes_in_development': 'Раздел в разработке: маршруты для документов',
            'warning_select_record': 'Выберите запись',
            'title_add_composition': 'Добавить подчиненную таблицу',
            'title_edit_composition': 'Редактировать связь',
'info_add_as_child_not_supported': 'Добавление как ребёнка не поддерживается (создаётся в родителе)',
            'error_composition_exists': 'Такая связь уже существует',
            'confirm_rename': 'Подтверждение переименования',
'confirm_rename_message': 'Изменение системного имени с \'{0}\' на \'{1}\' переименует таблицу расширения. Продолжить?',
            'info_table_renamed': 'Таблица ext.{0} успешно переименована в ext.{1}',
            'error_rename_table': 'Не удалось переименовать таблицу: {0}',
            'warning_table_not_found': 'Таблица ext.{0} не найдена, переименование не требуется',
            'error_rename_prod': 'Нельзя изменить системное имя для сущности в продуктивном статусе',
            'error_prod_entity_delete': 'Сущности в продуктивном статусе нельзя удалить: {0}',
            'confirm_delete_entities': 'Удалить выбранные сущности (%d шт.)?',
            'error_save_field': 'Ошибка сохранения поля:',
        },
        'en': {
            'window_title_main': 'MOZART ERP - Configurator',
            'menu_file': '&File',
            'menu_language': '&Language',
            'menu_help': '&Help',
            'action_new_project': 'New Project...',
            'action_open_project': 'Open Project...',
            'action_save_project': 'Save Settings',
            'action_exit': 'Exit',
            'action_about': 'About',
            'group_project_file': 'Project File',
            'group_db_connection': 'Database Connection',
            'group_db_work': 'Database Work',
            'group_server': 'Application Server',
            'group_client': 'Client',
            'btn_open_project': 'Open Project...',
            'btn_new_project': 'New Project...',
            'btn_save_project': 'Save Settings',
            'btn_connect': 'Connect',
            'btn_db_work': 'Database Work',
            'btn_start_server': 'Start Server',
            'btn_stop_server': 'Stop Server',
            'btn_start_client': 'Start Client',
            'btn_close': 'Close',
            'btn_add': 'Add',
            'btn_edit': 'Edit',
            'btn_delete': 'Delete',
            'btn_ok': 'OK',
            'btn_cancel': 'Cancel',
            'status_ready': 'Ready. Open or create a project.',
            'status_connected': 'Database connection successful',
            'status_disconnected': 'Not connected',
            'field_host': 'Host',
            'warning_select_record': 'Select a record',
            'field_port': 'Port',
            'field_db': 'Database',
            'field_user': 'User',
            'field_password': 'Password',
            'field_name': 'Name',
            'field_alias': 'Alias',
            'field_id': 'ID',
            'field_active': 'Active',
            'error': 'Error',
            'info': 'Information',
            'warning': 'Warning',
            'error_loading': 'Loading error',
            'dialog_open_project': 'Open Project',
            'dialog_save_project': 'Save Project',
            'status_new_project': 'New project created. Fill in database parameters and connect.',
            'status_project_loaded': 'Project loaded:',
            'status_project_saved': 'Project saved:',
            'title_edit_balance_unit': 'Edit Balance Unit',
            'label_is_balance_unit': 'Is balance unit',
            'field_parent': 'Parent',
            'tab_users': 'Users',
            'tab_roles': 'Roles',
            'tab_context': 'Context',
            'tab_users_roles': 'Users & Roles',
            'title_user_edit': 'Edit User',
            'title_role_edit': 'Edit Role',
            'title_context_assign': 'Assign Context',
            'field_login': 'Login',
            'field_password_hash': 'Password hash',
            'field_person_instance': 'Person instance',
            'label_active': 'Active',
            'label_interface_role': 'Interface role',
            'field_parent_role': 'Parent role',
            'field_interface_path': 'Interface path',
            'field_comment': 'Comment',
            'field_user': 'User',
            'field_role': 'Role',
            'field_balance_unit': 'Balance unit',
            'warning_select_user': 'Select a user',
            'warning_select_role': 'Select a role',
            'warning_select_context': 'Select a context entry',
            'warning_context_required': 'Users, roles and balance units are required',
            'confirm_delete': 'Confirm deletion',
            'confirm_delete_user': 'Delete user?',
            'confirm_delete_role': 'Delete role?',
            'confirm_delete_context': 'Delete context?',
            'field_interface': 'Interface',
            'tab_as_parent': 'As parent',
            'tab_as_child': 'As child',
            'field_detail': 'Detail',
            'field_master': 'Master',
            'field_role': 'Role',
            'field_link_field': 'Link field',
            'field_order': 'Order',
            'title_add_visibility': 'Add BU visibility',
            'policy_individual': 'Individual',
            'policy_all_visible': 'All visible',
            'policy_all_hidden': 'All hidden',
            'error_visibility_exists': 'Record already exists',
            'warning_select_entity': 'Select entity',
            'warning_select_field': 'Select field',
            'error_prod_field_delete': 'Fields in PROD status cannot be deleted',
            'confirm_delete_fields': 'Delete selected fields (%d items)?',
            'confirm_delete_records': 'Delete selected records (%d items)?',
            # New keys for entities and tabs
            'entity_types': 'Entity types',
            'title_entity_type_edit': 'Edit Entity Type',
            'field_system_name': 'System name',
            'field_description': 'Description',
            'field_entity_class': 'Entity class',
            'field_baseclass_legacy': 'Base class (legacy)',
            'label_dependent': 'Dependent',
            'label_hierarchical': 'Hierarchical',
            'label_versioned': 'Versioned',
            'field_security_strategy': 'Security strategy',
            'field_status': 'Status',
            'tab_fields': 'Fields',
            'tab_bu_visibility': 'BU Visibility',
            'tab_composition': 'Subordinate tables',
            'tab_routes': 'Routes',
            'field_type': 'Type',
            'field_decimals': 'Decimals',
            'field_index': 'Index',
            'field_ref_type': 'Ref type',
            'routes_in_development': 'Section under development: document routes',
            'warning_select_record': 'Select a record',
            'title_add_composition': 'Add subordinate table',
            'title_edit_composition': 'Edit composition',
            'info_add_as_child_not_supported': 'Adding as child is not supported (create in parent)',
            'error_composition_exists': 'Such composition already exists',
            'confirm_rename': 'Confirm rename',
'confirm_rename_message': 'Changing system name from \'{0}\' to \'{1}\' will rename the extension table. Continue?',
            'info_table_renamed': 'Table ext.{0} successfully renamed to ext.{1}',
            'error_rename_table': 'Failed to rename table: {0}',
            'warning_table_not_found': 'Table ext.{0} not found, rename not required',
            'error_prod_entity_delete': 'Entities in PROD status cannot be deleted: {0}',
            'confirm_delete_entities': 'Delete selected entities (%d items)?',
            'error_save_field': 'Error saving field:
```

```
        }
    }
    def _load_fallback(self, conn):
        cur = conn.cursor()
        for lang, data in self._get_fallback().items():
            for key, text in data.items():
                cur.execute("INSERT OR IGNORE INTO translations (label_key, language, translation) VALUES (?, ?, ?)",
                            (key, lang, text))
        conn.commit()
    def set_language(self, lang_code):
        self.current_language = lang_code
    def tr(self, key):
        conn = sqlite3.connect(self.db_path)
        cur = conn.cursor()
        cur.execute("SELECT translation FROM translations WHERE label_key=? AND language=?", (key,
self.current_language))
        row = cur.fetchone()
        if row:
            conn.close()
            return row[0]
        cur.execute("SELECT translation FROM translations WHERE label_key=? AND language='en'", (key,))
        row = cur.fetchone()
        conn.close()
        return row[0] if row else key
    def sync_from_postgres(self, db_service):
        rows = db_service.execute_query("SELECT label_key, language, translation FROM lang.translations")
        if not rows:
            return
        conn = sqlite3.connect(self.db_path)
        cur = conn.cursor()
        for key, lang, text in rows:
            cur.execute("INSERT OR REPLACE INTO translations (label_key, language, translation) VALUES (?, ?, ?)",
                        (key, lang, text))
        conn.commit()
        conn.close()
```

Файл: /home/sergey/Documents/configurate/lang/translator.py (относительно: lang/transla

```
# lang/translator.py
import threading
from database import DatabaseService
class Translator:
    _instance = None
    _lock = threading.Lock()
    def __new__(cls):
        if cls._instance is None:
            with cls._lock:
                if cls._instance is None:
                    cls._instance = super().__new__(cls)
                    cls._instance._initialized = False
        return cls._instance
    def __init__(self):
        if self._initialized:
            return
        self.db = DatabaseService()
        self.current_language = 'ru'
        self._cache = {}
        self._loaded = False
        self._initialized = True
    def _load_all(self):
        rows = self.db.execute_query("SELECT label_key, language, translation FROM lang.translations")
        for key, lang, text in rows:
            self._cache[(key, lang)] = text
        self._loaded = True
    def set_language(self, lang_code):
        self.current_language = lang_code
    def tr(self, key):
        if not self._loaded:
            self._load_all()
        if (key, self.current_language) in self._cache:
            return self._cache[(key, self.current_language)]
        if (key, 'en') in self._cache:
            return self._cache[(key, 'en')]
        return key
    def refresh(self):
        self._cache.clear()
        self._loaded = False
        # переводы будут перезагружены при следующем вызове tr()
```

Файл: /home/sergey/Documents/configurate/layer2_server/server.py (относительно: layer2

Файл: /home/sergey/Documents/configurate/layer3_client/starter.py (относительно: layer3

Файл: /home/sergey/Documents/configurate/makeJson.py (относительно: makeJson.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Скрипт для выгрузки структуры и содержимого текстовых файлов проекта в JSON.
Корень сканирования - директория, в которой находится сам скрипт.
Результат - project_dump.json в той же директории.
"""
import os
import json
import sys
from pathlib import Path
# Расширения текстовых файлов, которые включаем
TEXT_EXTENSIONS = {
    '.py', '.pyw', '.pyx', '.pxd', '.pyi',
    '.db', '.sqlite', '.ddl',
    '.html', '.htm', '.xhtml',
    '.js', '.jsx', '.ts', '.tsx', '.vue',
    '.css', '.scss', '.sass', '.less',
    '.json', '.yaml', '.yml', '.toml', '.ini', '.cfg', '.conf',
    '.md', '.markdown', '.rst', '.txt',
    '.sh', '.bash', '.zsh', '.bat', '.cmd', '.ps1',
    '.csv', '.tsv', '.h', '.hpp', '.cc', '.cxx',
    '.java', '.kt', '.kts', '.scala',
    '.go', '.rs', '.swift',
    '.tex', '.latex',
    '.bat',
    '.desktop', '.service', '.timer',
    '.dll',
    '.ps1', '.psm1', '.psd1',
}
# Расширения, которые игнорируем целиком (бинарные, архивы, медиа)
IGNORE_EXTENSIONS = {'.exe', '.bin',
    '.pyc', '.jpeg', '.jpg', '.gif', '.bmp', '.ico', '.webp',
    '.mp4', '.mp3', '.wav', '.ogg', '.flac', '.avi', '.mov', '.mkv',
    '.zip', '.tar', '.gz', '.bz2', '.xz', '.7z', '.rar',
    '.pdf', '.docx', '.xlsx', '.pptx',
    '.db', '.sqlite', '.sqlite3',
}
# Имена папок, которые не нужно обходить
IGNORE_DIRS = {'.git', '.svn', '.hg', '.idea', '.vscode',
    '__pycache__', '.venv', '.env', '.venv', '.env', 'virtualenv',
    'node_modules', 'dist', 'build', '.mypy_cache', '.pytest_cache', '.tox',
```

```
    '.ipynb_checkpoints', '.DS_Store', 'Thumbs.db', '.trash', '.tmp',
    'logs', 'backup', 'old', '__pycache__',
    # Если у вас есть папка с большими временными файлами, добавьте сюда
}
def should_include_file(file_path):
    """Проверяет, нужно ли включать файл в дамп."""
    ext = os.path.splitext(file_path)[1].lower()
    if ext in IGNORE_EXTENSIONS:
        return False
    if ext in TEXT_EXTENSIONS:
        return True
    # Если расширение не в списке, можно добавить проверку на текстовый файл с риском
    # Но лучше перечислить нужные расширения явно
    return False
def read_file_content(file_path):
    """Пытается прочитать файл как текст (UTF-8 или cp1251)."""
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            return f.read()
    except UnicodeDecodeError:
        try:
            with open(file_path, r', encoding='cp1251') as f:
                return f.read()
        except:
            return f"[Binary or unsupported encoding: {file_path}]"
    except Exception as e:
        return f"[Error reading file: {e}]"
def collect_directory(root_dir, base_path):
    """
    Рекурсивно обходит директорию и возвращает словарь:
    - если элемент файл -> записывает содержимое
    - если директория -> рекурсивный вызов
    """
    result = {}
    # Сортируем для стабильности порядка
    for entry in sorted(os.listdir(root_dir)):
        full_path = os.path.join(root_dir, entry)
        rel_path = os.path.relpath(full_path, base_path)
        if entry in IGNORE_DIRS:
            continue
        if os.path.isdir(full_path):
            sub = collect_directory(full_path, base_path)
            if sub:
                result[entry] = sub
        else:
            if should_include_file(entry):
                content = read_file_content(full_path)
                # Сохраняем как строку (можно также добавить мета-информацию)
                result[entry] = content
            # Иначе файл пропускаем (не включаем даже placeholder)
    return result
def main():
    # Определяем корень сканирования: директория, где находится скрипт
    script_dir = os.path.dirname(os.path.abspath(__file__))
    root_path = script_dir
    # Проверяем, что директория существует
    if not os.path.isdir(root_path):
        print(f"Ошибка: директория {root_path} не существует")
        sys.exit(1)
    output_file = os.path.join(root_path, "project_dump.json")
    print(f"Сканируем: {root_path}")
    print(f"Игнорируемые папки: {', '.join(sorted(IGNORE_DIRS))}")
    print("Ждите, это может занять некоторое время...")
    structure = collect_directory(root_path, root_path)
    print(f"Запись в файл: {output_file}")
    with open(output_file, 'w', encoding='utf-8') as f:
        json.dump(structure, f, ensure_ascii=False, indent=2)
    size = os.path.getsize(output_file)
    print(f"Готово! Размер файла: {size} байт ({size / 1024:.2f} КБ)")
    print(f"Файл {output_file} можно отправлять ассистенту.")
if __name__ == "__main__":
    main()
```

Файл: /home/sergey/Documents/configurate/makeJsonDB.py (относительно: makeJsonDB.p

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Расширенный дамп структуры PostgreSQL и выборочно данных.
Работает как из командной строки, так и из IDE (задать параметры в секции DEFAULT_RUNTIME_CONFIG).
"""
import json
import sys
import argparse
import psycopg2
from psycopg2 import sql
from datetime import date, datetime, time
from decimal import Decimal
# ========================= НАСТРОЙКИ ПО УМОЛЧАНИЮ (для запуска из IDE) =========================
# Отредактируйте эти переменные, если запускаете скрипт без аргументов командной строки.
DEFAULT_RUNTIME_CONFIG = {
    'all': 1,                  # 1 - только структура, 0: данные из meta/auth/autodoc/lang/public, 1: все данные
    'schemas': {               # 1 - обрабатывать схему, 0 - пропустить (None - использовать значение по умолчанию)
        'ext': None,           # None означает: обрабатывать, т.к. схема в ALL_SCHEMAS
        'meta': None,
        'data': None,
        'auth': None,
        'autodoc': None,
        'lang': None,
        'public': None,
    },
    'with_data_tables': [],    # список строк 'schema.table' для принудительной выгрузки данных
    'output_file': 'pg_schema_dump_full.json'
}
# ================================================================================================
# Все возможные схемы
ALL_SCHEMAS = ['ext', 'meta', 'data', 'auth', 'autodoc', 'lang', 'public']
# Конфигурация подключения (можно переопределить через аргумент config_file)
DEFAULT_DB_CONFIG = {
    "host": "localhost",
    "port": "5432",
    "dbname": "mozart_erp",
    "user": "postgres",
    "password": "SQLSerIra"
}
def get_connection(config):
    return psycopg2.connect(**config)
def json_serializer(obj):
    if isinstance(obj, (datetime, date)):
        return obj.isoformat()
    if isinstance(obj, time):
        return obj.strftime('%H:%M:%S')
    if isinstance(obj, Decimal):
        return float(obj)
    raise TypeError(f"Object of type {type(obj)} is not JSON serializable")
def safe_fetch(cursor, query, params=None):
    try:
        cursor.execute(query, params)
        return cursor.fetchall()
    except Exception as e:
        print(f"Ошибка запроса: {e}")
        return []
def get_schema_comment(cursor, schema_name):
    rows = safe_fetch(cursor, "SELECT obj_description(oid) FROM pg_namespace WHERE nspname = %s",
(schema_name,))
    return rows[0][0] if rows else None
```

```
def get_tables_metadata(cursor, schema_name):
    tables = []
    rows = safe_fetch(cursor, """
        SELECT t.table_name
        FROM information_schema.tables t
        WHERE t.table_schema = %s AND t.table_type = 'BASE TABLE'
    """, (schema_name,))
    for table_name, in rows:
        comment = None
        try:
            cursor.execute("SELECT obj_description('%s.%s'::regclass)", (schema_name, table_name))
            row = cursor.fetchone()
            if row:
                comment = row[0]
        except:
            pass
        columns = []
        col_rows = safe_fetch(cursor, """
            SELECT c.column_name, c.data_type, c.is_nullable, c.column_default
            FROM information_schema.columns c
            WHERE c.table_schema = %s AND c.table_name = %s
            ORDER BY c.ordinal_position
        """, (schema_name, table_name))
        for col_name, data_type, is_nullable, col_default = col_rows:
            col_comment = None
            try:
                cursor.execute("""
                    SELECT col_description(('%s.%s'::regclass), ordinal_position)
                    FROM information_schema.columns
                    WHERE table_schema = %s AND table_name = %s AND column_name = %s
                """, (schema_name, table_name, schema_name, table_name, col_name))
                cc = cursor.fetchone()
                if cc:
                    col_comment = cc[0]
            except:
                pass
            columns.append({
                "name": col_name,
                "data_type": data_type,
                "nullable": is_nullable == 'YES',
                "default": col_default,
                "comment": col_comment
            })
        pk_rows = safe_fetch(cursor, """
            SELECT kcu.column_name
            FROM information_schema.table_constraints tc
            JOIN information_schema.key_column_usage kcu
                ON tc.constraint_name = kcu.constraint_name
                AND tc.table_schema = kcu.table_schema
            WHERE tc.table_schema = %s AND tc.table_name = %s
                AND tc.constraint_type = 'PRIMARY KEY'
            ORDER BY kcu.ordinal_position
        """, (schema_name, table_name))
        pk = [row[0] for row in pk_rows]
        fk_rows = safe_fetch(cursor, """
            SELECT kcu.column_name, ccu.table_schema, ccu.table_name, ccu.column_name
            FROM information_schema.table_constraints tc
            JOIN information_schema.key_column_usage kcu
                ON tc.constraint_name = kcu.constraint_name
                AND tc.table_schema = kcu.table_schema
            JOIN information_schema.constraint_column_usage ccu
                ON ccu.constraint_name = tc.constraint_name
                AND ccu.table_schema = tc.table_schema
            WHERE tc.table_schema = %s AND tc.table_name = %s
                AND tc.constraint_type = 'FOREIGN KEY'
        """, (schema_name, table_name))
        fk = [{"column": r[0], "ref_table": f"{r[1]}.{r[2]}", "ref_column": r[3]} for r in fk_rows]
        unique_rows = safe_fetch(cursor, """
            SELECT kcu.column_name
            FROM information_schema.table_constraints tc
            JOIN information_schema.key_column_usage kcu
                ON tc.constraint_name = kcu.constraint_name
                AND tc.table_schema = kcu.table_schema
            WHERE tc.table_schema = %s AND tc.table_name = %s
                AND tc.constraint_type = 'UNIQUE'
            ORDER BY kcu.ordinal_position
        """, (schema_name, table_name))
        unique = [row[0] for row in unique_rows]
        tables.append({
            "name": table_name,
            "comment": comment,
            "columns": columns,
            "primary_key": pk,
            "foreign_keys": fk,
            "unique_constraints": unique
        })
    return tables
def get_views_metadata(cursor, schema_name):
    views = []
    rows = safe_fetch(cursor, """
        SELECT v.table_name, v.view_definition
        FROM information_schema.views v
        WHERE v.table_schema = %s
        ORDER BY v.table_name
    """, (schema_name,))
    for view_name, definition in rows:
        comment = None
        try:
            cursor.execute("SELECT obj_description('%s.%s'::regclass)", (schema_name, view_name))
            row = cursor.fetchone()
            if row:
                comment = row[0]
        except:
            pass
        columns = []
        col_rows = safe_fetch(cursor, """
            SELECT c.column_name, c.data_type
            FROM information_schema.columns c
            WHERE c.table_schema = %s AND c.table_name = %s
            ORDER BY c.ordinal_position
        """, (schema_name, view_name))
        columns = [{"name": r[0], "data_type": r[1]} for r in col_rows]
        views.append({
            "name": view_name,
            "definition": definition,
            "columns": columns,
            "comment": comment
        })
    return views
def get_functions_metadata(cursor, schema_name):
    functions = []
    rows = safe_fetch(cursor, """
        SELECT p.proname, pg_get_function_arguments(p.oid), pg_get_function_result(p.oid),
            pg_get_functiondef(p.oid), obj_description(p.oid), p.prokind
        FROM pg_proc p
        JOIN pg_namespace n ON p.pronamespace = n.oid
        WHERE n.nspname = %s
        ORDER BY p.proname
    """, (schema_name,))
    for row in rows:
        functions.append({
            "name": row[0],
            "arguments": row[1],
            "result_type": row[2] if row[2] else "void",
            "definition": row[3],
            "comment": row[4],
            "kind": "procedure" if row[5] == 'p' else "function"
        })
    return functions
def get_triggers_metadata(cursor, schema_name):
    triggers = []
    rows = safe_fetch(cursor, """
        SELECT t.tgname, t.relname, t.tgtype, t.tgfoid::regproc, pg_get_triggerdef(t.oid)
        FROM pg_trigger t
        JOIN pg_class c ON t.tgrelid = c.oid
        JOIN pg_namespace n ON c.relnamespace = n.oid
        WHERE n.nspname = %s AND NOT t.tgisinternal
        ORDER BY t.tgname
    """, (schema_name,))
    for tgname, relname, tgtype, func_name, definition in rows:
        event_parts = []
        if tgtype & 1: event_parts.append("ROW")
        if tgtype & 2: event_parts.append("BEFORE")
        if tgtype & 4: event_parts.append("AFTER")
        if tgtype & 8: event_parts.append("INSERT")
```

```
        if tgtype & 16: event_parts.append("DELETE")
        if tgtype & 32: event_parts.append("TRUNCATE")
        event_str = ",".join(event_parts) if event_parts else "UNKNOWN"
        triggers.append({
            "name": tgname,
            "table": f"{schema_name}.{relname}",
            "event": event_str,
            "function": func_name,
            "definition": definition
        })
    return triggers
def get_indexes_metadata(cursor, schema_name):
    indexes = []
    rows = safe_fetch(cursor, """
        SELECT i.relname, t.relname, pg_get_indexdef(idx.indexrelid), idx.indisunique, am.amname
        FROM pg_index idx
        JOIN pg_class i ON idx.indexrelid = i.oid
        JOIN pg_class t ON idx.indrelid = t.oid
        JOIN pg_namespace n ON t.relnamespace = n.oid
        JOIN pg_am am ON i.relam = am.oid
        WHERE n.nspname = %s
        ORDER BY t.relname
    """, (schema_name,))
    for idx_name, table_name, definition, is_unique, method in rows:
        indexes.append({
            "name": idx_name,
            "table": f"{schema_name}.{table_name}",
            "definition": definition,
            "unique": is_unique,
            "method": method
        })
    return indexes
def get_rules_metadata(cursor, schema_name):
    rules = []
    rows = safe_fetch(cursor, """
        SELECT r.rulename, relname, pg_get_ruledef(r.oid)
        FROM pg_rewrite r
        JOIN pg_class c ON r.ev_class = c.oid
        JOIN pg_namespace n ON c.relnamespace = n.oid
        WHERE n.nspname = %s
        AND r.rulename NOT LIKE '_RETURN'
        ORDER BY r.rulename
    """, (schema_name,))
    for rulename, relname, definition in rows:
        rules.append({
            "name": rulename,
            "table": f"{schema_name}.{relname}",
            "definition": definition
        })
    return rules
def get_table_data(cursor, schema_name, table_name, limit=500):
    try:
        cursor.execute(sql.SQL("SELECT * FROM {}.{} LIMIT %s").format(
            sql.Identifier(schema_name),
            sql.Identifier(table_name)
        ), (limit,))
        rows = cursor.fetchall()
        if not rows:
            return []
        columns = [desc[0] for desc in cursor.description]
        result = []
        for row in rows:
            row_dict = {}
            for i, col in enumerate(columns):
                val = row[i]
                if isinstance(val, (datetime, date, time)):
                    val = val.isoformat()
                elif isinstance(val, Decimal):
                    val = float(val)
                row_dict[col] = val
            result.append(row_dict)
        return result
    except Exception as e:
        print(f"  Ошибка выгрузки данных из {schema_name}.{table_name}: {e}")
        return []
def get_translations_data(cursor):
    rows = safe_fetch(cursor, "SELECT label_key, language, translation FROM lang.translations ORDER BY language,
label_key")
    return [{"label_key": r[0], "language": r[1], "translation": r[2]} for r in rows]
def main():
    # ============================================================================
    # Определяем, откуда брать параметры: из командной строки или из DEFAULT_RUNTIME_CONFIG
    # ============================================================================
    if len(sys.argv) == 1:
        # Запуск без аргументов: используем DEFAULT_RUNTIME_CONFIG
        print("Запуск без аргументов командной строки. Использую настройки из DEFAULT_RUNTIME_CONFIG.")
        all_mode = DEFAULT_RUNTIME_CONFIG['all']
        schemas_config = DEFAULT_RUNTIME_CONFIG['schemas']
        with_data_tables = DEFAULT_RUNTIME_CONFIG['with_data_tables']
        output_file = DEFAULT_RUNTIME_CONFIG['output_file']
        config_file = None
    else:
        # Парсим аргументы командной строки
        parser = argparse.ArgumentParser(description='Расширенный дамп структуры PostgreSQL и выборочно
данных')
        parser.add_argument('config_file', nargs='?', help='JSON файл с параметрами подключения (опционально)')
        parser.add_argument('-a', '--all', type=int, choices=[0,1], default=0,
                            help='1: вся структура, 0: данные из meta/auth/autodoc/lang/public, 1: все данные')
        for schema in ALL_SCHEMAS:
            parser.add_argument(f'--{schema}', type=int, choices=[0,1], default=None,
                                help=f'Обрабатывать схему {schema}: 1 или 0')
        parser.add_argument('-w', '--with-data-table', action='append', default=[],
                            help='Принудительно выгружать данные из таблицы (schema.table) (можно несколько)')
        parser.add_argument('-o', '--output', default='pg_schema_dump_full.json',
                            help='Имя выходного JSON файла')
        args = parser.parse_args()
        all_mode = args.all
        # Собираем словарь схем из аргументов
        schemas_config = {}
        for schema in ALL_SCHEMAS:
            val = getattr(args, schema, None)
            if val is not None:
                schemas_config[schema] = val
            else:
                schemas_config[schema] = None
        with_data_tables = args.with_data_table
        output_file = args.output
        config_file = args.config_file
    # Загрузка конфигурации подключения
    if config_file:
        with open(config_file, 'r', encoding='utf-8') as f:
            config_data = json.load(f)
        db_config = config_data.get("db", {})
        db_config = {
            "host": db_cfg.get("host", DEFAULT_DB_CONFIG["host"]),
            "port": int(db_cfg.get("port", DEFAULT_DB_CONFIG["port"])),
            "dbname": db_cfg.get("name", DEFAULT_DB_CONFIG["dbname"]),
            "user": db_cfg.get("user", DEFAULT_DB_CONFIG["user"]),
            "password": db_cfg.get("pass", DEFAULT_DB_CONFIG["password"])
        }
    else:
        db_config = DEFAULT_DB_CONFIG
    # Определяем, какие схемы обрабатывать
    schemas_to_process = []
    for schema in ALL_SCHEMAS:
        flag = schemas_config.get(schema)
        if flag is not None:
            if flag == 1:
                schemas_to_process.append(schema)
        else:
            # флаг не задан - обрабатываем схему (по умолчанию все схемы)
            schemas_to_process.append(schema)
    # Определяем, для каких схем выгружать данные таблиц
    schemas_to_load_data = set()
    if all_mode != 1:
        if all_mode == 1:
            schemas_to_load_data = set(schemas_to_process)
        else:  # all_mode == 0
            schemas_to_load_data = set(schemas_to_process) - {'data', 'ext'}
    # Добавляем таблицы, явно указанные в with_data_tables
    extra_tables_info = []
    for tab in with_data_tables:
```

```
        if ' ' in tab:
            sch, tbl = tab.split(' ', 1)
            if sch in schemas_to_process:
                schemas_to_load_data.add(sch)
                extra_tables_info.append((sch, tbl))
        else:
            print(f"Предупреждение: таблица {tab} не указана в формате schema.table, пропускаем")
    try:
        conn = get_connection(db_config)
        conn.autocommit = True
        cursor = conn.cursor()
        result = {}
        for schema in schemas_to_process:
            print(f"Обработка схемы: {schema}")
            schema_data = {
                "comment": get_schema_comment(cursor, schema),
                "tables": get_tables_metadata(cursor, schema),
                "views": get_views_metadata(cursor, schema),
                "functions": get_functions_metadata(cursor, schema),
                "triggers": get_triggers_metadata(cursor, schema),
                "indexes": get_indexes_metadata(cursor, schema),
                "rules": get_rules_metadata(cursor, schema)
            }
            if schema in schemas_to_load_data:
                print(f"  Выгрузка данных для схемы {schema}...")
                data_tables = {}
                # Если в этой схеме есть явно указанные таблицы, выгружаем только их, иначе все
                if extra_tables_info:
                    tables_to_load = [tbl for (sch, tbl) in extra_tables_info if sch == schema]
                else:
                    tables_to_load = [t["name"] for t in schema_data["tables"]]
                for table_name in tables_to_load:
                    print(f"    таблица: {schema}.{table_name}")
                    rows = get_table_data(cursor, schema, table_name, limit=500)
                    if rows:
                        data_tables[table_name] = rows
                if data_tables:
                    schema_data["table_data"] = data_tables
            result[schema] = schema_data
        if 'lang' in schemas_to_process:
            print("  Выгрузка содержимого lang translations в отдельный ключ...")
            result["lang_translations_data"] = get_translations_data(cursor)
        cursor.close()
        conn.close()
        with open(output_file, 'w', encoding='utf-8') as f:
            json.dump(result, f, ensure_ascii=False, indent=2, default=json_serializer)
        print(f"Готово. Результат сохранён в {output_file}")
    except Exception as e:
        print(f"Критическая ошибка: {e}")
        sys.exit(1)
if __name__ == "__main__":
    main()
```

Файл: /home/sergey/Documents/configurate/makePdf.py (относительно: makePdf.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Собирает все .py файлы из указанной директории (рекурсивно) и формирует PDF.
Каждый файл начинается с заголовка (полный путь).
Между файлами – разделитель.
"""
import os
import sys
from pathlib import Path
from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from reportlab.pdfgen import canvas
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.lib.colors import black, blue

# ---------- НАСТРОЙКИ ----------
ROOT_DIR = os.path.dirname(os.path.abspath(__file__))  # по умолчанию – папка скрипта
OUTPUT_PDF = os.path.join(ROOT_DIR, "all_python_files.pdf")

# Расширения, которые включаем – только .py
EXTENSIONS = {".py"}

# Игнорируемые папки
IGNORE_DIRS = {
    '.git', '.svn', '.hg', '.idea', '.vscode',
    'node_modules', 'venv', '.venv', 'env', '.virtualenv',
    'dist', 'build', '.mypy_cache', '.pytest_cache', '.tox',
    '.ipynb_checkpoints', '.DS_Store', 'Thumbs.db', '.trash', '.tmp',
    'logs', 'backup', 'old'
}

# Попробуем загрузить шрифт с поддержкой кириллицы (если есть)
CYRILLIC_FONT = None
for font_path in [
    "/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
    "/usr/share/fonts/truetype/liberation/LiberationSans-Regular.ttf",
]:
    if os.path.exists(font_path):
        try:
            pdfmetrics.registerFont(TTFont('CyrillicFont', font_path))
            CYRILLIC_FONT = 'CyrillicFont'
            break
        except:
            pass
if not CYRILLIC_FONT:
    print("Предупреждение: шрифт с поддержкой кириллицы не найден, используется стандартный (латиница)")
    CYRILLIC_FONT = 'Helvetica'

# ---------- ----------
def should_include_file(file_path):
    ext = os.path.splitext(file_path)[1].lower()
    return ext in EXTENSIONS
def collect_py_files(root_dir):
    files = []
    for dirpath, dirnames, filenames in os.walk(root_dir):
        dirnames[:] = [d for d in dirnames if d not in IGNORE_DIRS]
        for fname in filenames:
            full_path = os.path.join(dirpath, fname)
            if should_include_file(full_path):
                files.append(full_path)
    files.sort()
    return files
def read_file_content(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            return f.read()
    except UnicodeDecodeError:
        try:
            with open(file_path, 'r', encoding='cp1251') as f:
                return f.read()
        except:
            return f"[Не удалось прочитать файл {file_path}]"
    except Exception as e:
        return f"[Ошибка чтения: {e}]"
def add_text_to_canvas(canvas_obj, text, font_name, font_size,
                       x, y, max_width, page_height, margin_top, margin_bottom, leading):
    """
    Разбивает текст на строки, рисует на canvas с учётом ширины страницы.
    Возвращает новую позицию y.
    """
    canvas_obj.setFont(font_name, font_size)
    lines = []
    for line in text.splitlines():
        if canvas_obj.stringWidth(line, font_name, font_size) > max_width:
            words = line.split(' ')
            cur_line = ''
```

```
            for w in words:
                test_line = cur_line + (' ' if cur_line else '') + w
                if canvas_obj.stringWidth(test_line, font_name, font_size) <= max_width:
                    cur_line = test_line
                else:
                    if cur_line:
                        lines.append(cur_line)
                    cur_line = w
            if cur_line:
                lines.append(cur_line)
        else:
            lines.append(line)
    for line in lines:
        if y - leading < margin_bottom:
            canvas_obj.showPage()
            canvas_obj.setFont(font_name, font_size)
            y = page_height - margin_top
        canvas_obj.drawString(x, y, line)
        y -= leading
    return y
def create_pdf(files, output_pdf):
    c = canvas.Canvas(output_pdf, pagesize=A4)
    page_width, page_height = A4
    margin_top = 20 * mm
    margin_bottom = 20 * mm
    margin_left = 15 * mm
    margin_right = 15 * mm
    text_width = page_width - margin_left - margin_right
    leading = 4.5
    font_name = CYRILLIC_FONT
    header_font = 'Helvetica-Bold' if font_name == 'Helvetica' else font_name
    code_font_size = 9
    heading_font_size = 12
    y = page_height - margin_top
    for file_path in files:
        full_path_display = f"Файл: {file_path}"
        rel_path = os.path.relpath(file_path, ROOT_DIR)
        # Добавляем относительный путь после полного
        display = f"{full_path_display} (относительно: {rel_path})"
        if y - heading_font_size - leading < margin_bottom:
            c.showPage()
            y = page_height - margin_top
        c.setFont(header_font, heading_font_size)
        c.setFillColor(blue)
        c.drawString(margin_left, y, display)
        y -= heading_font_size + 2
        c.setFillColor(black)
        content = read_file_content(file_path)
        y = add_text_to_canvas(c, content, font_name, code_font_size,
                               margin_left, y, text_width,
                               page_height, margin_top, margin_bottom, leading)
        if y - 10 < margin_bottom:
            c.showPage()
            y = page_height - margin_top
        else:
            y -= 5
        c.setStrokeColor(black)
        c.line(margin_left, y, page_width - margin_right, y)
        y -= 10
    c.save()
    print(f"PDF создан: {output_pdf}")
def main():
    if not os.path.isdir(ROOT_DIR):
        print(f"Ошибка: директория {ROOT_DIR} не существует")
        sys.exit(1)
    print(f"Сканируем: {ROOT_DIR}")
    py_files = collect_py_files(ROOT_DIR)
    print(f"Найдено Python-файлов: {len(py_files)}")
    if not py_files:
        print("Нет .py файлов для обработки.")
        return
    create_pdf(py_files, OUTPUT_PDF)
if __name__ == "__main__":
    main()
```

Файл: /home/sergey/Documents/configurate/mozart_import/__init__.py (относительно: moz

```
# mozart_import/__init__.py
# -*- coding: utf-8 -*-
from .core.models import (
    SourceType, FileFormat, FieldType,
    SourceField, SourceTable, SourceSchema,
    MatchedField, MatchedEntity, MappingProposal,
    ImportProfile
)
from .core.mozart_matcher import MozartMatcher
from .core.metadata_creator import MetadataCreator
from .core.profile_manager import ProfileManager
from .core.schema_analyzer import SchemaAnalyzer
from .designer.designer_v2 import ImportDesignerV2
from .designer.scene_profile_adapter import SceneProfileAdapter
from .designer.entity_creator import EntityCreator
from .connectors.file_connector import FileConnector
from .connectors.db_connector import DBConnector
from .connectors.url_connector import URLConnector
from .utils.ext_helper import ExtHelper
from .utils.field_creator import FieldCreator
__all__ = [
    'SourceType', 'FileFormat', 'FieldType',
    'SourceField', 'SourceTable', 'SourceSchema',
    'MatchedField', 'MatchedEntity', 'MappingProposal',
    'ImportProfile',
    'MozartMatcher',
    'MetadataCreator',
    'ImportEngine',
    'ProfileManager',
    'SchemaAnalyzer',
    'ImportDesignerV2',
    'SceneProfileAdapter',
    'EntityCreator',
    'FileConnector',
    'DBConnector',
    'URLConnector',
    'ExtHelper',
    'FieldCreator'
]
```

Файл: /home/sergey/Documents/configurate/mozart_import/connectors/__init__.py (относит

```
# mozart_import/connectors/__init__.py
# (пустой файл, обозначает пакет)
```

Файл: /home/sergey/Documents/configurate/mozart_import/connectors/base_connector.py

```
# mozart_import/connectors/base_connector.py
from abc import ABC, abstractmethod
from typing import Any, Dict, List, Optional, Generator
from ..core.models import SourceSchema, SourceTable, SourceField, FieldType, SourceType

class BaseConnector(ABC):
    """Абстрактный базовый класс для всех коннекторов к источникам данных."""
```

```
    def __init__(self, source_path: str, **kwargs):
        self.source_path = source_path
        self.options = kwargs
    @abstractmethod
    def connect(self) -> bool:
        """Устанавливает соединение с источником. Возвращает True при успехе."""
        pass
    @abstractmethod
    def disconnect(self) -> None:
        """Закрывает соединение."""
        pass
    @abstractmethod
    def get_schema(self) -> SourceSchema:
        """Извлекает схему источника (таблицы, поля, типы, связи)."""
        pass
    @abstractmethod
    def get_table_sample(self, table_name: str, limit: int = 100) -> List[Dict[str, Any]]:
        """Возвращает первые limit записей из указанной таблицы."""
        pass
    @abstractmethod
    def iter_records(self, table_name: str, batch_size: int = 1000) -> Generator[List[Dict[str, Any]], None, None]:
        """Генератор, возвращающий записи пакетами для итеративной обработки."""
        pass
    @abstractmethod
    def get_row_count(self, table_name: str) -> Optional[int]:
        """Возвращает приблизительное количество записей в таблице."""
        pass
```

Файл: /home/sergey/Documents/configurate/mozart_import/connectors/db_connector.py (o

```
# mozart_import/connectors/db_connector.py
import urllib.parse
import sqlite3
from typing import List, Dict, Any, Optional, Generator
from .base_connector import BaseConnector
from ..core.models import SourceSchema, SourceTable, SourceField, FieldType, SourceType
try:
    import psycopg2
    import psycopg2.extras
    PSYCOPG2_AVAILABLE = True
except ImportError:
    PSYCOPG2_AVAILABLE = False
try:
    import pymysql
    PYMYSQL_AVAILABLE = True
except ImportError:
    PYMYSQL_AVAILABLE = False
try:
    import pyodbc
    PYODBC_AVAILABLE = True
except ImportError:
    PYODBC_AVAILABLE = False
try:
    import cx_Oracle
    CX_ORACLE_AVAILABLE = True
except ImportError:
    CX_ORACLE_AVAILABLE = False

class DBConnector(BaseConnector):
    """
    Коннектор для реляционных баз данных: PostgreSQL, MySQL, MSSQL, Oracle, SQLite.
    Параметры подключения передаются через kwargs:
    db_type: 'postgresql', 'mysql', 'mssql', 'oracle', 'sqlite'
    host, port, database, user, password (для SQLite только database — путь к файлу)
    """
    def __init__(self, source_path: str, **kwargs):
        super().__init__(source_path, **kwargs)
        self.db_type = kwargs.get('db_type', 'postgresql')
        self.host = kwargs.get('host', 'localhost')
        self.port = kwargs.get('port')
        self.database = kwargs.get('database')
        self.user = kwargs.get('user')
        self.password = kwargs.get('password')
        self._conn = None
        self._cursor = None
    def connect(self) -> bool:
        if self.db_type == 'postgresql':
            if not PSYCOPG2_AVAILABLE:
                raise ImportError("Установите psycopg2: pip install psycopg2-binary")
            self._conn = psycopg2.connect(
                host=self.host,
                port=self.port or 5432,
                database=self.database,
                user=self.user,
                password=self.password
            )
            self._cursor = self._conn.cursor()
        elif self.db_type == 'mysql':
            if not PYMYSQL_AVAILABLE:
                raise ImportError("Установите pymysql: pip install pymysql")
            self._conn = pymysql.connect(
                host=self.host,
                port=self.port or 3306,
                database=self.database,
                user=self.user,
                password=self.password,
                charset='utf8mb4'
            )
            self._cursor = self._conn.cursor()
        elif self.db_type == 'mssql':
            if not PYODBC_AVAILABLE:
                raise ImportError("Установите pyodbc: pip install pyodbc")
            driver = "{ODBC Driver 17 for SQL Server}"
            conn_str = f"DRIVER={driver};SERVER={self.host},{self.port or 1433};DATABASE={self.database};UID={self.user};PWD={self.password}"
            self._conn = pyodbc.connect(conn_str)
            self._cursor = self._conn.cursor()
        elif self.db_type == 'oracle':
            if not CX_ORACLE_AVAILABLE:
                raise ImportError("Установите cx_Oracle: pip install cx-Oracle")
            dsn = cx_Oracle.makedsn(self.host, self.port or 1521, service_name=self.database)
            self._conn = cx_Oracle.connect(user=self.user, password=self.password, dsn=dsn)
            self._cursor = self._conn.cursor()
        elif self.db_type == 'sqlite':
            # Для SQLite database - путь к файлу
            if not self.database:
                raise ValueError("Для SQLite необходимо указать database (путь к файлу)")
            self._conn = sqlite3.connect(self.database)
            self._conn.row_factory = sqlite3.Row
            self._cursor = self._conn.cursor()
        else:
            raise ValueError(f"Неподдерживаемый тип БД: {self.db_type}")
        return True
    def disconnect(self) -> None:
        if self._cursor:
            self._cursor.close()
        if self._conn:
            self._conn.close()
        self._cursor = None
        self._conn = None
    def get_sqlite_schema(self) -> SourceSchema:
        # Получаем список таблиц (исключаем служебные)
        self._cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite_%'")
        tables = [row[0] for row in self._cursor.fetchall()]
        source_tables = []
        for t in tables:
            self._cursor.execute(f"PRAGMA table_info('{t}')")
            cols = self._cursor.fetchall()
            fields = []
            for col in cols:
                col_name = col[1]
                col_type = col[2].upper()
                # Определяем тип поля
```

```
            if 'INT' in col_type:
                ftype = FieldType.INTEGER
                length = None
            elif 'REAL' in col_type or 'FLOAT' in col_type or 'DOUBLE' in col_type:
                ftype = FieldType.FLOAT
                length = None
            elif 'CHAR' in col_type or 'TEXT' in col_type or 'CLOB' in col_type:
                ftype = FieldType.STRING
                length = None
            elif 'BLOB' in col_type:
                ftype = FieldType.BINARY
                length = None
            else:
                ftype = FieldType.STRING
                length = None
            fields.append(SourceField(
                name=col_name,
                field_type=ftype,
                length=length,
                nullable=not col[3]
            ))
        source_tables.append(SourceTable(name=t, fields=fields))
    return SourceSchema(
        source_type=SourceType.DATABASE,
        source_path=f"sqlite://{self.database}",
        tables=source_tables
    )

def get_schema(self) -> SourceSchema:
    if self.db_type == 'postgresql':
        return self._get_postgresql_schema()
    elif self.db_type == 'sqlite':
        return self._get_sqlite_schema()
    # TODO: реализовать для MySQL, MSSQL, Oracle аналогично
    raise NotImplementedError(f"Получение схемы для {self.db_type} пока не реализовано")

def get_table_sample(self, table_name: str, limit: int = 100) -> List[Dict[str, Any]]:
    if self.db_type == 'sqlite':
        self._cursor.execute(f'SELECT * FROM "{table_name}" LIMIT ?', (limit,))
        rows = self._cursor.fetchall()
        columns = [desc[0] for desc in self._cursor.description]
        return [dict(zip(columns, row)) for row in rows]
    else:
        self._cursor.execute(f'SELECT * FROM "{table_name}" LIMIT %s', (limit,))
        columns = [desc[0] for desc in self._cursor.description]
        rows = self._cursor.fetchall()
        return [dict(zip(columns, row)) for row in rows]

def iter_records(self, table_name: str, batch_size: int = 1000) -> Generator[List[Dict[str, Any]], None, None]:
    if self.db_type == 'sqlite':
        offset = 0
        while True:
            self._cursor.execute(f'SELECT * FROM "{table_name}" LIMIT ? OFFSET ?', (batch_size, offset))
            rows = self._cursor.fetchall()
            if not rows:
                break
            columns = [desc[0] for desc in self._cursor.description]
            batch = [dict(zip(columns, row)) for row in rows]
            yield batch
            offset += batch_size
            if len(rows) < batch_size:
                break
    else:
        offset = 0
        while True:
            self._cursor.execute(f'SELECT * FROM "{table_name}" LIMIT %s OFFSET %s', (batch_size, offset))
            rows = self._cursor.fetchall()
            if not rows:
                break
            columns = [desc[0] for desc in self._cursor.description]
            batch = [dict(zip(columns, row)) for row in rows]
            yield batch
            offset += batch_size
            if len(rows) < batch_size:
                break

def get_row_count(self, table_name: str) -> Optional[int]:
    self._cursor.execute(f'SELECT COUNT(*) FROM "{table_name}"')
    return self._cursor.fetchone()[0]
```

Файл: /home/sergey/Documents/configurate/mozart_import/connectors/file_connector.py (o

```
# mozart_import/connectors/file_connector.py
import os
import json
import xml.etree.ElementTree as ET
from typing import List, Dict, Any, Optional, Generator
from .base_connector import BaseConnector
from ..core.models import SourceSchema, SourceTable, SourceField, FieldType, SourceType, FileFormat

try:
    from dbfread import DBF
    DBF_AVAILABLE = True
except ImportError:
    DBF_AVAILABLE = False

try:
    import pandas as pd
    PANDAS_AVAILABLE = True
except ImportError:
    PANDAS_AVAILABLE = False

try:
    from lxml import etree as lxml_etree
    LXML_AVAILABLE = True
except ImportError:
    LXML_AVAILABLE = False

def _clean_key(key: str) -> str:
    """
    Очищает ключ от пространства имён XML и заменяет проблемные символы.
    Например: '{urn:cbr-ru:ed:v2.0}ParticipantInfo' -> 'ParticipantInfo'
    """
    if '}' in key:
        key = key.split('}', 1)[1]
    # Замена дефисов и точек на подчёркивания (опционально)
    key = key.replace('-', '_').replace('.', '_')
    return key

def flatten_xml_element(elem, prefix=''):
    """
    Рекурсивно преобразует XML-элемент lxml в плоский словарь.
    При повторяющихся тегах оставляет ТОЛЬКО ПЕРВОЕ вхождение.
    Ключи очищаются от пространств имен.
    """
    result = {}
    # Атрибуты элемента
    for attr, value in elem.attrib.items():
        clean_attr = _clean_key(attr)
        full_key = f"{prefix}{clean_attr}" if prefix else clean_attr
        result[full_key] = value
    # Текстовое содержимое, если нет дочерних элементов
    if not len(elem) and elem.text and elem.text.strip():
        text_key = prefix + 'text' if prefix else 'text'
        result[text_key] = elem.text.strip()
    # Дочерние элементы
    for child in elem:
        child_tag = _clean_key(child.tag)
        child_prefix = f"{prefix}{child_tag}_" if prefix else f"{child_tag}_"
        child_data = flatten_xml_element(child, child_prefix)
        for k, v in child_data.items():
            if k not in result:
                result[k] = v
    return result

class FileConnector(BaseConnector):
    def __init__(self, source_path: str, file_format: FileFormat, **kwargs):
        super().__init__(source_path, **kwargs)
        self.file_format = file_format
        self._data = None
    def connect(self) -> bool:
        if not os.path.exists(self.source_path):
```

```
        raise FileNotFoundError(f"Файл не найден: {self.source_path}")
    self._load_data()
    return True
def disconnect(self) -> None:
    self._data = None
def _load_data(self):
    if self.file_format == FileFormat.JSON:
        with open(self.source_path, 'r', encoding='utf-8') as f:
            self.data = json.load(f)
            if isinstance(self._data, dict):
                self._data = [self._data]
    elif self.file_format == FileFormat.XML:
        if not LXML_AVAILABLE:
            raise ImportError("Установите lxml: pip install lxml")
        tree = lxml.etree.parse(self.source_path)
        root = tree.getroot()
        records = []
        for elem in root:
            record = flatten_xml_element(elem)
            records.append(record)
        self._data = records
    elif self.file_format == FileFormat.DBF:
        if not DBF_AVAILABLE:
            raise ImportError("Установите dbfread: pip install dbfread")
        table = DBF(self.source_path, ignore_missing_memofile=True)
        self._data = list(table)
    elif self.file_format in (FileFormat.CSV, FileFormat.EXCEL):
        if not PANDAS_AVAILABLE:
            raise ImportError("Установите pandas: pip install pandas")
        if self.file_format == FileFormat.CSV:
            df = pd.read_csv(self.source_path)
        else:
            df = pd.read_excel(self.source_path)
        self._data = df.to_dict(orient='records')
    else:
        raise ValueError(f"Неподдерживаемый формат: {self.file_format}")
def get_schema(self) -> SourceSchema:
    if self._data is None:
        self._load_data()
    if not self._data:
        return SourceSchema(source_type=SourceType.FILE, source_path=self.source_path)
    sample = self._data[0]
    fields = []
    for name, value in sample.items():
        if isinstance(value, str):
            type = FieldType.STRING
            length = len(value)
        elif isinstance(value, int):
            type = FieldType.INTEGER
            length = None
        elif isinstance(value, float):
            type = FieldType.FLOAT
            length = None
        elif isinstance(value, bool):
            type = FieldType.BOOLEAN
            length = None
        else:
            type = FieldType.STRING
            length = None
        fields.append(SourceField(
            name=name,
            field_type=type,
            length=length,
            nullable=False,
            sample_values=[value]
        ))
    table = SourceTable(
        name=os.path.basename(self.source_path),
        fields=fields,
        sample_rows=self._data[:10]
    )
    return SourceSchema(
        source_type=SourceType.FILE,
        source_path=self.source_path,
        tables=[table]
    )
def get_table_sample(self, table_name: str = None, limit: int = 100) -> List[Dict[str, Any]]:
    if self._data is None:
        self._load_data()
    return self._data[:limit]
def iter_records(self, table_name: str = None, batch_size: int = 1000) -> Generator[List[Dict[str, Any]], None,
None]:
    if self._data is None:
        self._load_data()
    for i in range(0, len(self._data), batch_size):
        yield self._data[i:i+batch_size]
def get_row_count(self, table_name: str = None) -> Optional[int]:
    if self._data is None:
        self._load_data()
    return len(self._data)
```

Файл: /home/sergey/Documents/configurate/mozart_import/connectors/url_connector.py (o

```
# mozart_import/connectors/url_connector.py
import requests
import ssl
from typing import List, Dict, Any, Optional, Generator, Callable, Union
from base_connector import BaseConnector
#from ..core.models import SourceSchema, SourceTable, SourceField, FieldType, SourceType
import zipfile
import tempfile
import os
from ..core.models import SourceSchema, SourceTable, SourceField, FieldType, SourceType, FileFormat
class URLConnector(BaseConnector):
    """
    Коннектор для HTTP/HTTPS источников (REST API, RSS, веб-страницы).
    Параметры:
        headers: dict с HTTP-заголовками
        pagination: dict с настройками пагинации
            - type: 'offset', 'page', 'cursor', 'next_url'
            - param_name: имя параметра (page, offset, cursor)
            - page_size: количество записей на страницу
            - next_url_path: JSONPath до URL следующей страницы
        data_path: JSONPath до массива данных (например, 'data.items')
        method: 'GET' или 'POST'
        body: dict для POST-запроса
    """
    def __init__(self, source_path: str, **kwargs):
        super().__init__(source_path, **kwargs)
        self.method = kwargs.get('method', 'GET')
        self.headers = kwargs.get('headers', {})
        self.pagination = kwargs.get('pagination', {})
        self.data_path = kwargs.get('data_path', None)
        self.body = kwargs.get('body', None)
        self.session = None
    def connect(self) -> bool:
        self.session = requests.Session()
        self.session.headers.update(self.headers)
        return True
    def disconnect(self) -> None:
        if self.session:
            self.session.close()
            self.session = None
    def _get_json_path(self, data: Any, path: str) -> Any:
        if not path:
            return data
        parts = path.split('.')
        for part in parts:
            if isinstance(data, dict) and part in data:
                data = data[part]
            elif isinstance(data, list) and part.isdigit() and int(part) < len(data):
                data = data[int(part)]
            else:
                return None
        return data
    def _extract_records(self, response_json: Dict) -> List[Dict]:
```

```
        if self.data_path:
            data = self._get_json_path(response_json, self.data_path)
            if isinstance(data, list):
                return data
            else:
                return []
        # Если data_path не задан, предполагаем, что JSON сам является массивом
        if isinstance(response_json, list):
            return response_json
        # Иначе смотрим первый ключ, который содержит список
        for value in response_json.values():
            if isinstance(value, list):
                return value
        return []
    def _get_next_url(self, response_json: Dict) -> Optional[str]:
        if self.pagination.get('type') == 'next_url':
            next_path = self.pagination.get('next_url_path')
            if next_path:
                return self._get_json_path(response_json, next_path)
        return None
    def fetch_all(self, limit: Optional[int] = None) -> List[Dict[str, Any]]:
        results = []
        url = self.source_path
        page = 1
        offset = 0
        while True:
            paginated_url = self._build_paginated_url(url, page, offset)
            response = self.session.request(self.method, paginated_url, json=self.body)
            response.raise_for_status()
            data = response.json()
            records = self._extract_records(data)
            results.extend(records)
            if limit and len(results) >= limit:
                results = results[:limit]
                break
            next_url = self._get_next_url(data)
            if next_url:
                url = next_url
                continue
            if self.pagination.get('type') == 'page':
                page += 1
                continue
            if self.pagination.get('type') == 'offset':
                offset += self.pagination.get('page_size', 100)
                continue
            break
        return results
    def _build_paginated_url(self, base_url: str, page: int, offset: int) -> str:
        pag_type = self.pagination.get('type')
        if pag_type == 'page':
            param = self.pagination.get('param_name', 'page')
            separator = '&' if '?' in base_url else '?'
            return f"{base_url}{separator}{param}={page}"
        elif pag_type == 'offset':
            param = self.pagination.get('param_name', 'offset')
            page_size = self.pagination.get('page_size', 100)
            separator = '&' if '?' in base_url else '?'
            return f"{base_url}{separator}{param}={offset}&limit={page_size}"
        else:
            return base_url
    def get_schema(self) -> SourceSchema:
        # Для URL источников схему получаем из первой страницы
        sample_data = self.fetch_all(limit=10)
        if not sample_data:
            return SourceSchema(source_type=SourceType.URL, source_path=self.source_path)
        sample = sample_data[0]
        fields = []
        for name, value in sample.items():
            if isinstance(value, str):
                ftype = FieldType.STRING
            elif isinstance(value, int):
                ftype = FieldType.INTEGER
            elif isinstance(value, float):
                ftype = FieldType.FLOAT
            elif isinstance(value, bool):
                ftype = FieldType.BOOLEAN
            elif isinstance(value, dict):
                ftype = FieldType.STRING
            elif isinstance(value, list):
                ftype = FieldType.STRING
            else:
                ftype = FieldType.STRING
            fields.append(SourceField(name=name, field_type=ftype, nullable=True))
        source_table = SourceTable(
            name="data",
            fields=fields,
            sample_rows=sample_data[:5]
        )
        return SourceSchema(
            source_type=SourceType.URL,
            source_path=self.source_path,
            tables=[source_table]
        )
    def get_table_sample(self, table_name: str = "data", limit: int = 100) -> List[Dict[str, Any]]:
        return self.fetch_all(limit=limit)
def iter_records(self, table_name: str = "data", batch_size: int = 1000) -> Generator[List[Dict[str, Any]], None,
None]:
        all_data = self.fetch_all()
        for i in range(0, len(all_data), batch_size):
            yield all_data[i:i+batch_size]
    def get_row_count(self, table_name: str = "data") -> Optional[int]:
        # Для API точное количество получить сложно, возвращаем None
        return None
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/__init__.py (относительно

```
# mozart_import/core/__init__.py
from .models import (
    SourceType, FileFormat, FieldType,
    SourceField, SourceTable, SourceSchema,
    MatchedField, MatchedEntity, MappingProposal,
    ImportProfile
)
from .mozart_matcher import MozartMatcher
from .metadata_creator import MetadataCreator
from .import_engine import ImportEngine
from .profile_manager import ProfileManager
from .schema_analyzer import SchemaAnalyzer

__all__ = [
    'SourceType', 'FileFormat', 'FieldType',
    'SourceField', 'SourceTable', 'SourceSchema',
    'MatchedField', 'MatchedEntity', 'MappingProposal',
    'ImportProfile',
    'MozartMatcher',
    'MetadataCreator',
    'ImportEngine',
    'ProfileManager',
    'SchemaAnalyzer'
]
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/import_engine.py (относи

```
# mozart_import/core/import_engine.py
# -*- coding: utf-8 -*-
from typing import List, Dict, Any, Optional, Callable
from datetime import datetime
from database.import_database_service
from models.import_models ImportProfile, MatchedEntity
from .metadata_creator import MetadataCreator
from ..connectors.base_connector import BaseConnector

class ImportEngine:
    def __init__(self, db: DatabaseService, connector: BaseConnector, profile: ImportProfile):
        self.db = db
        self.connector = connector
        self.profile = profile
        self.meta_creator = MetadataCreator(db)
```

```
        self.stats = {'inserted': 0, 'updated': 0, 'errors': 0, 'skipped': 0}
        self._reference_cache = {}
        self._source_id_field = 'sourceid'
    # ---------- Работа с ext-таблицами (используем cname, а не calias) ----------
    def _ensure_ext_table(self, entity_id: int, entity_cname: str) -> str:
        """Создает ext-таблицу, если ее нет, и возвращает полное имя."""
        ext_table = f"ext.ext_{entity_cname.lower()}"
        check = self.db.execute_query("SELECT to_regclass(%s)", (ext_table,))
        if check and check[0][0]:
            return ext_table
        print(f"Создаю отсутствующую таблицу: {ext_table}")
        sql = f"""
            CREATE TABLE {ext_table} (
                id SERIAL PRIMARY KEY,
                versionid INTEGER NOT NULL REFERENCES data.entity_versions(id) ON DELETE CASCADE
        """)
        self.db.execute_query(sql, fetch=False)
self.db.execute_query(f"ALTER TABLE {ext_table} ADD COLUMN {self._source_id_field} varchar(100)",
fetch=False)
        # Синхронизируем все поля сущности
        fields = self.db.execute_query("SELECT id FROM meta.fields WHERE entitytypeid = %s", (entity_id,))
        for (field_id,) in fields:
            try:
                self.db.execute_procedure("meta.p_field_sync", (field_id,))
            except Exception as e:
                print(f"Ошибка синхронизации поля {field_id}: {e}")
        return ext_table
    def _get_ext_table(self, entity_id: int) -> Optional[str]:
        """Возвращает имя ext-таблицы, используя cname сущности."""
        res = self.db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (entity_id,))
        if not res:
            return None
        cname = res[0][0]
        return self._ensure_ext_table(entity_id, cname)
    def _ensure_sourceid_field(self, entity_id: int):
        """Создает поле sourceid в meta.fields, если его нет."""
        exists = self.db.execute_query(
            "SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s",
            (entity_id, self._source_id_field)
        )
        if not exists:
            res = self.db.execute_query(
                """INSERT INTO meta.fields (entitytypeid, cfieldname, cfieldtype, calias, nlength, lisindexed, status_id)
                VALUES (%s, %s, 'C', 'Source ID', 255, true, 1) RETURNING id""",
                (entity_id, self._source_id_field)
            )
            if res:
                self.db.execute_procedure("meta.p_field_sync", (res[0][0],))
    # ---------- Основные операции с инстансами и версиями ----------
    def _create_instance(self, entity_id: int) -> int:
        res = self.db.execute_query(
            "INSERT INTO data.entity_instances (entitytypeid, tdt_birth) VALUES (%s, now()) RETURNING id",
            (entity_id,)
        )
        return res[0][0]
    def _create_version(self, instance_id: int, entity_id: int, values: Dict[str, Any],
                        fields_map: Dict[str, str], source_id: Optional[str] = None):
        cname = values.get('cname', f'Импорт {datetime.now()}')
        res = self.db.execute_query(
            """INSERT INTO data.entity_versions (instanceid, cname, cversionstatus, tdt_start)
            VALUES (%s, %s, 'Active', now()) RETURNING id""",
            (instance_id, cname)
        )
        version_id = res[0][0]
        ext_table = self._get_ext_table(entity_id)
        if not ext_table:
            raise RuntimeError(f"Не удалось получить/создать ext-таблицу для entity_id={entity_id}")
        columns = ['versionid']
        placeholders = ['%s']
        params = [version_id]
        if source_id is not None:
            columns.append(self._source_id_field)
            placeholders.append('%s')
            params.append(source_id)
        for mozart_field, source_field in fields_map.items():
            if mozart_field == self._source_id_field:
                continue
            columns.append(mozart_field)
            placeholders.append('%s')
            params.append(values.get(source_field))
        sql = f"INSERT INTO {ext_table} ({', '.join(columns)}) VALUES ({', '.join(placeholders)})"
        self.db.execute_query(sql, params, fetch=False)
    def _update_version(self, instance_id: int, entity_id: int, values: Dict[str, Any], fields_map: Dict[str, str]):
        self.db.execute_query(
"UPDATE data.entity_versions SET tdt_end = now(), cversionstatus = 'History' WHERE instanceid = %s AND
cversionstatus = 'Active'",
            (instance_id,)
        )
        self._create_version(instance_id, entity_id, values, fields_map)
    def _find_instance_by_sourceid(self, entity_id: int, source_id: str) -> Optional[int]:
        ext_table = self._get_ext_table(entity_id)
        if not ext_table:
            return None
        res = self.db.execute_query(
            f"SELECT v.instanceid FROM data.entity_versions v "
            f"JOIN {ext_table} e ON v.id = e.versionid "
            f"WHERE e.{self._source_id_field} = %s AND v.cversionstatus = 'Active'",
            (source_id,)
        )
        return res[0][0] if res else None
    def _get_or_create_instance(self, entity_id: int, source_id: Optional[str],
                                values: Dict[str, Any], fields_map: Dict[str, str]) -> int:
        if source_id:
            existing = self._find_instance_by_sourceid(entity_id, source_id)
            if existing:
                self._update_version(existing, entity_id, values, fields_map)
                self.stats['updated'] += 1
                return existing
        instance_id = self._create_instance(entity_id)
        self._create_version(instance_id, entity_id, values, fields_map, source_id)
        self.stats['inserted'] += 1
        return instance_id
    # ---------- Публичный метод для оркестратора ----------
    def import_single_entity(self, entity: MatchedEntity,
                             ref_rules: List[Dict[str, Any]],
                             instance_cache: Dict[str, Dict[str, int]]) -> (Dict[str, int], Dict[str, int]):
        entity_id = entity.target_entity_id
        if not entity_id:
            return {'inserted': 0, 'updated': 0, 'errors': 0, 'skipped': 0}, {}
        self._ensure_sourceid_field(entity_id)
        fields_map = {}
        ref_resolvers = []
        for field in entity.fields:
            fields_map[field.target_field] = field.source_field
            for rule in ref_rules:
                if rule['target_field'] == field.target_field:
                    ref_resolvers.append((field.target_field, rule['ref_entity_id'], rule['ref_key_field']))
                    break
        source_id_field = entity.source_id_field
        mapping_cache = {}
        stats = {'inserted': 0, 'updated': 0, 'errors': 0, 'skipped': 0, 'details': []}
        print(f"\n=== Импорт таблицы {entity.source_table} ===")
        print(f"  entity_id={entity_id}, fields_map={fields_map}, source_id_field={source_id_field}")
        for batch_num, batch in enumerate(self.connector.iter_records(entity.source_table, batch_size=100)):
            for record in batch:
                # Разрешение ссылок
                for rule, ref_entity, ref_key_field in ref_resolvers:
                    key = record.get(ref_key_field)
```

```
                if key:
                    record[fields_map[tgt_field]] = instance_cache.get(ref_entity, {}).get(str(key))
                else:
                    record[fields_map[tgt_field]] = None
            source_id = None
            if source_id_field:
                raw = record.get(source_id_field)
                if raw is not None:
                    source_id = str(raw)
                    if source_id.lower() == 'none':
                        source_id = None
            try:
                instance_id = self._get_or_create_instance(entity_id, source_id, record, fields_map)
                if source_id:
                    mapping_cache[source_id] = instance_id
            except Exception as e:
                stats['errors'] += 1
                err = f"Ошибка {entity.source_table}, source_id={source_id}: {e}"
                print(err)
                stats['details'].append(err)
print(f"Батч {batch_num+1}: вставлено {stats['inserted']}, обновлено {stats['updated']}, ошибок {stats['errors']}")
print(f"=== Итог: вставлено {stats['inserted']}, обновлено {stats['updated']}, ошибок {stats['errors']} ===")
        return stats, mapping_cache
    def run(self, progress_callback: Optional[Callable[[int, int], None]] = None):
        raise NotImplementedError("Используйте ImportOrchestrator.run() или import_single_entity()")
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/import_orchestrator.py (от

```
# mozart_import/core/import_orchestrator.py
# -*- coding: utf-8 -*-
from collections import deque
from typing import Dict, List, Optional, Any
from database.import_database_service
from .models import ImportProfile, MatchedEntity, MatchedField
from .import_engine import ImportEngine

class ImportOrchestrator:
    """
    Управляет порядком импорта таблиц источника на основе их зависимостей.
    Сначала загружаются таблицы без ссылок на другие таблицы (корневые),
    затем зависимые.
    """
    def __init__(self, db: DatabaseService, connector, profile: ImportProfile):
        self.db = db
        self.connector = connector
        self.profile = profile
        self.engine = ImportEngine(db, connector, profile)
        # кэш сопоставлений (source_table_name, source_id) -> instance_id
        self.instance_cache: Dict[str, Dict[str, int]] = {}
        # список ошибок
        self.errors: List[str] = []
    def _build_dependency_graph(self) -> Dict[str, List[str]]:
        """
        Строит граф зависимостей: ключ – имя таблицы, которая зависит от списка целевых таблиц.
        Возвращает словарь {таблица: [таблицы_от_которых_зависит]}
        """
        # Собираем все таблицы, участвующие в импорте
        tables = set()
        for link in self.profile.import_links:
            tables.add(link['source_table'])
        graph = {t: [] for t in tables}
        # Используем source_references, если они есть
        for ref in self.profile.source_references:
            src_table = ref.get('source_table')
            dst_table = ref.get('ref_table')
            if src_table and dst_table and src_table in graph and dst_table in graph:
                graph[src_table].append(dst_table)
            # иначе игнорируем (ссылка на таблицу, которая не в импорте)
        return graph
    def _order_tables(self, graph: Dict[str, List[str]]) -> List[str]:
        """
        Возвращает топологически отсортированный список таблиц, начиная с корневых.
        """
        in_degree = {t: 0 for t in graph}
        for src, deps in graph.items():
            for dep in deps:
                # src зависит от dep -> увеличиваем степень src
                in_degree[src] += 1
        # Корневые – те, у кого нет зависимостей (in_degree == 0)
        queue = deque([t for t, deg in in_degree.items() if deg == 0])
        order = []
        while queue:
            current = queue.popleft()
            order.append(current)
            # Уменьшаем степень у таблиц, которые зависят от current
            for src, deps in graph.items():
                if current in deps:
                    in_degree[src] -= 1
                    if in_degree[src] == 0:
                        queue.append(src)
        if len(order) != len(graph):
            # цикл
            remaining = set(graph.keys()) - set(order)
            raise ValueError(f"Циклическая зависимость между таблицами: {', '.join(remaining)}")
        return order
    def run(self, progress_callback=None) -> Dict[str, int]:
        """
        Выполняет импорт всех таблиц в правильном порядке.
        Возвращает статистику (aggregated stats).
        """
        # 1. Построить порядок
        graph = self._build_dependency_graph()
        try:
            ordered_tables = self._order_tables(graph)
        except ValueError as e:
            self.errors.append(str(e))
            return {'inserted': 0, 'updated': 0, 'errors': len(self.errors), 'skipped': 0}
        # 2. Для каждой таблицы в порядке очереди
        # Подготовим карту маппингов для быстрого доступа
        table_links = {}
        for link in self.profile.import_links:
            table_links[link['source_table']] = link
        total_stats = {'inserted': 0, 'updated': 0, 'errors': 0, 'skipped': 0}
        for table_name in ordered_tables:
            link = table_links.get(table_name)
            if not link:
                # таблица не связана с Mozart – пропускаем
                continue
            # 3. Создаём частичный профиль для одной таблицы
            entity_id = link['target_entity']
            # Безопасно получаем имя сущности
            target_entity_name = link.get('target_entity_name', '')
            if not target_entity_name:
                # попробуем получить из БД
                res = self.db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (entity_id,))
                if res:
                    target_entity_name = res[0][0]
                else:
                    target_entity_name = f"Entity_{entity_id}"
            matched_fields = []
            source_id_field = None
            ref_rules = []  # локальные правила разрешения ссылок для этой таблицы
            # Маппинги из профиля
            for m in link.get('mappings', []):
                mf = MatchedField(
                    source_field=m['source_field'],
```

```
                target_field=mf['target_field'],
                is_reference=m.get('is_reference', False),
            )
            matched_fields.append(mf)
            if m.get('is_source_id'):
                source_id_field = mf['source_field']
            # Если поле ссылочное, находим правило для разрешения
            if m.get('is_reference'):
                # Ищем правило в source_references
                ref_src_table = m.get('ref_source_table', '')
                ref_src_key_field = m.get('ref_source_key_field', '')
                # По ref_src_table находим target_entity_id
                target_entity = None
                for link in self.profile.import_links:
                    if link['source_table'] == ref_src_table:
                        target_entity = link['target_entity_id']
                        break
                if target_entity:
                    ref_rules.append({
                        'target_field': mf['target_field'],
                        'ref_entity_id': target_entity,
                        'ref_key_field': ref_src_key_field
                    })
        entity = MatchedEntity(
            source_table=table_name,
            target_entity_id=entity_id,
            target_entity_name=target_entity_name,
            confidence=1.0,
            fields=matched_fields,
            source_id_field=source_id_field
        )
        # 4. Вызвать импорт одной сущности
        try:
            stats, sourceid_mapping = self.engine.import_single_entity(
                entity, ref_rules, self.instance_cache
            )
            total_stats['inserted'] += stats['inserted']
            total_stats['updated'] += stats['updated']
            total_stats['errors'] += stats['errors']
            total_stats['skipped'] += stats.get('skipped', 0)
            # 5. Сохранить sourceid -> instanceid в кэш для этой таблицы
            cache_key = table_name
            self.instance_cache[cache_key] = sourceid_mapping
        except Exception as e:
            self.errors.append(f"Ошибка импорта {table_name}: {str(e)}")
            total_stats['errors'] += 1
    return total_stats
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/mapping_manager.py (отн

```
# -*- coding: utf-8 -*-
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/metadata_creator.py (отн

```
# mozart_import/core/metadata_creator.py
from typing import Optional, List, Dict, Any
from database.import DatabaseService
from models.import MatchedEntity, MatchedField, ImportProfile
from composition_manager import CompositionManager

class MetadataCreator:
    def __init__(self, db: DatabaseService):
        self.db = db
        self.comp_manager = CompositionManager(db=db)
        self.doc_class_id = self._get_class_id('DOC')
        self.dict_class_id = self._get_class_id('DICT')
    def _get_class_id(self, class_alias: str) -> Optional[int]:
        res = self.db.execute_query("SELECT id FROM meta.entity_classes WHERE calias = %s", (class_alias,))
        return res[0][0] if res else None
    def _create_entity_type(self, table_name: str, is_document: bool = False) -> int:
        cname = table_name.capitalize()
        calias = table_name.lower().replace('_', '_')
        class_id = self.doc_class_id if is_document else self.dict_class_id
        sql = """
INSERT INTO meta.entitytypes (cname, calias, class_id, cbaseclass, lisdependent, lishierarchy, lisversioned,
lisecuritystrategy, status_id)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """RETURNING id
        res = self.db.execute_query(sql, (
            cname, calias, class_id,
            'DOC' if is_document else '',
            False, False, True, 1, 1
        ))
        return res[0][0]
    def _create_field(self, entity_id: int, field_name: str, field_type: str, calias: str, ref_entity_id: Optional[int] = None)
-> int:
        sql = """
INSERT INTO meta.fields (entitytypeid, cfieldname, cfieldtype, calias, lisindexed, ref_entitytypeid, status_id)
        VALUES (%s, %s, %s, %s, %s, %s, %s)
        """RETURNING id
        res = self.db.execute_query(sql, (entity_id, field_name, field_type, calias, True, ref_entity_id, 1))
        field_id = res[0][0]
        self.db.execute_procedure("meta.p_field_sync", (field_id,))
        return field_id
    def ensure_entity_and_fields(self, matched_entity: MatchedEntity) -> int:
        if matched_entity.target_entity_id is not None and not matched_entity.is_new:
            entity_id = matched_entity.target_entity_id
        else:
            is_doc = matched_entity.is_document
            entity_id = self._create_entity_type(matched_entity.source_table, is_doc)
            matched_entity.target_entity_id = entity_id
            matched_entity.is_new = True
        for field in matched_entity.fields:
            if not field.target_field or field.target_field == '':
                source_field_info = None
                for src_field in getattr(matched_entity, 'source_table_fields', []):
                    if src_field.name == field.source_field:
                        source_field_info = src_field
                        break
                if source_field_info is None:
                    continue
                if field.is_reference:
                    mozart_type = 'R'
                    ref_entity = field.ref_entitytype_id
                elif source_field_info.field_type.name == 'INTEGER':
                    mozart_type = 'I'
                    ref_entity = None
                elif source_field_info.field_type.name == 'FLOAT':
                    mozart_type = 'N'
                    ref_entity = None
                else:
                    mozart_type = 'C'
                    ref_entity = None
                new_field_name = field.source_field.lower()
                new_calias = field.source_field.capitalize()
                self._create_field(entity_id, new_field_name, mozart_type, new_calias, ref_entity)
                field.target_field = new_field_name
                field.confidence = 1.0
        return entity_id
def create_composition_if_needed(self, parent_entity_id: int, child_entity_id: int, link_field: str, role_id: int =
None) -> bool:
        if role_id is None:
            roles = self.db.execute_query("SELECT id FROM meta.composition_roles WHERE calias = 'DEFAULT'")
            if roles:
                role_id = roles[0][0]
            else:
                res = self.db.execute_query(
"INSERT INTO meta.composition_roles (calias, cname) VALUES ('DEFAULT', 'Основная') RETURNING id"
                )
                role_id = res[0][0]
        exists = self.db.execute_query(
```

```
"SELECT id FROM meta.entity_composition WHERE parent_entityid=%s AND child_entityid=%s AND
role_id=%s",
        (parent_entity_id, child_entity_id, role_id)
        if exists:
            return True
        return self.comp_manager.add_composition(parent_entity_id, child_entity_id, role_id, link_field, 10)
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/models.py (относительно

```
# mozart_import/core/models.py
# -*- coding: utf-8 -*-
from dataclasses import dataclass, field
from typing import List, Dict, Optional, Any
from enum import Enum

class SourceType(Enum):
    FILE = "file"
    DATABASE = "database"
    URL = "url"

class FileFormat(Enum):
    JSON = "json"
    XML = "xml"
    DBF = "dbf"
    CSV = "csv"
    EXCEL = "excel"

class FieldType(Enum):
    STRING = "string"
    INTEGER = "integer"
    FLOAT = "float"
    BOOLEAN = "boolean"
    DATE = "date"
    DATETIME = "datetime"
    TIME = "time"
    BINARY = "binary"

# Вспомогательные классы для визуализации графа
class NodeType(Enum):
    SOURCE_TABLE = "source_table"
    MOZART_ENTITY = "mozart_entity"

class DataType(Enum):
    STRING = "string"
    INTEGER = "integer"
    FLOAT = "float"
    BOOLEAN = "boolean"
    DATE = "date"
    DATETIME = "datetime"

class LinkType(Enum):
    IMPORT = "import"
    FK_SOURCE = "fk_source"
    COMPOSITION = "composition"

@dataclass
class Field:
    name: str
    data_type: DataType = DataType.STRING
    is_primary_key: bool = False
    is_foreign_key: bool = False
    references: Optional[str] = None

@dataclass
class Node:
    id: str
    name: str
    node_type: NodeType
    fields: List[Field] = field(default_factory=list)
    x: float = 0
    y: float = 0
    table_name: Optional[str] = None  # для таблиц источника
    entity_id: Optional[int] = None  # для сущностей Mozart
    class_alias: Optional[str] = None

@dataclass
class SourceField:
    name: str
    field_type: FieldType
    length: Optional[int] = None
    nullable: bool = True
    is_primary_key: bool = False
    is_foreign_key: bool = False
    refers_to: Optional[str] = None
    sample_values: List[Any] = field(default_factory=list)

@dataclass
class SourceTable:
    name: str
    fields: List[SourceField] = field(default_factory=list)
    primary_key: Optional[List[str]] = None
    foreign_keys: Dict[str, str] = field(default_factory=dict)
    row_count: Optional[int] = None
    sample_rows: List[Dict[str, Any]] = field(default_factory=list)

@dataclass
class SourceSchema:
    source_type: SourceType
    source_path: str
    tables: List[SourceTable] = field(default_factory=list)

@dataclass
class MatchedField:
    source_field: str
    target_field: str
    confidence: float = 1.0
    transformation: Optional[str] = None
    is_reference: bool = False
    ref_entity_id: Optional[int] = None
    field_type: str = "C"
    field_length: int = 255
    is_source_id: bool = False

@dataclass
class MatchedEntity:
    source_table: str
    target_entity_id: Optional[int]
    target_entity_name: str
    confidence: float
    fields: List[MatchedField] = field(default_factory=list)
    is_new: bool = False
    is_document: bool = False
    source_id_field: Optional[str] = None

@dataclass
class MappingProposal:
    source_schema: SourceSchema
    entities: List[MatchedEntity] = field(default_factory=list)
    unresolved_tables: List[str] = field(default_factory=list)
    suggestions_for_new_entities: List[str] = field(default_factory=list)

@dataclass
class ImportProfile:
    id: Optional[int] = None
    name: str
    source_type: SourceType = SourceType.FILE
    source_path: str
    format: Optional[FileFormat] = None
    connection_params: Dict[str, str] = field(default_factory=dict)
    mapping_proposal: Optional[MappingProposal] = None
    scheduled: bool = False
```

```
    schedule_cron: Optional[str] = None
    last_run: Optional[str] = None
    last_status: Optional[str] = None
    scene_json: Optional[str] = None
    # Нормализованные поля
    source_tables: List[Dict] = field(default_factory=list)
    target_entities: List[Dict] = field(default_factory=list)
    import_links: List[Dict] = field(default_factory=list)
    source_links: List[Dict] = field(default_factory=list)
    # Новое: ручные привязки и справочники источника
    source_references: List[Dict] = field(default_factory=list)  # [{'source_table': str, 'source_field': str, 'ref_table':
str, 'ref_field': str}]
    # Параметры подключения
    file_format: Optional[str] = None
    db_type: Optional[str] = None
    db_host: Optional[str] = None
    db_port: Optional[int] = None
    db_name: Optional[str] = None
    db_user: Optional[str] = None
    db_pass: Optional[str] = None
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/mozart_matcher.py (относ

```
# mozart_import/core/mozart_matcher.py
from typing import List, Dict, Optional, Any, Tuple
from difflib import SequenceMatcher
from models import SourceSchema, SourceTable, SourceField, FieldType
from models import MatchedEntity, MatchedField, MappingProposal
from database import DatabaseService

class MozartMatcher:
    """
    Сопоставляет исходную схему (SourceSchema) с метаданными Mozart.
    Возвращает MappingProposal с предложениями по сущностям и полям.
    """
    def __init__(self, db: DatabaseService):
        self.db = db
        # Загружаем метаданные Mozart при инициализации
        self.entitytypes = {}  # {id: {cname, calias, class_id, class_alias, fields}}
        self._load_metadata()
    def _load_metadata(self):
        # Загружаем сущности
        rows = self.db.execute_query("""
            SELECT e.id, e.cname, e.calias, e.class_id, c.calias as class_alias
            FROM meta.entitytypes e
            LEFT JOIN meta.entity_classes c ON e.class_id = c.id
            ORDER BY e.cname
        """)
        for row in rows:
            eid, cname, calias, class_id, class_alias = row
            self.entitytypes[eid] = {
                'cname': cname,
                'calias': calias,
                'class_id': class_id,
                'class_alias': class_alias,
                'fields': {}
            }
        # Загружаем поля
        rows = self.db.execute_query("""
            SELECT f.entitytypeid, f.cfieldname, f.calias, f.cfieldtype,
                   f.ref_entitytypeid, ft.cname as type_name
            FROM meta.fields f
            LEFT JOIN meta.field_types ft ON f.cfieldtype = ft.ccode
            ORDER BY f.entitytypeid, f.isortorder
        """)
        for row in rows:
            eid, cfieldname, calias, cfieldtype, ref_eid, type_name = row
            if eid in self.entitytypes:
                self.entitytypes[eid]['fields'][cfieldname] = {
                    'calias': calias,
                    'cfieldtype': cfieldtype,
                    'ref_entitytypeid': ref_eid,
                    'type_name': type_name
                }
    def _similarity(self, a: str, b: str) -> float:
        return SequenceMatcher(None, a.lower(), b.lower()).ratio()
    def _match_table_to_entity(self, table: SourceTable) -> Tuple[Optional[int], str, float]:
        """
        Пытается найти подходящую сущность в Mozart по имени таблицы.
        Возвращает (entity_id, entity_name, confidence)
        """
        best_id = None
        best_name = None
        best_score = 0.0
        for eid, info in self.entitytypes.items():
            score = max(self._similarity(table.name, info['cname']),
                        self._similarity(table.name, info['calias']))
            if score > best_score:
                best_score = score
                best_id = eid
                best_name = info['cname']
        if best_score > 0.6:
            return best_id, best_name, best_score
        return None, table.name, 0.0
    def _match_field_to_field(self, source_field: SourceField, target_fields: Dict) -> Tuple[Optional[str], float]:
        """
        Сопоставляет поле источника с полем сущности Mozart по имени и типу.
        Возвращает (target_field_name, confidence)
        """
        best_field = None
        best_score = 0.0
        for fname, finfo in target_fields.items():
            name_score = max(self._similarity(source_field.name, fname),
                             self._similarity(source_field.name, finfo['calias']))
            type_score = 1.0
            # Упрощенное соответствие типов
            if finfo['cfieldtype'] == 'C' and source_field.field_type == FieldType.STRING:
                type_score = 1.0
            elif finfo['cfieldtype'] == 'I' and source_field.field_type == FieldType.INTEGER:
                type_score = 1.0
            elif finfo['cfieldtype'] == 'N' and source_field.field_type == FieldType.FLOAT:
                type_score = 1.0
            elif finfo['cfieldtype'] == 'D' and source_field.field_type == FieldType.DATE:
                type_score = 1.0
            elif finfo['cfieldtype'] == 'L' and source_field.field_type == FieldType.BOOLEAN:
                type_score = 1.0
            else:
                type_score = 0.5
            total = (name_score * 0.7 + type_score * 0.3)
            if total > best_score:
                best_score = total
                best_field = fname
        if best_score > 0.4:
            return best_field, best_score
        return None, 0.0
    def propose(self, source_schema: SourceSchema) -> MappingProposal:
        """
        Строит MappingProposal для всей исходной схемы.
        """
        entities = []
        unresolved = []
        suggestions = []
        for table in source_schema.tables:
            entity_id, entity_name, conf = self._match_table_to_entity(table)
            if entity_id is not None:
                # Существующая сущность
                target_fields = self.entitytypes[entity_id]['fields']
                matched_fields = []
                for src_field in table.fields:
                    target_field, conf = self._match_field_to_field(src_field, target_fields)
                    if target_field:
                        is_ref = target_fields[target_field]['cfieldtype'] == 'R'
                        ref_entity = target_fields[target_field]['ref_entitytypeid']
                        matched_fields.append(MatchedField(
                            source_field=src_field.name,
                            target_field=target_field,
                            confidence=conf,
                            is_reference=is_ref,
```

```
                ref_entitytype_id=ref_entity
            ))
        else:
            # Поле не найдено, предлагаем добавить
            matched_fields.append(MatchedField(
                source_field=src_field.name,
                target_field=None,  # будет новое
                confidence=0.0,
                is_reference=False
            ))
    entities.append(MatchedEntity(
        source_table=table.name,
        target_entity_id=entity.id,
        target_entity_name=entity.name,
        confidence=conf,
        fields=matched_fields,
        is_new=False
    ))
else:
    # Новая сущность
    unresolved.append(table.name)
    suggestions.append(table.name)
return MappingProposal(
    source_schema=source_schema,
    entities=entities,
    unresolved_tables=unresolved,
    suggestions_for_new_entities=suggestions
)
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/profile_manager.py (относ

```
# mozart_import/core/profile_manager.py
# -*- coding: utf-8 -*-
import json
from typing import Optional, List
from database.import_database_service
from .models import ImportProfile, SourceType

class ProfileManager:
    def __init__(self, db: DatabaseService):
        self.db = db

    def save_profile(self, profile: ImportProfile) -> int:
        source_type_code = profile.source_type.value if profile.source_type else None
        # 1. Сохраняем или обновляем основной профиль
        if profile.id:
            self.db.execute_query("""
                UPDATE meta.import_profiles
                SET profile_name=%s, source_type=%s, source_path=%s,
                    file_format=%s, db_type=%s, db_host=%s, db_port=%s,
                    db_name=%s, db_user=%s, db_pass=%s,
                    updated_at=now()
                WHERE id=%s
            """, (profile.name, source_type_code, profile.source_path,
                  profile.file_format, profile.db_type, profile.db_host,
                  profile.db_port, profile.db_name, profile.db_user,
                  profile.db_pass, profile.id), fetch=False)
            # Удаляем старые связанные данные
self.db.execute_query("DELETE FROM meta.import_source_links WHERE profile_id=%s", (profile.id,),
fetch=False)
self.db.execute_query("DELETE FROM meta.import_mappings WHERE link_id IN (SELECT id FROM
meta.import_links WHERE profile_id=%s)", (profile.id,), fetch=False)
self.db.execute_query("DELETE FROM meta.import_target_links WHERE profile_id=%s", (profile.id,), fetch=False)
self.db.execute_query("DELETE FROM meta.import_target_entities WHERE profile_id=%s", (profile.id,),
fetch=False)
self.db.execute_query("DELETE FROM meta.import_source_references WHERE source_table_id IN (SELECT id
FROM meta.import_source_tables WHERE profile_id=%s)", (profile.id,), fetch=False)
self.db.execute_query("DELETE FROM meta.import_source_tables WHERE profile_id=%s", (profile.id,),
fetch=False)
        else:
            res = self.db.execute_query("""
                INSERT INTO meta.import_profiles
                    (profile_name, source_type, source_path, file_format,
                     db_type, db_host, db_port, db_name, db_user, db_pass)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                RETURNING id
            """, (profile.name, source_type_code, profile.source_path,
                  profile.file_format, profile.db_type, profile.db_host,
                  profile.db_port, profile.db_name, profile.db_user,
                  profile.db_pass))
            profile.id = res[0][0]
        # 2. Сохраняем таблицы источника
        source_table_ids = {}
        for table in profile.source_tables:
            res = self.db.execute_query("""
                INSERT INTO meta.import_source_tables (profile_id, table_name, row_count, sample_data)
                VALUES (%s, %s, %s, %s)
                RETURNING id
            """, (profile.id, table['name'], table.get('row_count'), json.dumps(table.get('sample_rows', []))))
            source_table_ids[table['name']] = res[0][0]
        # 3. Сохраняем целевые сущности
        target_entity_ids = {}
        for entity in profile.target_entities:
            res = self.db.execute_query("""
                INSERT INTO meta.import_target_entities (profile_id, entity_id, entity_name)
                VALUES (%s, %s, %s)
                RETURNING id
            """, (profile.id, entity['id'], entity['name']))
            target_entity_ids[entity['id']] = res[0][0]
        # 4. Сохраняем связи и маппинги (с обработкой дубликатов)
        for link in profile.import_links:
            src_id = source_table_ids.get(link['source_table'])
            tgt_id = target_entity_ids.get(link['target_entity_id'])
            if src_id and tgt_id:
                res = self.db.execute_query("""
                    INSERT INTO meta.import_links (profile_id, source_table_id, target_entity_id)
                    VALUES (%s, %s, %s)
                    RETURNING id
                """, (profile.id, src_id, tgt_id))
                link_id = res[0][0]
                # Удаляем старые маппинги для этого link_id (на случай обновления)
self.db.execute_query("DELETE FROM meta.import_mappings WHERE link_id=%s", (link_id,), fetch=False)
                for mapping in link['mappings']:
                    self.db.execute_query("""
                        INSERT INTO meta.import_mappings
                            (link_id, source_field, target_field, field_type, field_length, is_source_id,
                             is_reference, ref_src_table, ref_src_key_field)
                        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
                        ON CONFLICT (link_id, source_field) DO UPDATE SET
                            target_field = EXCLUDED.target_field,
                            field_type = EXCLUDED.field_type,
                            field_length = EXCLUDED.field_length,
                            is_source_id = EXCLUDED.is_source_id,
                            is_reference = EXCLUDED.is_reference,
                            ref_src_table = EXCLUDED.ref_src_table,
                            ref_src_key_field = EXCLUDED.ref_src_key_field
                    """, (link_id, mapping['source_field'], mapping['target_field'],
                          mapping.get('field_type'), mapping.get('field_length', 255),
                          mapping.get('is_source_id', False),
                          mapping.get('is_reference', False),
                          mapping.get('ref_source_table'),
                          mapping.get('ref_source_key_field', '')), fetch=False)
        # 5. Сохраняем внутренние связи источника (FK)
        for fk in profile.source_fk_links:
            src_id = source_table_ids.get(fk['source_table'])
            ref_id = source_table_ids.get(fk['target_table'])
            if src_id and ref_id:
                self.db.execute_query("""
INSERT INTO meta.import_source_links (profile_id, source_table_id, source_field, ref_table_id, ref_field)
                    VALUES (%s, %s, %s, %s, %s)
                    ", (profile.id, src_id, fk['source_field'], ref_id, fk['target_field']), fetch=False)
        # 6. Сохраняем перекрестные ссылки источника (source_references)
        for ref in profile.source_references:
            src_id = source_table_ids.get(ref['source_table'])
            dst_id = source_table_ids.get(ref['ref_table'])
            if src_id and dst_id:
                self.db.execute_query("""
INSERT INTO meta.import_source_references (source_table_id, source_field, ref_table_id, ref_key_field)
                    VALUES (%s, %s, %s, %s)
```

```
            ON CONFLICT (source_table_id, source_field)
        """DO UPDATE SET ref_table_id = EXCLUDED.ref_table_id, ref_key_field = EXCLUDED.ref_key_field
        , (src_id, ref['source_field'], dst_id, ref['ref_field']), fetch=False)
    return profile.id
def load_profile(self, profile_id: int) -> Optional[ImportProfile]:
    rows = self.db.execute_query(
        SELECT id, profile_name, source_type, source_path,
        file_format, db_type, db_host, db_port, db_name, db_user, db_pass
        FROM meta.import_profiles
        WHERE id = %s
    """, (profile_id,))
    if not rows:
        return None
    row = rows[0]
    profile = ImportProfile(
        id=row[0],
        name=row[1],
        source_type=SourceType(row[2]),
        source_path=row[3],
        file_format=row[4],
        db_type=row[5],
        db_host=row[6],
        db_port=row[7],
        db_name=row[8],
        db_user=row[9],
        db_pass=row[10]
    )
    source_tables = self.db.execute_query("""
        SELECT id, table_name, row_count, sample_data
        FROM meta.import_source_tables
        WHERE profile_id = %s
    """, (profile_id,))
    profile.source_tables = []
    source_table_map = {}
    for st in source_tables:
        table = {
            'id': st[0],
            'name': st[1],
            'row_count': st[2],
            'sample_rows': st[3] if st[3] else []
        }
        profile.source_tables.append(table)
        source_table_map[st[1]] = st[0]
    target_entities = self.db.execute_query("""
        SELECT id, entity_id, entity_name
        FROM meta.import_target_entities
        WHERE profile_id = %s
    """, (profile_id,))
    profile.target_entities = []
    target_entity_map = {}
    for te in target_entities:
        entity = {'id': te[1], 'name': te[2]}
        profile.target_entities.append(entity)
        target_entity_map[te[1]] = te[0]
    links = self.db.execute_query("""
        SELECT l.id, st.table_name, te.entity_id, te.entity_name
        FROM meta.import_links l
        JOIN meta.import_source_tables st ON l.source_table_id = st.id
        JOIN meta.import_target_entities te ON l.target_entity_id = te.id
    """, (profile_id,))
    profile.import_links = []
    for link in links:
        mappings = self.db.execute_query("""
            SELECT source_field, target_field, field_type, field_length, is_source_id,
            is_reference, ref_src_table, ref_src_key_field
            FROM meta.import_mappings
            WHERE link_id = %s
        """, (link[0],))
        profile.import_links.append({
            'source_table': link[1],
            'target_entity_id': link[2],
            'target_entity_name': link[3],
            'mappings': [{'source_field': m[0], 'target_field': m[1], 'field_type': m[2],
                          'field_length': m[3], 'is_source_id': m[4],
                          'is_reference': m[5], 'ref_source_table': m[6],
                          'ref_source_key_field': m[7]} for m in mappings]
        })
    source_fks = self.db.execute_query("""
        SELECT st1.table_name, ls.source_field, st2.table_name, ls.ref_field
        FROM meta.import_source_links ls
        JOIN meta.import_source_tables st1 ON ls.source_table_id = st1.id
        JOIN meta.import_source_tables st2 ON ls.ref_table_id = st2.id
        WHERE ls.profile_id = %s
    """, (profile_id,))
    profile.source_fk_links = [{'source_table': f[0], 'source_field': f[1],
                                'target_table': f[2], 'target_field': f[3]} for f in source_fks]
    source_refs = self.db.execute_query("""
        SELECT st1.table_name, srf.source_field, st2.table_name, srf.ref_key_field
        FROM meta.import_source_references srf
        JOIN meta.import_source_tables st1 ON srf.source_table_id = st1.id
        JOIN meta.import_source_tables st2 ON srf.ref_table_id = st2.id
        WHERE st1.profile_id = %s
    """, (profile_id,))
    profile.source_references = [{'source_table': r[0], 'source_field': r[1],
                                  'ref_table': r[2], 'ref_field': r[3]} for r in source_refs]
    return profile
def list_profiles(self) -> List[ImportProfile]:
    rows = self.db.execute_query(
        SELECT id, profile_name, source_type, source_path
        FROM meta.import_profiles
        ORDER BY created_at DESC
    """)
    profiles = []
    for row in rows:
        profiles.append(ImportProfile(
            id=row[0],
            name=row[1],
            source_type=SourceType(row[2]),
            source_path=row[3]
        ))
    return profiles
def delete_profile(self, profile_id: int):
    self.db.execute_query("DELETE FROM meta.import_profiles WHERE id=%s", (profile_id,), fetch=False)
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/schema_analyzer.py (отно

```
# mozart_import/core/schema_analyzer.py
from typing import Dict, List, Optional, Any
from collections import Counter
from .models import SourceSchema, SourceTable, SourceField, FieldType

class SchemaAnalyzer:
    """
    Анализирует исходную схему, добавляет метаинформацию:
    - предполагаемые первичные ключи (по именам или уникальности)
    - предполагаемые внешние ключи (по именам полей)
    - статистику по полям (уникальность, заполненность)
    - кандидатов на иерархические связи (parent_id и т.п.)
    """
    def __init__(self, schema: SourceSchema):
        self.schema = schema
        self._analyze()
    def _analyze(self):
        for table in self.schema.tables:
            self._analyze_table(table)
    def _analyze_table(self, table: SourceTable):
        # 1. Определяем вероятный первичный ключ
        pk_candidates = []
        for field in table.fields:
            if field.is_primary_key:
                pk_candidates.append(field.name)
            elif field.name.lower() in ('id', 'code', 'num', 'number', 'key'):
                pk_candidates.append(field.name)
```

```
        if pk_candidates:
            table.primary_key = pk_candidates
        else:
            # если нет очевидного, предлагаем первый подходящий по типу целочисленный
            int_fields = [f.name for f in table.fields if f.field_type in (FieldType.INTEGER)]
            if int_fields:
                table.primary_key = [int_fields[0]]
        # 2. Определяем вероятные внешние ключи (по именам)
        fk_candidates = {}
        for field in table.fields:
            name_lower = field.name.lower()
            if name_lower.endswith('_id') or name_lower.startswith('id_') or name_lower in ('parentid', 'parent_id'):
                # предполагаем, что ссылается на таблицу без последнего суффикса
                target_table = name_lower.replace('_id', '').replace('id_', '')
                if target_table == 'parent':
                    target_table = table.name  # ссылка на саму себя (иерархия)
                fk_candidates[field.name] = target_table
        table.foreign_keys = fk_candidates
        # 3. Собираем статистику по полям (если есть sample_rows)
        if table.sample_rows:
            for field in table.fields:
                values = [row.get(field.name) for row in table.sample_rows if row.get(field.name) is not None]
                if len(values) < len(table.sample_rows):
                    field.nullable = True
                # уникальность можно оценить на основе sample
                unique_vals = len(set(values))
                if unique_vals > 0 and len(values) > 0:
                    uniqueness = unique_vals / len(values)
                    # если в sample все значения уникальны, предполагаем, что поле уникально
                    if uniqueness == 1.0 and len(values) == len(table.sample_rows):
                        # помечаем как возможный кандидат на ключ
                        if not table.primary_key:
                            table.primary_key = [field.name]
    def get_primary_key_candidates(self, table_name: str) -> List[str]:
        for table in self.schema.tables:
            if table.name == table_name and table.primary_key:
                return table.primary_key
        return []
    def get_foreign_key_candidates(self, table_name: str) -> Dict[str, str]:
        for table in self.schema.tables:
            if table.name == table_name:
                return table.foreign_keys
        return {}
    def suggest_hierarchy_field(self, table_name: str) -> Optional[str]:
        """Возвращает имя поля, вероятно, указывающего на родителя (для иерархий)."""
        for table in self.schema.tables:
            if table.name == table_name:
                for field in table.fields:
                    name_lower = field.name.lower()
                    if name_lower in ('parentid', 'parent_id', 'pid', 'parent'):
                        return field.name
        return None
    def suggest_reference(self, field_name: str) -> Optional[str]:
        """По имени поля предполагает целевую сущность для внешней ссылки."""
        name_lower = field_name.lower()
        if name_lower.endswith('_id'):
            target = name_lower[:-3]
        elif name_lower.startswith('id_'):
            target = name_lower[3:]
        else:
            return None
        # ищем в схеме таблицу с таким именем
        for table in self.schema.tables:
            if table.name.lower() == target:
                return table.name
        return target  # не найдено, но возвращаем предположение
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/source_analyzer.py (относ

```
# -*- coding: utf-8 -*-
```

Файл: /home/sergey/Documents/configurate/mozart_import/core/source_connector.py (отно

Файл: /home/sergey/Documents/configurate/mozart_import/designer/__init__.py (относител

```
# mozart_import/designer/__init__.py
# -*- coding: utf-8 -*-
from .designer_v2 import ImportDesignerV2
from .designer_trees import SourceTreeManager, MozartTreeManager
from .designer_links import ImportLinksManager
from .designer_import_ops import ImportOperations
from .scene_profile_adapter import SceneProfileAdapter
from .entity_creator import EntityCreator
from .designer_mass_field_dialog import MassFieldDialog
from .designer_utils import to_field_name

__all__ = [
    'ImportDesignerV2',
    'SourceTreeManager',
    'MozartTreeManager',
    'ImportLinksManager',
    'ImportOperations',
    'SceneProfileAdapter',
    'EntityCreator',
    'MassFieldDialog',
    'to_field_name'
]
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_import.py (о

```
# mozart_import/designer/designer_import.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QProgressDialog, QMessageBox
from PyQt5.QtCore import Qt
from ..core.models import ImportProfile, SourceType, MappingProposal, MatchedEntity, MatchedField
from ..utils.field_creator import FieldCreator
from ..core.import_engine import ImportEngine
from .designer_mass_field_dialog import MassFieldDialog

class ImportOperations:
    """Методы импорта и создания полей."""
    def __init__(self, parent, db, connector, translator, source_tables, mozart_entities, import_links):
        self.parent = parent
        self.db = db
        self.connector = connector
        self.translator = translator
        self.source_tables = source_tables
        self.mozart_entities = mozart_entities
        self.import_links = import_links
    def set_source_tables(self, tables):
        self.source_tables = tables
    def set_mozart_entities(self, entities):
        self.mozart_entities = entities
    def set_import_links(self, links):
        self.import_links = links
    def create_fields_from_checked(self, target_entity_item, source_tree_manager):
        entity_id = target_entity_item.data(0, Qt.UserRole)
        checked_fields = source_tree_manager.get_checked_fields()
        if not checked_fields:
            QMessageBox.warning(self.parent, self.translator.tr('warning'), self.translator.tr('no_fields_selected'))
            return
        dlg = MassFieldDialog(self.parent, checked_fields, self.translator)
        if dlg.exec_():
            fields_to_create = dlg.get_fields()
            if not fields_to_create:
```

```
            return
        creator = FieldCreator(self.db)
        for f in fields_to_create:
            creator.create_fields(entity_id, [{
                'cfieldname': f['cfieldname'][:63],
                'cfieldtype': f['cfieldtype'],
                'calias': f['calias'][:50],
                'nlength': f['nlength']
            }])
    def perform_import(self):
        creator = FieldCreator(self.db)
        for link in self.import_links:
            if link['type'] == 'import' and link['target_field']:
                check = self.db.execute_query(
                    "SELECT id FROM meta_fields WHERE entitytypeid=%s AND cfieldname=%s",
                    (link['target_entity_id'], link['target_field'])
                )
                if not check:
                    creator.create_fields(link['target_entity_id'], [{
                        'cfieldname': link['target_field'][:63],
                        'cfieldtype': 'C',
                        'calias': link['target_field'].capitalize()[:50],
                        'nlength': 255
                    }])
            elif link['type'] == 'sourceid':
                check = self.db.execute_query(
                    "SELECT id FROM meta_fields WHERE entitytypeid=%s AND cfieldname='sourceid'",
                    (link['target_entity_id'],)
                )
                if not check:
                    creator.create_fields(link['target_entity_id'], [{
                        'cfieldname': 'sourceid',
                        'cfieldtype': 'C',
                        'calias': 'Source ID',
                        'nlength': 100
                    }])
        # Группировка связей по целевой сущности
        groups = {}
        for link in self.import_links:
            eid = link['target_entity_id']
            if eid not in groups:
                groups[eid] = []
            groups[eid].append(link)
        entities = []
        for eid, links in groups.items():
            target_entity = None
            for ent in self.mozart_entities:
                if ent['id'] == eid:
                    target_entity = ent
                    break
            if not target_entity:
                continue
            source_table_name = links[0]['source_table']
            matched_fields = []
            for link in links:
                if link['type'] == 'import' and link['target_field']:
                    matched_fields.append(MatchedField(
                        source_field=link['source_field'],
                        target_field=link['target_field']
                    ))
            entities.append(MatchedEntity(
                source_table=source_table_name,
                target_entity_id=eid,
                target_entity_name=target_entity['cname'],
                confidence=1.0,
                fields=matched_fields,
                is_new=False
            ))
        mapping_proposal = MappingProposal(
            source_schema=self.connector.get_schema(),
            entities=entities,
            unresolved_tables=[],
            suggestions_for_new_entities=[]
        )
        profile = ImportProfile(
            name=self.connector.source_path,
            source_type=self.connector.type,
            source_path=self.connector.source_path,
            mapping_proposal=mapping_proposal
        )
progress = QProgressDialog(self.translator.tr('import_progress'), self.translator.tr('cancel'), 0, 0, self.parent)
        progress.setWindowModality(Qt.WindowModal)
        try:
            engine = ImportEngine(self.db, self.connector, profile)
            stats = engine.run()
            progress.close()
            QMessageBox.information(self.parent, self.translator.tr('import_finished_title'),
self.translator.tr('import_finished_stats').format(stats['inserted'], stats['updated'], stats['errors']))
        except Exception as e:
            progress.close()
            QMessageBox.critical(self.parent, self.translator.tr('error'), str(e))
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_links.py (отн

```
# mozart_import/designer/designer_links.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QDialog, QFormLayout, QComboBox, QVBoxLayout, QDialogButtonBox,
QPushButton, QTableWidgetItem, QMessageBox
from PyQt5.QtCore import Qt

class ImportLinksManager:
    """Управление связями импорта (таблица links_table)"""
    def __init__(self, parent, links_table, translator, source_tables, mozart_entities):
        self.parent = parent
        self.links_table = links_table
        self.translator = translator
        self.source_tables = source_tables
        self.mozart_entities = mozart_entities
        self.import_links = []
    def set_source_tables(self, tables):
        self.source_tables = tables
    def set_mozart_entities(self, entities):
        self.mozart_entities = entities
    def add_import_link(self):
        dlg = QDialog(self.parent)
        dlg.setWindowTitle(self.translator.tr('add_import_link_title'))
        layout = QVBoxLayout(dlg)
        form = QFormLayout()
        src_table_combo = QComboBox()
        for t in self.source_tables:
            src_table_combo.addItem(t.name, t.name)
        form.addRow(self.translator.tr('source_table'), src_table_combo)
        src_field_combo = QComboBox()
src_table_combo.currentIndexChanged.connect(lambda: self.update_src_fields(src_table_combo,
src_field_combo))
        self.update_src_fields(src_table_combo, src_field_combo)
        form.addRow(self.translator.tr('source_field'), src_field_combo)
        target_entity_combo = QComboBox()
        for ent in self.mozart_entities:
            target_entity_combo.addItem(f"{ent['cname']} ({ent['calias']})", ent['id'])
        form.addRow(self.translator.tr('target_entity'), target_entity_combo)
        target_field_combo = QComboBox()
target_entity_combo.currentIndexChanged.connect(lambda: self.update_target_fields(target_entity_combo,
target_field_combo))
        self.update_target_fields(target_entity_combo, target_field_combo)
        form.addRow(self.translator.tr('target_field'), target_field_combo)
        type_combo = QComboBox()
        type_combo.addItem(self.translator.tr('import_type'), 'import')
        type_combo.addItem(self.translator.tr('sourceid_type'), 'sourceid')
        form.addRow(self.translator.tr('type'), type_combo)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(dlg.accept)
        btn_box.rejected.connect(dlg.reject)
        layout.addWidget(btn_box)
        if dlg.exec():
            target_field = target_field_combo.currentData()
            self.import_links.append({
                'source_table': src_table_combo.currentData(),
```

```
            'source_field': src_field_combo.currentData(),
            'target_entity_id': target_entity_combo.currentData(),
            'target_field': target_field_combo.currentData()
        })
        self.update_import_links_table()
    def update_src_fields(self, src_table_combo, src_field_combo):
        table_name = src_table_combo.currentData()
        src_field_combo.clear()
        for t in self.source_tables:
            if t['name'] == table_name:
                for f in t['fields']:
                    src_field_combo.addItem(f.name, f.name)
                break
    def update_target_fields(self, target_entity_combo, target_field_combo):
        entity_id = target_entity_combo.currentData()
        target_field_combo.clear()
        for ent in self.mozart_entities:
            if ent['id'] == entity_id:
                for fname in ent['fields']:
                    target_field_combo.addItem(fname, fname)
                break
    def update_import_links_table(self):
        self.links_table.setRowCount(len(self.import_links))
        for row, link in enumerate(self.import_links):
            self.links_table.setItem(row, 0, QTableWidgetItem(link['source_table']))
            self.links_table.setItem(row, 1, QTableWidgetItem(link['source_field']))
            entity_name = ''
            for ent in self.mozart_entities:
                if ent['id'] == link['target_entity_id']:
                    entity_name = ent['name']
                    break
            self.links_table.setItem(row, 2, QTableWidgetItem(entity_name))
            self.links_table.setItem(row, 3, QTableWidgetItem(link['target_field'] or ''))
            self.links_table.setItem(row, 4, QTableWidgetItem(link['type']))
            btn = QPushButton(self.translator.tr('delete'))
            btn.clicked.connect(lambda checked, r=row: self.remove_import_link(r))
            self.links_table.setCellWidget(row, 5, btn)
    def remove_import_link(self, row):
        self.import_links.pop(row)
        self.update_import_links_table()
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_main.py (отн

```
# mozart_import/designer/designer_main.py
# -*- coding: utf-8 -*-
import json
from PyQt5.QtWidgets import (QMainWindow, QSplitter, QVBoxLayout, QWidget, QTreeWidget,
                             QTreeWidgetItem, QPushButton, QMessageBox, QDialog,
                             QFormLayout, QLineEdit, QComboBox, QDialogButtonBox,
                             QTableWidget, QTableWidgetItem, QHeaderView, QMenu,
                             QAction, QTabWidget, QProgressDialog, QHBoxLayout,
                             QLabel, QCheckBox, QSpinBox, QFrame)
from PyQt5.QtCore import Qt
from database_import.database_service
from land.local_translator.import LocalTranslator
from ..core.models import ImportProfile, SourceType, FileFormat
from ..core.profile_manager import ProfileManager
from .designer_trees.import SourceTreeManager, MozartTreeManager
from .designer_links.import ImportLinksManager
from .designer_import.import ImportOperations

class ImportDesigner(QMainWindow):
    def __init__(self, db, connector, parent=None):
        super().__init__(parent)
        self.db = db
        self.connector = connector
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('import_designer_title'))
        self.resize(1200, 800)
        self.editing_profile_id = None
        central = QWidget()
        self.setCentralWidget(central)
        main_layout = QVBoxLayout(central)
        top_widget = QWidget()
        top_layout = QHBoxLayout(top_widget)
        self.splitter = QSplitter(Qt.Horizontal)
        top_layout.addWidget(self.splitter)
        left_tabs = QTabWidget()
        self.source_tree = QTreeWidget()
        self.source_tree.setHeaderLabels([self.translator.tr('source_tables'), self.translator.tr('rows')])
        self.source_tree.setContextMenuPolicy(Qt.CustomContextMenu)
        self.source_links_table = QTableWidget(0, 4)
        self.source_links_table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_table'),
            self.translator.tr('target_field')
        ])
        self.source_links_table.horizontalHeader().setStretchLastSection(True)
        left_tabs.addTab(self.source_tree, self.translator.tr('source_tables'))
        left_tabs.addTab(self.source_links_table, self.translator.tr('source_links'))
        self.splitter.addWidget(left_tabs)
        self.mozart_tree = QTreeWidget()
        self.mozart_tree.setHeaderLabels([self.translator.tr('mozart_entities'), ""])
        self.mozart_tree.setContextMenuPolicy(Qt.CustomContextMenu)
        self.splitter.addWidget(self.mozart_tree)
        main_layout.addWidget(top_widget, 2)
        splitter_handle = QFrame()
        splitter_handle.setFrameShape(QFrame.HLine)
        splitter_handle.setFrameShadow(QFrame.Sunken)
        splitter_handle.setCursor(Qt.SizeVerCursor)
        splitter_handle.mouseMoveEvent = self.start_splitter_drag
        splitter_handle.mouseReleaseEvent = self.stop_splitter_drag
        main_layout.addWidget(splitter_handle)
        self.splitter_handle = splitter_handle
        self.dragging = False
        self.drag_start_y = 0
        self.start_height = 0
        self.links_table = QTableWidget(0, 6)
        self.links_table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_entity'),
            self.translator.tr('target_field'),
            self.translator.tr('type'),
        ])
        self.links_table.horizontalHeader().setStretchLastSection(True)
        main_layout.addWidget(self.links_table, 1)
        btn_layout = QHBoxLayout()
        self.btn_add_link = QPushButton(self.translator.tr('add_import_link'))
        self.btn_add_link.clicked.connect(self.add_import_link)
        self.btn_save = QPushButton(self.translator.tr('save_profile'))
        self.btn_save.clicked.connect(self.save_profile)
        self.btn_import = QPushButton(self.translator.tr('start_import'))
        self.btn_import.clicked.connect(self.perform_import)
        self.btn_create_entity = QPushButton(self.translator.tr('create_entity'))
        self.btn_create_entity.clicked.connect(self.create_entity)
        btn_layout.addWidget(self.btn_create_entity)
        btn_layout.addWidget(self.btn_add_link)
        btn_layout.addWidget(self.btn_save)
        btn_layout.addWidget(self.btn_import)
        main_layout.addLayout(btn_layout)
        # Инициализация менеджеров
        self.source_manager = SourceTreeManager(self, self.source_tree, self.source_links_table,
                                                self.translator, self.connector)
        self.mozart_manager = MozartTreeManager(self, self.mozart_tree, self.translator, self.db)
        self.links_manager = None
        self.import_ops = None
        self.load_initial_data()
```

```
        self.setup_context_menus()
        self.links_table.setMinimumHeight(150)
        self.links_table.setMaximumHeight(400)
    def load_initial_data(self):
        self.source_manager.load_source()
        self.mozart_manager.load_mozart()
        self.links_manager = ImportLinksManager(self, self.links_table, self.translator,
                                                self.source_manager.source_tables,
                                                self.mozart_manager.mozart_entities)
        self.import_ops = ImportOperations(self, self.db, self.connector, self.translator,
                                           self.source_manager.source_tables,
                                           self.mozart_manager.mozart_entities,
                                           self.links_manager.import_links)
    def setup_context_menus(self):
        self.source_tree.customContextMenuRequested.connect(self.source_context_menu)
        self.mozart_tree.customContextMenuRequested.connect(self.mozart_context_menu)
    def source_context_menu(self, pos):
        item = self.source_tree.itemAt(pos)
        if not item or item.parent() is not None:
            return
        menu = QMenu()
        preview_action = QAction(self.translator.tr('preview_table'), None)
        preview_action.triggered.connect(lambda: self.source_manager.preview_table(item))
        menu.addAction(preview_action)
        add_link_action = QAction(self.translator.tr('add_source_link'), None)
        add_link_action.triggered.connect(lambda: self.source_manager.add_source_link(item))
        menu.addAction(add_link_action)
        select_all_action = QAction(self.translator.tr('select_all_fields'), None)
        select_all_action.triggered.connect(lambda: self.source_manager.set_checks_for_table(item, Qt.Checked))
        menu.addAction(select_all_action)
        deselect_all_action = QAction(self.translator.tr('deselect_all_fields'), None)
deselect_all_action.triggered.connect(lambda: self.source_manager.set_checks_for_table(item, Qt.Unchecked))
        menu.addAction(deselect_all_action)
        menu.exec_(self.source_tree.mapToGlobal(pos))
    def mozart_context_menu(self, pos):
        item = self.mozart_tree.itemAt(pos)
        if not item or item.parent() is not None:
            return
        menu = QMenu()
        create_action = QAction(self.translator.tr('create_fields_from_checked'), None)
create_action.triggered.connect(lambda: self.import_ops.create_fields_from_checked(item,
self.source_manager))
        menu.addAction(create_action)
        menu.exec_(self.mozart_tree.mapToGlobal(pos))
    def start_splitter_drag(self, event):
        self.dragging = True
        self.drag_start_y = event.globalY()
        self.start_height = self.links_table.height()
        self.setCursor(Qt.SizeVerCursor)
    def drag_splitter(self, event):
        if self.dragging:
            delta = event.globalY() - self.drag_start_y
            new_height = self.start_height + delta
            new_height = max(100, min(600, new_height))
            self.links_table.setMinimumHeight(new_height)
            self.links_table.setMaximumHeight(new_height)
    def stop_splitter_drag(self, event):
        self.dragging = False
        self.setCursor(Qt.ArrowCursor)
    def add_import_link(self):
        self.links_manager.add_import_link()
        self.import_ops.set_import_links(self.links_manager.import_links)
    def save_profile(self):
        # 1. Сбор данных о таблицах источника
        source_tables = []
        for table in self.source_manager.source_tables:
            sample_rows = table.sample_rows[:5] if table.sample_rows else []
            source_tables.append({
                'name': table.name,
                'row_count': self.connector.get_row_count(table.name),
                'sample_rows': sample_rows
            })
        # 2. Сбор целевых сущностей
        target_entities = []
        seen = set()
        for link in self.links_manager.import_links:
            eid = link['target_entity_id']
            if eid not in seen:
                for ent in self.mozart_manager.mozart_entities:
                    if ent['id'] == eid:
                        target_entities.append({'id': eid, 'name': ent['cname']})
                        seen.add(eid)
                        break
        # 3. Собираем связи импорта
        import_links = []
        for link in self.links_manager.import_links:
            mappings = []
            mappings.append({
                'source_field': link['source_field'],
                'target_field': link['target_field'],
                'field_type': 'C',
                'field_length': 255,
                'is_source_id': (link['type'] == 'sourceid')
            })
            import_links.append({
                'source_table': link['source_table'],
                'target_entity_id': link['target_entity_id'],
                'mappings': mappings
            })
        # 4. Сбор параметров подключения
        file_format = None
        db_type = None
        db_host = None
        db_port = None
        db_name = None
        db_user = None
        db_pass = None
        if hasattr(self.connector, 'file_format'):
            # FileConnector
file_format = self.connector.file_format.value if isinstance(self.connector.file_format, FileFormat) else
str(self.connector.file_format)
        elif hasattr(self.connector, 'db_type'):
            # DBConnector
            db_type = self.connector.db_type
            db_host = self.connector.host
            db_port = self.connector.port
            db_name = self.connector.database
            db_user = self.connector.user
            db_pass = self.connector.password
        # URLConnector не имеет дополнительных полей, кроме source_path
        # Определяем тип источника
        if hasattr(self.connector, 'file_format'):
            source_type = SourceType.FILE
        elif hasattr(self.connector, 'db_type'):
            source_type = SourceType.DATABASE
        else:
            source_type = SourceType.URL
        profile = ImportProfile(
            name=self.connector.source_path,
            source_type=source_type,
            source_path=self.connector.source_path,
            source_tables=source_tables,
            target_entities=target_entities,
            import_links=import_links,
            source_fk_links=self.source_manager.source_fk_links,
            file_format=file_format,
            db_type=db_type,
            db_host=db_host,
            db_port=db_port,
            db_name=db_name,
            db_user=db_user,
            db_pass=db_pass
        )
        if self.editing_profile_id:
            profile.id = self.editing_profile_id
        prof_mgr = ProfileManager(self.db)
```

```
        prof_mgr.save_profile(profile)
        QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('profile_saved'))
    def perform_import(self):
        self.import_ops.set_import_links(self.links_manager.import_links)
        self.import_ops.perform_import()
    def create_entity(self):
        from .wizards.wizard_dialogs import NewEntityDialog
        dlg = NewEntityDialog(self, self.db)
        if dlg.exec():
            data = dlg.get_data()
            res = self.db.execute_query(
"""INSERT INTO meta_entitytypes (cname, calias, class_id, chaseclass, lisversioned, isecuritystrategy, status_id)
                VALUES (%s, %s, %s, true, 1, 1) RETURNING id""",
                (data['cname'], data['calias'], data['class_id'])
            )
            if res:
                self.mozart_manager.load_mozart()
                self.links_manager.set_mozart_entities(self.mozart_manager.mozart_entities)
                self.import_ops.set_mozart_entities(self.mozart_manager.mozart_entities)
                QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('entity_created'))
    def set_profile_for_edit(self, profile):
        self.editing_profile_id = profile.id
        self.restore_profile(profile)
    def restore_profile(self, profile):
        self.source_manager.source_tables = []
        self.source_manager.source_fk_links = profile.source_fk_links
        self.source_manager.update_source_links_table()
        self.links_manager.import_links = []
        for link in profile.import_links:
            for mapping in link['mappings']:
                self.links_manager.import_links.append({
                    'source_table': link['source_table'],
                    'source_field': mapping['source_field'],
                    'target_entity_id': link['target_entity_id'],
                    'target_field': mapping['target_field'],
                    'type': 'sourceid' if mapping.get('is_source_id') else 'import'
                })
        self.links_manager.update_import_links_table()
        self.import_ops.set_import_links(self.links_manager.import_links)
        QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('profile_loaded'))
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_mass_field_d

```
# mozart_import/designer/designer_mass_field_dialog.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QTableWidget, QTableWidgetItem, QComboBox,
QSpinBox, QDialogButtonBox
from PyQt5.QtCore import Qt
from .designer_utils import to_field_name

class MassFieldDialog(QDialog):
    def __init__(self, parent, source_fields, translator):
        super().__init__(parent)
        self.translator = translator
        self.setWindowTitle(self.translator.tr('mass_field_dialog_title'))
        self.resize(800, 500)
        layout = QVBoxLayout(self)
        self.table = QTableWidget(len(source_fields), 4)
        self.table.setHorizontalHeaderLabels([
            self.translator.tr('source_field'),
            self.translator.tr('mozart_field_name'),
            self.translator.tr('type'),
            self.translator.tr('length')
        ])
        self.table.horizontalHeader().setStretchLastSection(True)
        for row, sf in enumerate(source_fields):
            self.table.setItem(row, 0, QTableWidgetItem(sf['source_field_name']))
            suggested = to_field_name(sf['source_field_name'])
            name_item = QTableWidgetItem(suggested)
            self.table.setItem(row, 1, name_item)
            type_combo = QComboBox()
            type_combo.addItems(["N", "I", "D", "T", "L", "M", "R"])
            type_combo.setCurrentText("C")
            self.table.setCellWidget(row, 2, type_combo)
            length_spin = QSpinBox()
            length_spin.setRange(0, 10000)
            length_spin.setValue(255)
            self.table.setCellWidget(row, 3, length_spin)
        layout.addWidget(self.table)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self.accept)
        btn_box.rejected.connect(self.reject)
        layout.addWidget(btn_box)
    def get_fields(self):
        fields = []
        for row in range(self.table.rowCount()):
            source_field = self.table.item(row, 0).text()
            target_field = self.table.item(row, 1).text()
            type_combo = self.table.cellWidget(row, 2)
            length_spin = self.table.cellWidget(row, 3)
            if target_field:
                fields.append({
                    'source_field': source_field,
                    'fieldname': target_field,
                    'fieldtype': type_combo.currentText(),
                    'length': length_spin.value(),
                    'calias': source_field.capitalize()[:50]
                })
        return fields
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_models.py (с

```
# mozart_import/designer/designer_models.py
# -*- coding: utf-8 -*-
import re
def to_field_name(source_name: str) -> str:
    """Преобразует имя исходного поля в допустимое имя поля PostgreSQL (латиница, подчёркивания)."""
    name = re.sub(r'[^a-zA-Z0-9_]', '_', source_name)
    if name[0].isdigit():
        name = '_' + name
    return name.lower()[:63]
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_trees.py (отн

```
# mozart_import/designer/designer_trees.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QTreeWidgetItem, QMenu, QAction, QMessageBox, QDialog, QFormLayout,
QComboBox, QVBoxLayout, QDialogButtonBox, QLineEdit, QTableWidget, QTableWidgetItem
from PyQt5.QtCore import Qt
from .designer_utils import to_field_name

class SourceTreeManager:
    """Управление деревом таблиц источника."""
    def __init__(self, parent, source_tree, source_links_table, translator, connector):
        self.parent = parent
        self.source_tree = source_tree
        self.source_links_table = source_links_table
        self.translator = translator
        self.connector = connector
        self.source_tables = []
        self.source_fk_links = []
    def load_source(self):
        self.source_tree.clear()
```

```
        schema = self.connector.get_schema()
        self.source_tables = schema.tables
        for table in self.source_tables:
            row_count = self.connector.get_row_count(table.name)
            count_str = str(row_count) if row_count is not None else '?'
            item = QTreeWidgetItem([table.name, count_str])
            item.setData(0, Qt.UserRole, table.name)
            for field in table.fields:
                field_item = QTreeWidgetItem([f"  {field.name} ({field.field_type.name})", ""])
                field_item.setData(0, Qt.UserRole, field.name)
                field_item.setFlags(field_item.flags() | Qt.ItemIsUserCheckable)
                field_item.setCheckState(0, Qt.Unchecked)
                item.addChild(field_item)
            self.source_tree.addTopLevelItem(item)
        self.detect_source_fk_links()
        self.update_source_links_table()
    def detect_source_fk_links(self):
        self.source_fk_links = []
        for table in self.source_tables:
            for field in table.fields:
                if field.is_foreign_key and field.refers_to:
                    self.source_fk_links.append({
                        'source_table': table.name,
                        'source_field': field.name,
                        'target_table': field.refers_to,
                        'target_field': 'id'
                    })
    def update_source_links_table(self):
        self.source_links_table.setRowCount(len(self.source_fk_links))
        for row, link in enumerate(self.source_fk_links):
            self.source_links_table.setItem(row, 0, QTableWidgetItem(link['source_table']))
            self.source_links_table.setItem(row, 1, QTableWidgetItem(link['source_field']))
            self.source_links_table.setItem(row, 2, QTableWidgetItem(link['target_table']))
            self.source_links_table.setItem(row, 3, QTableWidgetItem(link['target_field']))
    def get_checked_fields(self):
        checked = []
        root = self.source_tree.invisibleRootItem()
        for i in range(root.childCount()):
            table_item = root.child(i)
            for j in range(table_item.childCount()):
                field_item = table_item.child(j)
                if field_item.checkState(0) == Qt.Checked:
                    checked.append({
                        'source_field_name': field_item.data(0, Qt.UserRole),
                        'source_table': table_item.data(0, Qt.UserRole)
                    })
        return checked
    def set_checks_for_table(self, table_item, state):
        for i in range(table_item.childCount()):
            field_item = table_item.child(i)
            field_item.setCheckState(0, state)
    def add_source_link(self, table_item):
        table_name = table_item.data(0, Qt.UserRole)
        dlg = QDialog(self.parent)
        dlg.setWindowTitle(self.translator.tr('add_source_link_title'))
        layout = QVBoxLayout(dlg)
        form = QFormLayout()
        source_field_combo = QComboBox()
        for t in self.source_tables:
            if t.name == table_name:
                for f in t.fields:
                    source_field_combo.addItem(f.name, f.name)
                break
        form.addRow(self.translator.tr('source_field'), source_field_combo)
        target_table_combo = QComboBox()
        for t in self.source_tables:
            target_table_combo.addItem(t.name, t.name)
        form.addRow(self.translator.tr('target_table'), target_table_combo)
        target_field_combo = QComboBox()
target_table_combo.currentIndexChanged.connect(lambda: self.update_target_fields(target_table_combo,
target_field_combo))
        self.update_target_fields(target_table_combo, target_field_combo)
        form.addRow(self.translator.tr('target_field'), target_field_combo)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(dlg.accept)
        btn_box.rejected.connect(dlg.reject)
        layout.addWidget(btn_box)
        if dlg.exec_():
            self.source_fk_links.append({
                'source_table': table_name,
                'source_field': source_field_combo.currentData(),
                'target_table': target_table_combo.currentData(),
                'target_field': target_field_combo.currentData()
            })
            self.update_source_links_table()
    def update_target_fields(self, target_table_combo, target_field_combo):
        target_table = target_table_combo.currentData()
        target_field_combo.clear()
        for t in self.source_tables:
            if t.name == target_table:
                for f in t.fields:
                    target_field_combo.addItem(f.name, f.name)
                break
    def preview_table(self, table_item):
        table_name = table_item.data(0, Qt.UserRole)
        sample = self.connector.get_table_sample(table_name, limit=20)
        if not sample:
            QMessageBox.information(self.parent, self.translator.tr('info'), self.translator.tr('no_data'))
            return
        dlg = QDialog(self.parent)
        dlg.setWindowTitle(self.translator.tr('preview_table_title').format(table_name))
        layout = QVBoxLayout(dlg)
        table = QTableWidget()
        headers = list(sample[0].keys())
        table.setColumnCount(len(headers))
        table.setHorizontalHeaderLabels(headers)
        table.setRowCount(len(sample))
        for row, r in enumerate(sample):
            for col in enumerate(headers):
                table.setItem(row, col, QTableWidgetItem(str(r.get(col, ''))))
        layout.addWidget(table)
        dlg.exec_()

class MozartTreeManager:
    def __init__(self, parent, mozart_tree, translator, db):
        self.parent = parent
        self.mozart_tree = mozart_tree
        self.translator = translator
        self.db = db
        self.mozart_entities = []
    def load_mozart(self):
        self.mozart_tree.clear()
rows = self.db.execute_query("SELECT id, cname, calias, class_id FROM meta.entitytypes ORDER BY cname")
        self.mozart_entities = []
        for eid, cname, calias, class_id in rows:
            class_rows = self.db.execute_query("SELECT calias FROM meta.entity_classes WHERE id=%s", (class_id,))
            class_alias = class_rows[0][0] if class_rows else "DIC"
fields = self.db.execute_query("SELECT cfieldname, cfieldtype FROM meta.fields WHERE entitytypeid=%s",
(eid,))
            item = QTreeWidgetItem([f"{cname} ({calias} - {class_alias})", ""])
            item.setData(0, Qt.UserRole, eid)
            for fname, ftype in fields:
                fitem = QTreeWidgetItem([f"  {fname} ({ftype})", ""])
                fitem.setData(0, Qt.UserRole, fname)
                item.addChild(fitem)
            self.mozart_tree.addTopLevelItem(item)
            self.mozart_entities.append({
                'id': eid, 'cname': cname, 'calias': calias, 'class_alias': class_alias, 'fields': [f[0] for f in fields]
            })
        return self.mozart_entities
    def get_entity_by_id(self, entity_id):
        for ent in self.mozart_entities:
            if ent['id'] == entity_id:
                return ent
        return None
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_utils.py (отн

```
# mozart_import/designer/designer_utils.py
# -*- coding: utf-8 -*-
import re
def to_field_name(source_name: str) -> str:
    """Преобразует имя исходного поля в допустимое имя поля PostgreSQL (латиница, подчёркивания)."""
    name = re.sub(r'[^a-zA-Z0-9_]', '_', source_name)
    if name[0].isdigit():
        name = 'f_' + name
    return name.lower()[:63]
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/designer_v2.py (относ

```
# mozart_import/designer/designer_v2.py (часть 1)
# -*- coding: utf-8 -*-
from ..core.import_orchestrator import ImportOrchestrator
from PyQt5.QtWidgets import (QMainWindow, QVBoxLayout, QWidget, QPushButton,
                             QHBoxLayout, QMessageBox, QDialog, QTableWidget,
                             QTableWidgetItem)
from PyQt5.QtCore import Qt
from database.import_database_service
from lang.local_translator import LocalTranslator
from ..graph.graphics_view import GraphicsView
from ..graph.relation_scene import RelationScene
from ..graph.graphics_items import SOURCE_B_FIELD
from ..core.models.import_models, NodeType, Field, DataType, LinkType,
                                  ImportProfile, SourceType
from ..core.profile_manager import ProfileManager
from ..core.import_engine import ImportEngine
from ..utils.field_creator import FieldCreator
from ..utils.ext_helper import ExtHelper
from ..entity_creator.entity_creator import EntityCreator
from ..link_editor.link_editor_dialog import LinkEditorDialog
from ..source_ref_dialog.source_ref_dialog import SourceRefDialog
from .scene_profile_adapter import SceneProfileAdapter

class ImportDesignerV2(QMainWindow):
    """Визуальный конструктор импорта."""
    def __init__(self, db: DatabaseService, connector, parent=None):
        super().__init__(parent)
        self.db = db
        self.connector = connector
        self.translator = LocalTranslator()
        self.editing_profile_id = None
        self.field_creator = FieldCreator(self.db)
        self.ext_helper = ExtHelper(self.db)
        self.entity_creator = EntityCreator(self.db)
        self.setWindowTitle(self.translator.tr('import_designer_title'))
        self.resize(1400, 800)
        central = QWidget()
        self.setCentralWidget(central)
        main_layout = QVBoxLayout(central)
        # Сцена
        self.scene = RelationScene(self)
        self.scene.on_preview_table = self.preview_table
        self.scene.on_create_entity = self.create_entity_from_source
        self.scene.on_create_fields_from_source = self.create_fields_for_mozart
        self.scene.on_edit_link = self.edit_link_between
        self.scene.on_set_source_refs = self.edit_source_refs
        self.view = GraphicsView(self.scene)
        main_layout.addWidget(self.view, 1)
        # Панель управления
        btn_layout = QHBoxLayout()
        self.btn_save = QPushButton(self.translator.tr('save_profile'))
        self.btn_save.clicked.connect(self.save_profile)
        self.btn_import = QPushButton(self.translator.tr('start_import'))
        self.btn_import.clicked.connect(self.perform_import)
        btn_layout.addWidget(self.btn_save)
        btn_layout.addWidget(self.btn_import)
        main_layout.addLayout(btn_layout)
        self.load_source_tables()
        self.load_mozart_entities()
    # ------------------------------------------------------------------
    # Загрузка таблиц источника
    # ------------------------------------------------------------------
    def load_source_tables(self):
        schema = self.connector.get_schema()
        divider = self.scene.divider_x
        x, y = 30, 30
        for table in schema.tables:
            fields = []
            for f in table.fields:
                dt = DataType.STRING
                if f.field_type_name == 'INTEGER':
                    dt = DataType.INTEGER
                elif f.field_type_name == 'FLOAT':
                    dt = DataType.FLOAT
                elif f.field_type_name == 'BOOLEAN':
                    dt = DataType.BOOLEAN
                elif f.field_type_name == 'DATE':
                    dt = DataType.DATE
                fields.append(Field(name=f.name, data_type=dt,
                                    is_primary_key=f.is_primary_key,
                                    is_foreign_key=f.is_foreign_key,
                                    references=f.refers_to))
            node = Node(id=f"src_{table.name}", name=table.name,
                        node_type=NodeType.SOURCE_TABLE, fields=fields,
                        x=x, y=y, table_name=table.name)
            node.refs = []
            self.scene.add_node(node)
            y += 180
            if y > self.scene.height() - 200:
                y = 30
                x += 280
                if x > divider - 240:
                    break
        for node_item in list(self.scene.nodes.values()):
            if node_item.node.node_type != NodeType.SOURCE_TABLE:
                continue
            for field in node_item.node.fields:
                if field.is_foreign_key and field.references:
                    for target_item in self.scene.nodes.values():
                        if (target_item.node.table_name == field.references
                            and target_item.node.node_type == NodeType.SOURCE_TABLE):
                            pk = next((f for f in target_item.node.fields if f.is_primary_key), None)
                            if pk:
                                src_fi = self._find_field_item(node_item, field.name)
                                tgt_fi = self._find_field_item(target_item, pk.name)
                                if src_fi and tgt_fi:
                                    self.scene.add_link(src_fi, tgt_fi, LinkType.FK_SOURCE)
    # ------------------------------------------------------------------
    # Загрузка сущностей Mozart
    # ------------------------------------------------------------------
    def load_mozart_entities(self):
        existing_links = []
        for link in self.scene.links:
            if link.link_type == LinkType.IMPORT:
                existing_links.append({
                    'source_table': link.start_item.node.table_name,
                    'source_field': link.start_field_item.field.name,
                    'target_entity_id': link.end_item.node.entity_id,
                    'target_field': link.end_field_item.field.name,
                    'mappings': link.mappings
                })
        to_remove = [nid for nid, ni in self.scene.nodes.items()
                     if ni.node.node_type == NodeType.MOZART_ENTITY]
        for nid in to_remove:
            self.scene.removeItem(self.scene.nodes[nid])
            del self.scene.nodes[nid]
        for link in self.scene.links[:]:
```

```
        if link.link_type == LinkType.IMPORT:
            self.scene.removeItem(link)
            self.scene.links.remove(link)
    rows = self.db.execute_query(
        "SELECT id, cname, calias, class_id FROM meta.entitytypes ORDER BY cname"
    )
    divider = self.scene.divider_x
    x, y = divider + 30, 30
    for eid, cname, calias, class_id in rows:
        class_res = self.db.execute_query(
            "SELECT calias FROM meta.entity_classes WHERE id=%s", (class_id,))
        class_alias = class_res[0][0] if class_res else "DICT"
        self.ext_helper.ensure_field(
            entity_id=eid,
            field_name=SOURCE_ID_FIELD,
            field_type='C',
            calias='Source ID',
            length=255
        )
        fields = []
        frows = self.db.execute_query(
            "SELECT cfieldname, cfieldtype FROM meta.fields WHERE entitytypeid=%s ORDER BY isortorder",
            (eid,))
        for fname, ftype in frows:
            dt = DataType.STRING
            if ftype == 'I': dt = DataType.INTEGER
            elif ftype == 'N': dt = DataType.FLOAT
            elif ftype == 'L': dt = DataType.BOOLEAN
            fields.append(Field(name=fname, data_type=dt))
        node = Node(id=f"moz_{eid}", name=f"{cname} ({calias})",
                    node_type=NodeType.MOZART_ENTITY, fields=fields,
                    x=x, y=y, entity_id=eid, class_alias=class_alias)
        self.scene.add_node(node)
        x += 180
        if y > self.scene.height() - 200:
            y += 30
            x += 280
    for ld in existing_links:
        src_node = self._find_node_by_table(ld['source_table'])
        tgt_node = self._find_node_by_entity(ld['target_entity_id'])
        if src_node and tgt_node:
            self.scene.add_link(src_node, tgt_node, LinkType.IMPORT, mappings=ld.get('mappings'))
# ------------------------------------------------------------------
# Диалог настройки связи (импорта)
# ------------------------------------------------------------------
def edit_link_between(self, source_node_item, mozart_node_item, existing_mappings, checked_source_fields,
source_refs):
    if not checked_source_fields:
        checked_source_fields = [fi.field.name for fi in source_node_item.field_items]
    target_fields = [fi.field.name for fi in mozart_node_item.field_items]
    if SOURCE_ID_FIELD not in target_fields:
        target_fields.append(SOURCE_ID_FIELD)
    dlg = LinkEditDialog(checked_source_fields, target_fields, existing_mappings, source_refs, self)
    if dlg.exec():
        new_mappings = dlg.get_mappings()
        if not new_mappings:
            return
        old_link = self.scene.find_link_between(source_node_item, mozart_node_item)
        if old_link:
            self.scene.removeItem(old_link)
            self.scene.links.remove(old_link)
        self.scene.add_link(source_node_item, mozart_node_item, LinkType.IMPORT, mappings=new_mappings)
        self.scene.update_all_links()
        QMessageBox.information(self, "Связь обновлена", "Маппинг полей применён.")
# ------------------------------------------------------------------
# Диалог настройки ссылочных полей источника
# ------------------------------------------------------------------
def edit_source_refs(self, source_node_item):
    table_name = source_node_item.node.table_name
    fields = [fi.field.name for fi in source_node_item.field_items]
    all_tables = []
    for ni in self.scene.nodes.values():
        if ni.node.node_type == NodeType.SOURCE_TABLE:
            all_tables.append(ni.node.table_name)
    current_refs = getattr(source_node_item.node, 'refs', {})
    dlg = SourceRefDialog(table_name, fields, all_tables, current_refs, self)
    if dlg.exec():
        new_refs = dlg.get_refs()
        for ni in self.scene.nodes.values():
            if ni.node.node_type == NodeType.SOURCE_TABLE and ni.node.table_name == table_name:
                ni.node.refs = new_refs
                break
        QMessageBox.information(self, "Связность", "Настройки ссылочных полей сохранены.")
# ------------------------------------------------------------------
# Превью, создание сущности, создание полей
# ------------------------------------------------------------------
def preview_table(self, node_item):
    table_name = node_item.node.table_name
    try:
        sample = self.connector.get_table_sample(table_name, limit=100)
    except Exception as e:
        QMessageBox.warning(self, "Ошибка", f"Не удалось получить данные: {e}")
        return
    if not sample:
        QMessageBox.information(self, self.translator.tr('info'),
                                self.translator.tr('no_data'))
        return
    dlg = QDialog(self)
    dlg.setWindowTitle(table_name)
    layout = QVBoxLayout(dlg)
    table = QTableWidget()
    headers = list(sample[0].keys())
    table.setColumnCount(len(headers))
    table.setHorizontalHeaderLabels(headers)
    table.setRowCount(len(sample))
    for r, row in enumerate(sample):
        for c, col in enumerate(headers):
            table.setItem(r, c, QTableWidgetItem(str(row.get(col, ''))))
    layout.addWidget(table)
    dlg.exec()
def create_entity_from_source(self, source_node_item):
    from ..wizards.wizard_dialogs import NewEntityDialog
    dlg = NewEntityDialog(self, self.db)
    dlg.name_edit.setText(source_node_item.node.name.capitalize())
    dlg.alias_edit.setText(source_node_item.node.name.lower().replace(' ', '_'))
    if dlg.exec():
        data = dlg.get_data()
        sample = None
        try:
            sample_rows = self.connector.get_table_sample(
                source_node_item.node.table_name, limit=1)
            if sample_rows:
                sample = sample_rows[0]
        except Exception:
            pass
        checked = source_node_item.get_checked_fields()
        entity_id = self.entity_creator.create_entity_from_source(
            entity_name=data['name'],
            entity_calias=data['calias'],
            class_id=data['class_id'],
            source_table_name=source_node_item.node.table_name,
            checked_fields=checked,
            source_sample=sample
        )
        if entity_id:
            self.load_mozart_entities()
            QMessageBox.information(self, self.translator.tr('info'),
                                    self.translator.tr('entity_created'))
def create_fields_for_mozart(self, mozart_node_item):
    src_item = None
    for ni in self.scene.nodes.values():
        if ni.node.node_type == NodeType.SOURCE_TABLE and ni.is_checked():
            src_item = ni
            break
    if not src_item:
```

```
            QMessageBox.warning(self, "Предупреждение", "Отметьте таблицу источника")
            return
        sample = None
        try:
            sample_rows = self.connector.get_table_sample(
                src_item.node.table_name, limit=1)
            if sample_rows:
                sample = sample_rows[0]
        except Exception:
            pass
        fields_to_create = []
        for src_field in src_item.get_checked_fields():
            from designer_utils import to_field_name
            mozart_name = to_field_name(src_field)
            field_type = 'C'
            length = 255
            if sample and src_field in sample:
                val = sample[src_field]
                if isinstance(val, int):
                    field_type = 'I'; length = 0
                elif isinstance(val, float):
                    field_type = 'N'; length = 15
                elif isinstance(val, bool):
                    field_type = 'L'; length = 0
            fields_to_create.append({
                'fieldname': mozart_name,
                'fieldtype': field_type,
                'alias': src_field.capitalize()[:50],
                'length': length,
                'isindexed': True
            })
        if fields_to_create:
            self.field_creator.create_fields(mozart_node_item.node.entity_id, fields_to_create)
            self.load_mozart_entities()
            QMessageBox.information(self, "Поля созданы",
                                    f"Создано {len(fields_to_create)} полей")
    # ---------------------------------------------------------------
    # Сохранение и импорт
    # ---------------------------------------------------------------
    def save_profile(self):
        profile = SceneProfileAdapter.scene_to_profile(self.scene, self.connector, self.db, self.editing_profile_id)
        prof_mgr = ProfileManager(self.db)
        prof_mgr.save_profile(profile)
        QMessageBox.information(self, self.translator.tr('info'),
                                self.translator.tr('profile_saved'))
    def perform_import(self):
        profile = SceneProfileAdapter.scene_to_profile(
            self.scene, self.connector, self.editing_profile_id)
        orchestrator = ImportOrchestrator(self.db, self.connector, profile)
        try:
            stats = orchestrator.run()
            msg = self.translator.tr('import_finished_stats').format(
                stats['inserted'], stats['updated'], stats['errors'])
            QMessageBox.information(self, self.translator.tr('import_finished_title'), msg)
            if orchestrator.errors:
                # Показываем первые 20 ошибок
                error_text = "\n".join(orchestrator.errors[:20])
                if len(orchestrator.errors) > 20:
                    error_text += f"\n... и ещё {len(orchestrator.errors)-20} ошибок"
                QMessageBox.warning(self, "Предупреждение", error_text)
            # Дополнительно выведем в консоль первые 10 ошибок из stats, если есть 'details'
            if 'details' in stats:
                print("Детали ошибок из движка:")
                for err in stats['details'][:10]:
                    print(err)
        except Exception as e:
            QMessageBox.critical(self, self.translator.tr('error'), str(e))
    def load_profile_links(self, profile: ImportProfile):
        for link in list(self.scene.links):
            if link.link_type == LinkType.IMPORT:
                self.scene.removeItem(link)
                self.scene.links.remove(link)
        SceneProfileAdapter.restore_scene(self.scene, profile)
    # ---------------------------------------------------------------
    # Вспомогательные методы
    # ---------------------------------------------------------------
    def _find_field_item(self, node_item, field_name):
        for fi in node_item.field_items:
            if fi.field_name == field_name:
                return fi
        return None
    def _find_node_by_table(self, table_name):
        for ni in self.scene.nodes.values():
            if ni.node.node_type == NodeType.SOURCE_TABLE and ni.node.table_name == table_name:
                return ni
        return None
    def _find_node_by_entity(self, entity_id):
        for ni in self.scene.nodes.values():
            if ni.node.node_type == NodeType.MOZART_ENTITY and ni.node.entity_id == entity_id:
                return ni
        return None
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/entity_creator.py (отн

```
# mozart_import/designer/entity_creator.py
# -*- coding: utf-8 -*-
from typing import Dict, Any, Optional
from database import DatabaseService
from ..utils.field_creator import FieldCreator
from ..utils.ext_helper import ExtHelper
from lang.local_translator import LocalTranslator

class EntityCreator:
    """
    Создаёт новую сущность Mozart на основе таблицы источника.
    Копирует все отмеченные поля из источника в сущность.
    """
    def __init__(self, db: DatabaseService):
        self.db = db
        self.field_creator = FieldCreator(db)
        self.ext_helper = ExtHelper(db)
        self.translator = LocalTranslator()
    def create_entity_from_source(self,
                                  entity_name: str,
                                  entity_alias: str,
                                  class_id: int,
                                  source_table_name: str,
                                  checked_fields: list,  # список имён полей источника
                                  source_sample: Optional[Dict[str, Any]] = None) -> Optional[int]:
        """
        Создаёт тип сущности, ext-таблицу и поля meta.fields.
        Возвращает entity_id новой сущности.
        """
        # 1. Создаём запись в meta.entitytypes
        res = self.db.execute_query(
            "INSERT INTO meta.entitytypes "
            "(name, alias, class_id, cbaseclass, isversioned, isecuritystrategy, status_id) "
            "VALUES (%s, %s, %s, true, 1, 1) RETURNING id",
            (entity_name, entity_alias, class_id)
        )
        if not res:
            return None
        entity_id = res[0][0]
        # 2. Обеспечиваем существование ext-таблицы
        self.ext_helper.ensure_ext_table(entity_id)
        # 3. Создаём поля
        fields_to_create = []
        for src_field in checked_fields:
            # Переводим имя поля в Mozart (латиница, без спецсимволов)
            from designer_utils import to_field_name
            mozart_field = to_field_name(src_field)
            # Определяем тип по образцу (если есть)
            field_type = 'C'  # строка по умолчанию
            length = 255
            if source_sample and src_field in source_sample:
```

```
            val = source_sample[src_field]
            if isinstance(val, int):
                field_type = 'I'
                length = 0
            elif isinstance(val, float):
                field_type = 'N'
                length = 15
            elif isinstance(val, bool):
                field_type = 'L'
                length = 0
            else:
                field_type = 'C'
                length = max(255, len(str(val)) * 2)
        fields_to_create.append({
            'fieldname': mozart_field,
            'fieldtype': field_type,
            'caption': src_field.capitalize()[:50],
            'length': length,
            'isindexed': True
        })
    if fields_to_create:
        self.field_creator.create_fields(entity_id, fields_to_create)
    return entity_id
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/import_designer.py (o

```
# mozart_import/designer/designer_main.py
# -*- coding: utf-8 -*-
import json
from PyQt5.QtWidgets import (QMainWindow, QSplitter, QVBoxLayout, QWidget, QTreeWidget,
                             QTreeWidgetItem, QPushButton, QMessageBox, QDialog,
                             QFormLayout, QLineEdit, QComboBox, QDialogButtonBox,
                             QTableWidget, QTableWidgetItem, QHeaderView, QMenu,
                             QAction, QTabWidget, QProgressDialog, QHBoxLayout,
                             QLabel, QCheckBox, QSpinBox, QFrame)
from PyQt5.QtCore import Qt
from database.import_database_service
from language.translator.translator import LocalTranslator
from ..core.models.import_models import (SourceTable, SourceField, FieldType, ImportProfile,
                                         SourceType, MappingProposal, MatchedEntity, MatchedField)
from ..utils.field_creator.field_creator import FieldCreator
from ..core.import_engine.import_engine import ImportEngine
from ..core.profile_manager.profile_manager import ProfileManager
from .designer_mass_field_dialog import MassFieldDialog

class ImportDesigner(QMainWindow):
    def __init__(self, db_connector, parent=None):
        super().__init__(parent)
        self.db = db
        self.connector = connector
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('import_designer_title'))
        self.resize(1200, 800)
        self.editing_profile_id = None
        central = QWidget()
        self.setCentralWidget(central)
        main_layout = QVBoxLayout(central)
        top_widget = QWidget()
        top_layout = QHBoxLayout(top_widget)
        self.splitter = QSplitter(Qt.Horizontal)
        top_layout.addWidget(self.splitter)
        left_tabs = QTabWidget()
        self.source_tree = QTreeWidget()
        self.source_tree.setHeaderLabels([self.translator.tr('source_tables'), self.translator.tr('rows')])
        self.source_tree.setContextMenuPolicy(Qt.CustomContextMenu)
        self.source_tree.customContextMenuRequested.connect(self.source_context_menu)
        left_tabs.addTab(self.source_tree, self.translator.tr('source_tables'))
        self.source_links_table = QTableWidget(0, 4)
        self.source_links_table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_table'),
            self.translator.tr('target_field')
        ])
        self.source_links_table.horizontalHeader().setStretchLastSection(True)
        left_tabs.addTab(self.source_links_table, self.translator.tr('source_links'))
        self.splitter.addWidget(left_tabs)
        self.mozart_tree = QTreeWidget()
        self.mozart_tree.setHeaderLabels([self.translator.tr('mozart_entities'), ""])
        self.mozart_tree.setContextMenuPolicy(Qt.CustomContextMenu)
        self.mozart_tree.customContextMenuRequested.connect(self.mozart_context_menu)
        self.splitter.addWidget(self.mozart_tree)
        main_layout.addWidget(top_widget, 2)
        splitter_handle = QFrame()
        splitter_handle.setFrameShape(QFrame.HLine)
        splitter_handle.setFrameShadow(QFrame.Sunken)
        splitter_handle.setCursor(Qt.SizeVerCursor)
        splitter_handle.mousePressEvent = self.start_splitter_drag
        splitter_handle.mouseMoveEvent = self.drag_splitter
        splitter_handle.mouseReleaseEvent = self.stop_splitter_drag
        main_layout.addWidget(splitter_handle)
        self.splitter_handle = splitter_handle
        self.dragging = False
        self.drag_start_y = 0
        self.start_height = 0
        self.links_table = QTableWidget(0, 6)
        self.links_table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_entity'),
            self.translator.tr('target_field'),
            self.translator.tr('type'),
        ])
        self.links_table.horizontalHeader().setStretchLastSection(True)
        main_layout.addWidget(self.links_table, 1)
        btn_layout = QHBoxLayout()
        self.btn_add_link = QPushButton(self.translator.tr('add_import_link'))
        self.btn_add_link.clicked.connect(self.add_import_link)
        self.btn_save = QPushButton(self.translator.tr('save_profile'))
        self.btn_save.clicked.connect(self.save_profile)
        self.btn_import = QPushButton(self.translator.tr('start_import'))
        self.btn_import.clicked.connect(self.perform_import)
        self.btn_create_entity = QPushButton(self.translator.tr('create_entity'))
        self.btn_create_entity.clicked.connect(self.create_entity)
        btn_layout.addWidget(self.btn_create_entity)
        btn_layout.addWidget(self.btn_add_link)
        btn_layout.addWidget(self.btn_save)
        btn_layout.addWidget(self.btn_import)
        main_layout.addLayout(btn_layout)
        self.source_tables = []
        self.mozart_entities = []
        self.source_fk_links = []
        self.import_links = []
        self.load_source()
        self.load_mozart()
        self.retranslate_ui()
        self.links_table.setMinimumHeight(150)
        self.links_table.setMaximumHeight(400)
    def set_profile_for_edit(self, profile):
        self.editing_profile_id = profile.id
        self.restore_profile(profile)
    def restore_profile(self, profile):
        self.source_tables = []
        self.import_links = []
        self.source_fk_links = []
        self.source_tree.clear()
        self.source_links_table.setRowCount(0)
        self.links_table.setRowCount(0)
        self.load_source()
        self.source_fk_links = profile.source_fk_links
```

```
        self.update_source_links_table()
        self.import_links = []
        for link in profile.import_links:
            target_entity_id = None
            for ent in self.mozart_entities:
                if ent['id'] == link['target_entity_id']:
                    target_entity_id = ent['id']
                    break
            if not target_entity_id:
                continue
            for mapping in link['mappings']:
                self.import_links.append({
                    'source_table': link['source_table'],
                    'source_field': mapping['source_field'],
                    'target_entity_id': target_entity_id,
                    'target_field': mapping['target_field'],
                    'type': 'sourceId' if mapping.get('is_source_id') else 'import'
                })
        self.update_import_links_table()
        QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('profile_loaded'))
    def start_splitter_drag(self, event):
        self.dragging = True
        self.drag_start_y = event.globalY()
        self.start_height = self.links_table.height()
        self.setCursor(Qt.SizeVerCursor)
    def drag_splitter(self, event):
        if self.dragging:
            delta = event.globalY() - self.drag_start_y
            new_height = self.start_height + delta
            new_height = max(100, min(600, new_height))
            self.links_table.setMinimumHeight(new_height)
            self.links_table.setMaximumHeight(new_height)
    def stop_splitter_drag(self, event):
        self.dragging = False
        self.setCursor(Qt.ArrowCursor)
    def retranslate_ui(self):
        self.setWindowTitle(self.translator.tr('import_designer_title'))
        self.source_tree.setHeaderLabels([self.translator.tr('source_tables'), self.translator.tr('rows')])
        self.mozart_tree.setHeaderLabels([self.translator.tr('mozart_entities'), ""])
        self.source_links_table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_table'),
            self.translator.tr('target_field'),
        ])
        self.links_table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_entity'),
            self.translator.tr('target_field'),
            self.translator.tr('type'),
        ])
        self.btn_create_entity.setText(self.translator.tr('create_entity'))
        self.btn_add_link.setText(self.translator.tr('add_import_link'))
        self.btn_save.setText(self.translator.tr('save_profile'))
        self.btn_import.setText(self.translator.tr('start_import'))
    def load_source(self):
        self.source_tree.clear()
        schema = self.connector.get_schema()
        self.source_tables = schema.get_tables
        for table in self.source_tables:
            row_count = self.connector.get_row_count(table.name)
            count_str = str(row_count) if row_count is not None else '?'
            item = QTreeWidgetItem([table.name, count_str])
            item.setData(0, Qt.UserRole, table.name)
            for field in table.fields:
                field_item = QTreeWidgetItem([f"  {field.name} ({field.field_type.name})", ""])
                field_item.setData(0, Qt.UserRole, field.name)
                field_item.setFlags(field_item.flags() | Qt.ItemIsUserCheckable)
                field_item.setCheckState(0, Qt.Unchecked)
                item.addChild(field_item)
            self.source_tree.addTopLevelItem(item)
        self.detect_source_fk_links()
        self.update_source_links_table()
    def detect_source_fk_links(self):
        self.source_fk_links = []
        for table in self.source_tables:
            for field in table.fields:
                if field.is_foreign_key and field.refers_to:
                    self.source_fk_links.append({
                        'source_table': table.name,
                        'source_field': field.name,
                        'target_table': field.refers_to,
                        'target_field': 'id'
                    })
    def update_source_links_table(self):
        self.source_links_table.setRowCount(len(self.source_fk_links))
        for row, link in enumerate(self.source_fk_links):
            self.source_links_table.setItem(row, 0, QTableWidgetItem(link['source_table']))
            self.source_links_table.setItem(row, 1, QTableWidgetItem(link['source_field']))
            self.source_links_table.setItem(row, 2, QTableWidgetItem(link['target_table']))
            self.source_links_table.setItem(row, 3, QTableWidgetItem(link['target_field']))
    def load_mozart(self):
        self.mozart_tree.clear()
rows = self.db.execute_query("SELECT id, cname, calias, class_id FROM meta.entitytypes ORDER BY cname")
        self.mozart_entities = []
        for eid, cname, calias, class_id in rows:
            class_res = self.db.execute_query("SELECT calias FROM meta.entity_classes WHERE id=%s", (class_id,))
class_alias = class_res[0][0] if class_res else "DICT"
fields = self.db.execute_query("SELECT cfieldname, cfieldtype FROM meta.fields WHERE entitytypeid=%s",
(eid,))
            item = QTreeWidgetItem([f"{cname} ({calias} - {class_alias})", ""])
            item.setData(0, Qt.UserRole, eid)
            for fname, ftype in fields:
                fitem = QTreeWidgetItem([f"  {fname} ({ftype})", ""])
                fitem.setData(0, Qt.UserRole, name)
                item.addChild(fitem)
            self.mozart_tree.addTopLevelItem(item)
            self.mozart_entities.append({
                'id': eid, 'cname': cname, 'calias': calias, 'class_alias': class_alias, 'fields': [f[0] for f in fields]
            })
    def source_context_menu(self, pos):
        item = self.source_tree.itemAt(pos)
        if not item:
            return
        menu = QMenu()
        if item.parent() is None:
            preview_action = QAction(self.translator.tr('preview_table'), None)
            preview_action.triggered.connect(lambda: self.preview_table(item))
            menu.addAction(preview_action)
            add_link_action = QAction(self.translator.tr('add_source_link'), None)
            add_link_action.triggered.connect(lambda: self.add_source_link(item))
            menu.addAction(add_link_action)
            select_all_action = QAction(self.translator.tr('select_all_fields'), None)
            select_all_action.triggered.connect(lambda: self.set_checks_for_table(item, Qt.Checked))
            menu.addAction(select_all_action)
            deselect_all_action = QAction(self.translator.tr('deselect_all_fields'), None)
            deselect_all_action.triggered.connect(lambda: self.set_checks_for_table(item, Qt.Unchecked))
            menu.addAction(deselect_all_action)
        menu.exec_(self.source_tree.mapToGlobal(pos))
    def set_checks_for_table(self, table_item, state):
        for i in range(table_item.childCount()):
            field_item = table_item.child(i)
            field_item.setCheckState(0, state)
    def add_source_link(self, table_item):
        table_name = table_item.data(0, Qt.UserRole)
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('add_source_link_title'))
        layout = QVBoxLayout(dlg)
        form = QFormLayout()
        source_field_combo = QComboBox()
        for t in self.source_tables:
            if t.name == table_name:
                for f in t.fields:
                    source_field_combo.addItem(f.name, f.name)
                break
        form.addRow(self.translator.tr('source_field'), source_field_combo)
        target_table_combo = QComboBox()
```

```
        for t in self.source_tables:
            if t.name == table_name:
                target_table_combo.addItem(t.name, t.name)
        form.addRow(self.translator.tr('target_table'), target_table_combo)
        target_field_combo = QComboBox()
target_table_combo.currentIndexChanged.connect(lambda: self.update_target_fields(target_table_combo,
target_field_combo))
        self.update_target_fields(target_table_combo, target_field_combo)
        form.addRow(self.translator.tr('target_field'), target_field_combo)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(dlg.accept)
        btn_box.rejected.connect(dlg.reject)
        layout.addWidget(btn_box)
        if dlg.exec_():
            self.source_fk_links.append({
                'source_table': table_name,
                'source_field': source_field_combo.currentData(),
                'target_table': target_table_combo.currentData(),
                'target_field': target_field_combo.currentData()
            })
            self.update_source_links_table()
    def update_target_fields(self, target_table_combo, target_field_combo):
        target_table = target_table_combo.currentData()
        target_field_combo.clear()
        for t in self.source_tables:
            if t.name == target_table:
                for f in t.fields:
                    target_field_combo.addItem(f.name, f.name)
                    break
    def mozart_context_menu(self, pos):
        item = self.mozart_tree.itemAt(pos)
        if not item:
            return
        menu = QMenu()
        if item.parent() is None:
            create_action = QAction(self.translator.tr('create_fields_from_checked'), None)
            create_action.triggered.connect(lambda: self.create_fields_from_checked(item))
            menu.addAction(create_action)
        menu.exec_(self.mozart_tree.mapToGlobal(pos))
    def create_fields_from_checked(self, target_entity_item):
        entity_id = target_entity_item.data(0, Qt.UserRole)
        checked_fields = []
        root = self.source_tree.invisibleRootItem()
        for i in range(root.childCount()):
            table_item = root.child(i)
            for j in range(table_item.childCount()):
                field_item = table_item.child(j)
                if field_item.checkState(0) == Qt.Checked:
                    checked_fields.append({
                        'source_field_name': field_item.data(0, Qt.UserRole),
                        'source_table': table_item.data(0, Qt.UserRole)
                    })
        if not checked_fields:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('no_fields_selected'))
            return
        dlg = MassFieldDialog(self, checked_fields, self.translator)
        if dlg.exec_():
            fields_to_create = dlg.get_fields()
            if not fields_to_create:
                return
            creator = FieldCreator(self.db)
            for f in fields_to_create:
                creator.create_fields(entity_id, [{
                    'fieldname': f['fieldname'][:63],
                    'fieldtype': f['fieldtype'],
                    'calias': f['calias'][:50],
                    'nlength': f['nlength']
                }])
            self.load_mozart()
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('fields_created'))
    def preview_table(self, table_item):
        table_name = table_item.data(0, Qt.UserRole)
        sample = self.connector.get_table_sample(table_name, limit=20)
        if not sample:
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('no_data'))
            return
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('preview_table_title').format(table_name))
        layout = QVBoxLayout(dlg)
        table = QTableWidget()
        headers = list(sample[0].keys())
        table.setColumnCount(len(headers))
        table.setHorizontalHeaderLabels(headers)
        table.setRowCount(len(sample))
        for r, row in enumerate(sample):
            for c, col in enumerate(headers):
                table.setItem(r, c, QTableWidgetItem(str(row.get(col, ''))))
        layout.addWidget(table)
        dlg.exec_()
    def create_entity(self):
        from wizards.wizard_dialogs import NewEntityDialog
        dlg = NewEntityDialog(self, self.db)
        if dlg.exec_():
            data = dlg.get_data()
            res = self.db.execute_query(
"""INSERT INTO entitytypes (cname, calias, class_id, chaseclass, lisversioned, isecuritystrategy, status_id)
                VALUES (%s, %s, %s, true, 1, 1) RETURNING id""",
                (data['cname'], data['calias'], data['class_id'])
            )
            if res:
                entity_id = res[0][0]
                self.load_mozart()
                QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('entity_created'))
    def add_import_link(self):
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('add_import_link_title'))
        layout = QVBoxLayout(dlg)
        form = QFormLayout()
        src_table_combo = QComboBox()
        for t in self.source_tables:
            src_table_combo.addItem(t.name, t.name)
        form.addRow(self.translator.tr('source_table'), src_table_combo)
        src_field_combo = QComboBox()
src_table_combo.currentIndexChanged.connect(lambda: self.update_src_fields(src_table_combo,
src_field_combo))
        self.update_src_fields(src_table_combo, src_field_combo)
        form.addRow(self.translator.tr('source_field'), src_field_combo)
        target_entity_combo = QComboBox()
        for ent in self.mozart_entities:
            target_entity_combo.addItem(f"{ent['cname']} ({ent['calias']})", ent['id'])
        form.addRow(self.translator.tr('target_entity'), target_entity_combo)
        target_field_combo = QComboBox()
target_entity_combo.currentIndexChanged.connect(lambda:
self.update_target_fields(target_entity_combo, target_field_combo))
        self.update_target_fields(target_entity_combo, target_field_combo)
        form.addRow(self.translator.tr('target_field'), target_field_combo)
        type_combo = QComboBox()
        type_combo.addItem(self.translator.tr('import_type'), 'import')
        type_combo.addItem(self.translator.tr('sourceid_type'), 'sourceid')
        form.addRow(self.translator.tr('type'), type_combo)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(dlg.accept)
        btn_box.rejected.connect(dlg.reject)
        layout.addWidget(btn_box)
        if dlg.exec_():
            target_field = target_field_combo.currentData()
            self.import_links.append({
                'source_table': src_table_combo.currentData(),
                'source_field': src_field_combo.currentData(),
                'target_entity_id': target_entity_combo.currentData(),
                'target_field': target_field,
                'type': type_combo.currentData()
            })
            self.update_import_links_table()
    def update_src_fields(self, src_table_combo, src_field_combo):
        table_name = src_table_combo.currentData()
        src_field_combo.clear()
        for t in self.source_tables:
            if t.name == table_name:
                for f in t.fields:
                    src_field_combo.addItem(f.name, f.name)
                    break
```

```
def update_target_fields_entity(self, target_entity_combo, target_field_combo):
    entity_id = target_entity_combo.currentData()
    target_field_combo.clear()
    for ent in self.mozart_entities:
        if ent['id'] == entity_id:
            for fname in ent['fields']:
                target_field_combo.addItem(fname, fname)
            break
def update_import_links_table(self):
    self.links_table.setRowCount(len(self.import_links))
    for row, link in enumerate(self.import_links):
        self.links_table.setItem(row, 0, QTableWidgetItem(link['source_table']))
        self.links_table.setItem(row, 1, QTableWidgetItem(link['source_field']))
        entity_name = ''
        for ent in self.mozart_entities:
            if ent['id'] == link['target_entity_id']:
                entity_name = ent['cname']
                break
        self.links_table.setItem(row, 2, QTableWidgetItem(entity_name))
        self.links_table.setItem(row, 3, QTableWidgetItem(link['target_field'] or ''))
        self.links_table.setItem(row, 4, QTableWidgetItem(link['type']))
        btn = QPushButton(self.translator.tr('delete'))
        btn.clicked.connect(lambda checked, r=row: self.remove_import_link(r))
        self.links_table.setCellWidget(row, 5, btn)
def remove_import_link(self, row):
    self.import_links.pop(row)
    self.update_import_links_table()
def save_profile(self):
    source_tables = []
    for table in self.source_tables:
        sample_rows = []
        if hasattr(table, 'sample_rows') and table.sample_rows:
            sample_rows = table.sample_rows[:5]
        source_tables.append({
            'name': table.name,
            'row_count': self.connector.get_row_count(table.name),
            'sample_rows': sample_rows
        })
    target_entities = []
    seen = set()
    for link in self.import_links:
        eid = link['target_entity_id']
        if eid not in seen:
            for ent in self.mozart_entities:
                if ent['id'] == eid:
                    target_entities.append({'id': eid, 'name': ent['cname']})
                    seen.add(eid)
                    break
    import_links = []
    for link in self.import_links:
        mappings = {
            'source_field': link['source_field'],
            'target_field': link['target_field'],
            'field_type': 'C',
            'field_length': 255,
            'is_source_id': (link['type'] == 'sourceid')
        }
        import_links.append({
            'source_table': link['source_table'],
            'target_entity_id': link['target_entity_id'],
            'mappings': mappings
        })
    profile = ImportProfile(
        name=self.connector.source_path,
        source_type=SourceType.FILE,
        source_path=self.connector.source_path,
        source_tables=source_tables,
        target_entities=target_entities,
        import_links=import_links,
        source_fk_links=self.source_fk_links
    )
    if self.editing_profile_id:
        profile.id = self.editing_profile_id
    prof_mgr = ProfileManager(self.db)
    prof_mgr.save_profile(profile)
    QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('profile_saved'))
def perform_import(self):
    creator = FieldCreator(self.db)
    for link in self.import_links:
        if link['type'] == 'import' and link['target_field']:
            check = self.db.execute_query(
                "SELECT id FROM meta_fields WHERE entitytypeid=%s AND cfieldname=%s",
                (link['target_entity_id'], link['target_field'])
            )
            if not check:
                creator.create_fields(link['target_entity_id'], [{
                    'cfieldname': link['target_field'][:63],
                    'cfieldtype': 'C',
                    'calias': link['target_field'].capitalize()[:50],
                    'nlength': 255
                }])
        elif link['type'] == 'sourceid':
            check = self.db.execute_query(
                "SELECT id FROM meta_fields WHERE entitytypeid=%s AND cfieldname='sourceid'",
                (link['target_entity_id'],)
            )
            if not check:
                creator.create_fields(link['target_entity_id'], [{
                    'cfieldname': 'sourceid',
                    'cfieldtype': 'C',
                    'calias': 'Source ID',
                    'nlength': 100
                }])
    groups = {}
    for link in self.import_links:
        eid = link['target_entity_id']
        if eid not in groups:
            groups[eid] = []
        groups[eid].append(link)
    entities = []
    for eid, links in groups.items():
        target_entity = None
        for ent in self.mozart_entities:
            if ent['id'] == eid:
                target_entity = ent
                break
        if not target_entity:
            continue
        source_table_name = links[0]['source_table']
        matched_fields = []
        for link in links:
            if link['type'] == 'import' and link['target_field']:
                matched_fields.append(MatchedField(
                    source_field=link['source_field'],
                    target_field=link['target_field']
                ))
        entities.append(MatchedEntity(
            source_table=source_table_name,
            target_entity_id=eid,
            target_entity_name=target_entity['cname'],
            confidence=1.0,
            fields=matched_fields,
            is_new=False
        ))
    mapping_proposal = MappingProposal(
        source_schema=self.connector.get_schema(),
        entities=entities,
        unresolved_tables=[],
        suggestions_for_new_entities=[]
    )
    profile = ImportProfile(
        name=self.connector.source_path,
        source_type=SourceType.FILE,
        source_path=self.connector.source_path,
        mapping_proposal=mapping_proposal
    )
    progress = QProgressDialog(self.translator.tr('import_progress'), self.translator.tr('cancel'), 0, 0, self)
    progress.setWindowModality(Qt.WindowModal)
    try:
        engine = ImportEngine(self.db, self.connector, profile)
        stats = engine.run()
        progress.close()
        QMessageBox.information(self, self.translator.tr('import_finished_title'),
```

```
self.translator.tr('import_finished_stats').format(stats['inserted'], stats['updated'], stats['errors']))
    except Exception as e:
        traceback.print_exc()
        QMessageBox.critical(self, self.translator.tr('error'), str(e))
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/link_edit_dialog.py (от

```
# mozart_import/designer/link_edit_dialog.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QTableWidget, QTableWidgetItem,
                             QComboBox, QCheckBox, QDialogButtonBox, QHeaderView,
                             QMessageBox)
from PyQt5.QtCore import Qt
from lang.local_translator import LocalTranslator

class LinkEditDialog(QDialog):
    """
    Диалог настройки связи импорта.
    source_fields – список имен полей источника, уже отфильтрованных пользователем (чекбоксы).
    target_fields – полный список полей Mozart (включая sourceid).
    source_refs – словарь ссылок {field_name: {'is_ref': bool, 'ref_table': str, 'ref_field': str}}
    """
    def __init__(self, source_fields: list, target_fields: list, current_mappings=None, source_refs=None,
                 parent=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('mapping_details_title'))
        self.resize(800, 400)
        self.source_fields = source_fields
        self.target_fields = target_fields
        self.current_mappings = current_mappings or []
        self.source_refs = source_refs or {}
        self.mappings = []
        layout = QVBoxLayout(self)
        self.table = QTableWidget(len(source_fields), 4)
        self.table.setHorizontalHeaderLabels([
            self.translator.tr('source_field'),
            self.translator.tr('target_field'),
            self.translator.tr('sourceid_type'),
            "Ссылка"
        ])
        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.table.verticalHeader().setVisible(False)
        for row, src_field in enumerate(source_fields):
            # Поле источника (только чтение)
            item = QTableWidgetItem(src_field)
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(row, 0, item)
            # Выпадающий список полей Mozart
            combo = QComboBox()
            combo.addItem(self.translator.tr('ignore_field'), None)  # ключ None = игнорировать
            for tgt in target_fields:
                combo.addItem(tgt, tgt)
            self.table.setCellWidget(row, 1, combo)
            # Чекбокс ключевого поля (sourceid)
            chk_key = QCheckBox()
            chk_key.stateChanged.connect(lambda state, r=row: self._on_source_id_changed(r, state))
            self.table.setCellWidget(row, 2, chk_key)
            # Чекбокс "ссылка" (информативный)
            chk_ref = QCheckBox()
            ref_info = self.source_refs.get(src_field, {})
            if ref_info.get('is_ref'):
                chk_ref.setChecked(True)
                chk_ref.setText(f"{ref_info.get('ref_table','')}.{ref_info.get('ref_field','')}")
                chk_ref.setEnabled(True)
            else:
                chk_ref.setChecked(False)
                chk_ref.setEnabled(False)
                chk_ref.setText('')
            self.table.setCellWidget(row, 3, chk_ref)
        # Восстанавливаем сохранённые маппинги (регистронезависимо)
        for mapping in self.current_mappings:
            src = mapping.get('source_field')
            if src in self.source_fields:
                row = self.source_fields.index(src)
                tgt = mapping.get('target_figld')
                if tgt:
                    # Регистронезависимый поиск
                    combo = self.table.cellWidget(row, 1)
                    idx = -1
                    for i in range(combo.count()):
                        if combo.itemData(i) and combo.itemData(i).lower() == tgt.lower():
                            idx = i
                            break
                    if idx >= 0:
                        combo.setCurrentIndex(idx)
                if mapping.get('is_source_id'):
                    self.table.cellWidget(row, 2).setChecked(True)
        layout.addWidget(self.table)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self._on_accept)
        btn_box.rejected.connect(self.reject)
        layout.addWidget(btn_box)
    def _on_source_id_changed(self, changed_row, state):
        if state == Qt.Checked:
            for row in range(self.table.rowCount()):
                if row != changed_row:
                    self.table.cellWidget(row, 2).setChecked(False)
    def _on_accept(self):
        has_mapping = False
        source_id_row = None
        for row in range(self.table.rowCount()):
            combo = self.table.cellWidget(row, 1)
            tgt = combo.currentData()
            if tgt:
                has_mapping = True
                if self.table.cellWidget(row, 2).isChecked():
                    source_id_row = row
        if not has_mapping:
            QMessageBox.warning(self, self.translator.tr('warning'),
                                "Не выбрано ни одного сопоставления полей.")
            return
        if source_id_row is None:
            QMessageBox.warning(self, self.translator.tr('warning'),
                                "Не указано ключевое поле источника (sourceid).")
            return
        self.mappings = []
        for row in range(self.table.rowCount()):
            combo = self.table.cellWidget(row, 1)
            tgt = combo.currentData()
            if tgt:
                src_field = self.source_fields[row]
                ref_info = self.source_refs.get(src_field, {})
                mapping = {
                    'source_field': src_field,
                    'target_field': tgt,
                    'is_source_id': self.table.cellWidget(row, 2).isChecked(),
                    'is_reference': ref_info.get('is_ref', False),
                    'ref_source_table': ref_info.get('ref_table', ''),
                    'ref_source_key_field': ref_info.get('ref_field', '')
                }
                self.mappings.append(mapping)
        self.accept()
    def get_mappings(self):
        return self.mappings
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/scene_profile_adapter

```
# -*- coding: utf-8 -*-
```

```
from typing import Optional
from database import DatabaseService
from ..core.models import ImportProfile, SourceType, FileFormat, NodeType, LinkType

class SceneProfileAdapter:
    @staticmethod
    def get_entity_name(db: DatabaseService, entity_id: int) -> str:
        """Возвращает cname сущности по ее ID, или пустую строку, если не найдено."""
        if not entity_id:
            return ''
        rows = db.execute_query("SELECT cname FROM meta.entitytypes WHERE id = %s", (entity_id,))
        return rows[0][0] if rows else ''
    @staticmethod
    def scene_to_profile(scene, connector, db: DatabaseService, editing_profile_id: Optional[int] = None) ->
ImportProfile:
        """Преобразует сцену в объект ImportProfile с гарантированным заполнением имени целевой
сущности."""
        # Параметры подключения
        src_type = SourceType.FILE
        file_format = None
        db_type = None
        db_host = None
        db_port = None
        db_name = None
        db_user = None
        db_pass = None
        if hasattr(connector, 'file_format'):
            src_type = SourceType.FILE
            file_format = connector.file_format.value if isinstance(connector.file_format, FileFormat) else
str(connector.file_format)
        elif hasattr(connector, 'db_type'):
            src_type = SourceType.DATABASE
            db_type = connector.db_type
            db_host = connector.host
            db_port = int(connector.port) if connector.port else None
            db_name = connector.database
            db_user = connector.user
            db_pass = connector.password
        else:
            src_type = SourceType.URL
        # Таблицы источника
        source_tables = []
        for node_item in scene.nodes.values():
            if node_item.node.node_type != NodeType.SOURCE_TABLE:
                continue
            row_count = None
            try:
                row_count = connector.get_row_count(node_item.node.table_name)
            except Exception:
                pass
            sample_rows = []
            try:
                sample = connector.get_table_sample(node_item.node.table_name, limit=5)
                sample_rows = sample
            except Exception:
                pass
            source_tables.append({
                'name': node_item.node.table_name,
                'row_count': row_count,
                'sample_rows': sample_rows
            })
        # Группируем связи импорта по (source_table, target_entity_id)
        links_by_target = {}
        for link in scene.links:
            if link.link_type != LinkType.IMPORT:
                continue
            start_node = link.start_item.node
            end_node = link.end_item.node
            if end_node.entity_id is None:
                continue
            refs = getattr(start_node, 'refs', {})
            key = (start_node.table_name, end_node.entity_id)
            if key not in links_by_target:
                # Получаем имя сущности: сначала из end_node.name, если пусто – из БД
                entity_name = end_node.name
                if not entity_name and end_node.entity_id:
                    entity_name = SceneProfileAdapter.get_entity_name(db, end_node.entity_id)
                # Если все равно пусто, используем имя из узла или заглушку
                if not entity_name:
                    entity_name = end_node.name or f"Entity_{end_node.entity_id}"
                links_by_target[key] = {
                    'source_table': start_node.table_name,
                    'target_entity_id': end_node.entity_id,
                    'target_entity_name': entity_name,
                    'mappings': []
                }
            # Обрабатываем маппинги линии (принудительно берём из link.mappings)
            if link.mappings:
                for m in link.mappings:
                    src_field = m.get('source_field')
                    ref_info = refs.get(src_field, {})
                    enriched = {
                        'source_field': src_field,
                        'target_field': m.get('target_field'),
                        'is_source_id': m.get('is_source_id', False),
                        'is_reference': ref_info.get('is_ref', False),
                        'ref_source_table': ref_info.get('ref_table', ''),
                        'ref_source_key_field': ref_info.get('ref_field', '')
                    }
                    # Защита от дубликатов
                    if not any(ex['source_field'] == enriched['source_field'] and ex['target_field'] == enriched['target_field']
                               for ex in links_by_target[key]['mappings']):
                        links_by_target[key]['mappings'].append(enriched)
            else:
                # Одиночный маппинг
                src_field = link.start_field_item.field.name
                tgt_field = link.end_field_item.field.name
                ref_info = refs.get(src_field, {})
                mapping = {
                    'source_field': src_field,
                    'target_field': tgt_field,
                    'is_source_id': tgt_field == 'sourceid',
                    'is_reference': ref_info.get('is_ref', False),
                    'ref_source_table': ref_info.get('ref_table', ''),
                    'ref_source_key_field': ref_info.get('ref_field', '')
                }
                if not any(ex['source_field'] == mapping['source_field'] and ex['target_field'] == mapping['target_field']
                           for ex in links_by_target[key]['mappings']):
                    links_by_target[key]['mappings'].append(mapping)
        # Автоматическое добавление ключевой пары (sourceid) если отсутствует
        for key, entry in links_by_target.items():
            source_id_mapping = next((m for m in entry['mappings'] if m.get('is_source_id')), None)
            if source_id_mapping:
                continue
            source_table_name = entry['source_table']
            source_node = None
            for ni in scene.nodes.values():
                if (ni.node.node_type == NodeType.SOURCE_TABLE and
                    ni.node.table_name == source_table_name):
                    source_node = ni
                    break
            key_field = None
            refs_of_node = getattr(source_node.node, 'refs', {}) if source_node else {}
            for fname, info in refs_of_node.items():
                if not info.get('is_ref'):
                    key_field = fname
                    break
            if not key_field and source_node:
                key_field = source_node.node.fields[0].name if source_node.node.fields else None
            if key_field:
                entry['mappings'].append({
                    'source_field': key_field,
                    'target_field': 'sourceid',
                    'is_source_id': True,
                    'is_reference': False,
                    'ref_source_table': '',
                    'ref_source_key_field': ''
                })
```

```
        target_entities = []
        import_links = []
        for key, entry in links_by_target_items():
            target_entities.append({'id': entry['target_entity_id'], 'name': entry['target_entity_name']})
            import_links.append({
                'source_table': entry['source_table'],
                'target_entity_id': entry['target_entity_id'],
                'mappings': entry['mappings']
            })
        # FK связи источника
        source_fk_links = []
        for link in scene.links:
            if link.link_type == LinkType.FK_SOURCE:
                source_fk_links.append({
                    'source_table': link.start_item.node.table_name,
                    'source_field': link.start_field_item.field.name,
                    'target_table': link.end_item.node.table_name,
                    'target_field': link.end_field_item.field.name
                })
        # Сбор ручных настроек ссылок источника (source_references)
        source_references = []
        for node_item in scene.nodes.values():
            if node_item.node.node_type != NodeType.SOURCE_TABLE:
                continue
            refs = getattr(node_item.node, 'refs', {})
            for field_name, ref_info in refs.items():
                if ref_info.get('is_ref'):
                    source_references.append({
                        'source_table': node_item.node.table_name,
                        'source_field': field_name,
                        'ref_table': ref_info['ref_table'],
                        'ref_field': ref_info['ref_field']
                    })
        return ImportProfile(
            id=editing_profile_id,
            name=connector.source_path,
            source_type=src_type,
            source_path=connector.source_path,
            source_tables=source_tables,
            target_entities=target_entities,
            import_links=import_links,
            source_fk_links=source_fk_links,
            source_references=source_references,
            file_format=file_format,
            db_type=db_type,
            db_host=db_host,
            db_port=db_port,
            db_name=db_name,
            db_user=db_user,
            db_pass=db_pass
        )
    @staticmethod
    def restore_scene(scene, profile: ImportProfile, mozart_entities=None):
        """ Восстанавливает связи импорта и внутренние ссылки источника на сцене """
        # Восстанавливаем refs для всех узлов источника из profile.source_references
        for ref in profile.source_references:
            for nitem in scene.nodes.values():
                if (nitem.node.node_type == NodeType.SOURCE_TABLE and
                    nitem.node.table_name == ref['source_table']):
                    refs = getattr(nitem.node, 'refs', {})
                    refs[ref['source_field']] = {
                        'is_ref': True,
                        'ref_table': ref['ref_table'],
                        'ref_field': ref['ref_field']
                    }
                    nitem.node.refs = refs
                    break
        # Восстанавливаем связи импорта
        for link_data in profile.import_links:
            source_table = link_data['source_table']
            target_entity_id = link_data['target_entity_id']
            source_node = None
            target_node = None
            for nitem in scene.nodes.values():
                if nitem.node.node_type == NodeType.SOURCE_TABLE and nitem.node.table_name == source_table:
                    source_node = nitem
elif nitem.node.node_type == NodeType.MOZART_ENTITY and nitem.node.entity_id == target_entity_id:
                    target_node = nitem
            if not source_node or not target_node:
                continue
            mappings = link_data.get('mappings', [])
            # Удаляем старые линии между этими узлами
            for link in scene.links[:]:
if link.link_type == LinkType.IMPORT and link.start_item == source_node and link.end_item == target_node:
                    scene.removeItem(link)
                    scene.links.remove(link)
            # Создаём новую линию и принудительно записываем маппинги
            if mappings:
                link = scene.add_link(source_node, target_node, LinkType.IMPORT, mappings=mappings)
                if link:
                    link.mappings = mappings
            else:
                link = scene.add_link(source_node, target_node, LinkType.IMPORT)
        scene.update_all_links()
```

Файл: /home/sergey/Documents/configurate/mozart_import/designer/source_ref_dialog.py (

```
# mozart_import/designer/source_ref_dialog.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QTableWidget, QTableWidgetItem,
                             QCheckBox, QComboBox, QDialogButtonBox, QHeaderView, QMessageBox)
from PyQt5.QtCore import Qt

class SourceRefDialog(QDialog):
    """
    Диалог настройки ссылочных полей внутри таблицы источника.
    Для каждого поля можно указать, является ли оно ссылкой на другую таблицу источника,
    выбрать целевую таблицу и поле, которое служит ключом в этой таблице.
    """
    def __init__(self, source_table_name: str, source_fields: list, all_source_table_names: list,
                 current_refs: dict = None, parent=None):
        source_fields -- список имён полей текущей таблицы
        all_source_table_names -- список имен всех таблиц источника (для выбора целевой таблицы)
        current_refs -- словарь {имя_поля: {'is_ref': bool, 'ref_table': str, 'ref_field': str}}
        super().__init__(parent)
        self.setWindowTitle("Ссылочные поля таблицы «{source_table_name}»")
        self.resize(800, 400)
        self.source_fields = source_fields
        self.all_tables = all_source_table_names
        self.current_refs = current_refs or {}
        self.result_refs = {}
        layout = QVBoxLayout(self)
        self.table = QTableWidget(len(source_fields), 3)
        self.table.setHorizontalHeaderLabels(["Поле", "Ссылка?", "Целевая таблица.поле"])
        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.table.verticalHeader().setVisible(False)
        for row, field_name in enumerate(source_fields):
            # Имя поля
            item = QTableWidgetItem(field_name)
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(row, 0, item)
            # Чекбокс "Ссылка"
            chk = QCheckBox()
            ref_info = self.current_refs.get(field_name, {})
            chk.setChecked(ref_info.get('is_ref', False))
            self.table.setCellWidget(row, 1, chk)
            # Комбобокс с целевой таблицей и полем
            combo = QComboBox()
            for tbl in all_source_table_names:
```

```
# Добавляем поля этой таблицы (упрощённо: предполагаем, что ключевое поле называется 'id' или первое поле)
# В реальном приложении нужно знать схему всех таблиц, но здесь используем заглушку: поле 'id'
        combo.addItem("id", "id")  # Пока предполагаем, что ключевое поле всегда 'id'
        # Если текущая ссылка уже установлена, выбираем соответствующий пункт
        if ref_info.get('is_ref'):
            target = ref_info.get('ref_table')
            field = ref_info.get('ref_field')
            idx = combo.findData((target, field))
            if idx >= 0:
                combo.setCurrentIndex(idx)
        self.table.setCellWidget(row, 2, combo)
    layout.addWidget(self.table)
    btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
    btn_box.accepted.connect(self._save)
    btn_box.rejected.connect(self.reject)
    layout.addWidget(btn_box)
def _save(self):
    self.result_refs = {}
    for row in range(self.table.rowCount()):
        field_name = self.table.item(row, 0).text()
        is_ref = self.table.cellWidget(row, 1).isChecked()
        combo = self.table.cellWidget(row, 2)
        target_table, target_field = combo.currentData()
        self.result_refs[field_name] = {
            'is_ref': is_ref,
            'ref_table': target_table,
            'ref_field': target_field
        }
    self.accept()
def get_refs(self):
    return self.result_refs
```

Файл: /home/sergey/Documents/configurate/mozart_import/dialogs/profile_manager_dialog

```
# mozart_import/dialogs/profile_manager_dialog.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout, QTableWidget,
                             QTableWidgetItem, QPushButton, QMessageBox, QHeaderView,
                             QDialogBox, QGroupBox, QLabel, QLineEdit, QFileDialog,
                             QDialogButtonBox)
from PyQt5.QtCore import Qt
from database.import_DatabaseService
from lang.local_translator import LocalTranslator
from ..core.profile_manager import ProfileManager
from ..designer.designer_v2_import_import_designer_v2
from ..designer.scene_profile_adapter.import_ScenesProfileAdapter
from ..connectors.file_connector.import_FileConnector
from ..connectors.db_connector.import_DBConnector
from ..connectors.url_connector.import_URLConnector
from ..core.models import SourceType, FileFormat

class ProfileManagerDialog(QDialog):
    def __init__(self, db, parent=None):
        super().__init__(parent)
        self.db = db
        self.translator = LocalTranslator()
        self.prof_manager = ProfileManager(self.db)
        self.setWindowTitle(self.translator.tr('profile_manager_title'))
        self.resize(800, 500)
        layout = QVBoxLayout(self)
        self.table = QTableWidget()
        self.table.setColumnCount(4)
        self.table.setHorizontalHeaderLabels([
            self.translator.tr('profile_name'),
            self.translator.tr('source_type'),
            self.translator.tr('created_at'),
        ])
        self.table.horizontalHeader().setStretchLastSection(True)
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        self.table.setSelectionMode(QTableWidget.SingleSelection)
        layout.addWidget(self.table)
        btn_layout = QHBoxLayout()
        self.btn_add = QPushButton(self.translator.tr('profile_add'))
        self.btn_add.clicked.connect(self.add_profile)
        self.btn_edit = QPushButton(self.translator.tr('profile_edit'))
        self.btn_edit.clicked.connect(self.edit_profile)
        self.btn_delete = QPushButton(self.translator.tr('profile_delete'))
        self.btn_delete.clicked.connect(self.delete_profile)
        self.btn_close = QPushButton(self.translator.tr('close'))
        self.btn_close.clicked.connect(self.accept)
        btn_layout.addWidget(self.btn_add)
        btn_layout.addWidget(self.btn_edit)
        btn_layout.addWidget(self.btn_delete)
        btn_layout.addStretch()
        btn_layout.addWidget(self.btn_close)
        layout.addLayout(btn_layout)
        self.load_profiles()
    def load_profiles(self):
        profiles = self.prof_manager.list_profiles()
        self.table.setRowCount(len(profiles))
        for row, prof in enumerate(profiles):
            self.table.setItem(row, 0, QTableWidgetItem(str(prof.id)))
            self.table.setItem(row, 1, QTableWidgetItem(prof.name))
            self.table.setItem(row, 2, QTableWidgetItem(prof.source_type.value if prof.source_type else ''))
            self.table.setItem(row, 3, QTableWidgetItem(str(prof.created_at) if hasattr(prof, 'created_at') else ''))
            self.table.item(row, 0).setData(Qt.UserRole, prof.id)
    def get_selected_profile_id(self):
        current = self.table.currentRow()
        if current >= 0:
            return self.table.item(current, 0).data(Qt.UserRole)
        return None
    def add_profile(self):
        self.run_import_designer()
    def edit_profile(self):
        profile_id = self.get_selected_profile_id()
        if not profile_id:
            QMessageBox.warning(self, self.translator.tr('warning'),
                                self.translator.tr('select_profile_first'))
            return
        profile = self.prof_manager.load_profile(profile_id)
        if not profile:
            QMessageBox.critical(self, self.translator.tr('error'),
                                 self.translator.tr('profile_load_error'))
            return
        self.run_import_designer(existing_profile=profile)
    def delete_profile(self):
        profile_id = self.get_selected_profile_id()
        if not profile_id:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('select_profile_first'))
            return
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('delete_profile_confirm')) == QMessageBox.Yes:
            self.prof_manager.delete_profile(profile_id)
            self.load_profiles()
    def run_import_designer(self, existing_profile=None):
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('select_source_dialog_title'))
        dlg.resize(500, 400)
        layout = QVBoxLayout(dlg)
        type_combo = QComboBox()
        type_combo.addItem(self.translator.tr('source_type_file'), SourceType.FILE)
        type_combo.addItem(self.translator.tr('source_type_db'), SourceType.DATABASE)
        type_combo.addItem(self.translator.tr('source_type_url'), SourceType.URL)
        layout.addWidget(type_combo)
        # Группа для файла
        file_group = QGroupBox(self.translator.tr('file_group'))
        file_layout = QVBoxLayout(file_group)
```

```
format_combo = QComboBox()
for fmt in FileFormat:
    format_combo.addItem(fmt.value, fmt)
file_layout.addWidget(QLabel(self.translator.tr('file_format_label')))
file_layout.addWidget(format_combo)
path_edit = QLineEdit()
browse_btn = QPushButton(self.translator.tr('browse_button'))
browse_btn.clicked.connect(lambda: browse_file())
path_layout = QHBoxLayout()
path_layout.addWidget(path_edit)
path_layout.addWidget(browse_btn)
file_layout.addLayout(path_layout)
layout.addWidget(file_group)
# Группа для БД
db_group = QGroupBox(self.translator.tr('db_group'))
db_layout = QVBoxLayout(db_group)
db_type_combo = QComboBox()
db_type_combo.addItems(["postgresql", "mysql", "mssql", "oracle", "sqlite"])
db_layout.addWidget(QLabel(self.translator.tr('db_type_label')))
db_layout.addWidget(db_type_combo)
self.db_host_label = QLabel(self.translator.tr('db_host_label'))
self.db_host = QLineEdit("localhost")
self.db_port_label = QLabel(self.translator.tr('db_port_label'))
self.db_port = QLineEdit()
self.db_name_label = QLabel(self.translator.tr('db_name_label'))
self.db_name = QLineEdit()
self.db_user_label = QLabel(self.translator.tr('db_user_label'))
self.db_user = QLineEdit()
self.db_pass_label = QLabel(self.translator.tr('db_pass_label'))
self.db_pass = QLineEdit()
self.db_pass.setEchoMode(QLineEdit.Password)
db_layout.addWidget(self.db_host_label)
db_layout.addWidget(self.db_host)
db_layout.addWidget(self.db_port_label)
db_layout.addWidget(self.db_port)
db_layout.addWidget(self.db_name_label)
db_layout.addWidget(self.db_name)
db_layout.addWidget(self.db_user_label)
db_layout.addWidget(self.db_user)
db_layout.addWidget(self.db_pass_label)
db_layout.addWidget(self.db_pass)
layout.addWidget(db_group)
# Группа для URL
url_group = QGroupBox(self.translator.tr('url_group'))
url_layout = QVBoxLayout(url_group)
url_edit = QLineEdit()
method_combo = QComboBox()
method_combo.addItems(["GET", "POST"])
url_layout.addWidget(QLabel(self.translator.tr('url_label')))
url_layout.addWidget(url_edit)
url_layout.addWidget(QLabel(self.translator.tr('http_method_label')))
url_layout.addWidget(method_combo)
layout.addWidget(url_group)
def toggle_groups():
    st = type_combo.currentData()
    file_group.setVisible(st == SourceType.FILE)
    db_group.setVisible(st == SourceType.DATABASE)
    url_group.setVisible(st == SourceType.URL)
    if st == SourceType.DATABASE:
        toggle_db_fields()
def toggle_db_fields():
    is_sqlite = db_type_combo.currentText() == 'sqlite'
    self.db_host_label.setVisible(not is_sqlite)
    self.db_host.setVisible(not is_sqlite)
    self.db_port_label.setVisible(not is_sqlite)
    self.db_port.setVisible(not is_sqlite)
    self.db_user_label.setVisible(not is_sqlite)
    self.db_user.setVisible(not is_sqlite)
    self.db_pass_label.setVisible(not is_sqlite)
    self.db_pass.setVisible(not is_sqlite)
    self.db_name_label.setText("Путь к файлу SQLite:" if is_sqlite else self.translator.tr('db_name_label'))
    if is_sqlite:
        if not hasattr(self, 'browse_db_btn'):
            self.browse_db_btn = QPushButton("Обзор...")
            self.browse_db_btn.clicked.connect(browse_db_file)
            db_layout.insertWidget(db_layout.indexOf(self.db_name) + 1, self.browse_db_btn)
        self.browse_db_btn.setVisible(True)
    else:
        if hasattr(self, 'browse_db_btn'):
            self.browse_db_btn.setVisible(False)
def browse_db_file():
    path = QFileDialog.getOpenFileName(dlg, "Выберите файл SQLite")
    if path:
        self.db_name.setText(path)
def browse_file():
    path = QFileDialog.getOpenFileName(dlg, self.translator.tr('select_file_dialog'))
    if path:
        path_edit.setText(path)
type_combo.currentIndexChanged.connect(lambda: toggle_groups())
db_type_combo.currentTextChanged.connect(lambda: toggle_db_fields())
toggle_groups()
# Заполнение полей из профиля
if existing_profile:
    profile = existing_profile
    idx = type_combo.findData(profile.source_type)
    if idx >= 0:
        type_combo.setCurrentIndex(idx)
        toggle_groups()
    if profile.source_type == SourceType.FILE:
        if profile.source_path:
            path_edit.setText(profile.source_path)
        if profile.file_format:
            fmt_idx = format_combo.findData(FileFormat(profile.file_format))
            if fmt_idx >= 0:
                format_combo.setCurrentIndex(fmt_idx)
    elif profile.source_type == SourceType.DATABASE:
        if profile.db_type:
            db_idx = db_type_combo.findText(profile.db_type)
            if db_idx >= 0:
                db_type_combo.setCurrentIndex(db_idx)
                toggle_db_fields()
        if profile.db_host: self.db_host.setText(profile.db_host)
        if profile.db_port: self.db_port.setText(str(profile.db_port))
        if profile.db_name: self.db_name.setText(profile.db_name)
        if profile.db_user: self.db_user.setText(profile.db_user)
        if profile.db_pass: self.db_pass.setText(profile.db_pass)
    elif profile.source_type == SourceType.URL:
        if profile.source_path: url_edit.setText(profile.source_path)
btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
btn_box.accepted.connect(dlg.accept)
btn_box.rejected.connect(dlg.reject)
layout.addWidget(btn_box)
if dlg.exec_() != QDialog.Accepted:
    return
try:
    st = type_combo.currentData()
    if st == SourceType.FILE:
        connector = FileConnector(path_edit.text(), format_combo.currentData())
    elif st == SourceType.DATABASE:
        params = {
            'db_type': db_type_combo.currentText(),
            'host': self.db_host.text(),
            'port': self.db_port.text() or None,
            'database': self.db_name.text(),
            'user': self.db_user.text(),
            'password': self.db_pass.text()
        }
        connector = DBConnector("", **params)
    else:
        connector = URLConnector(url_edit.text())
    connector.connect()
except Exception as e:
    QMessageBox.critical(self, self.translator.tr('error'), str(e))
    return
designer = ImportDesignerV2(self.db, connector, self)
if existing_profile:
```

```
        designer.editing_profile_id = existing_profile.id
        # Восстанавливаем связи после загрузки сцены
        designer.load_profile_links(existing_profile)
    designer.show()
    self.load_profiles()
```

Файл: /home/sergey/Documents/configurate/mozart_import/graph/__init__.py (относительн

```
# -*- coding: utf-8 -*-
```

Файл: /home/sergey/Documents/configurate/mozart_import/graph/graphics_items.py (относ

```
# mozart_import/graph/graphics_items.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import (QGraphicsRectItem, QGraphicsTextItem,
                             QGraphicsProxyWidget, QCheckBox, QGraphicsItem)
from PyQt5.QtCore import QRectF, Qt
from PyQt5.QtGui import QBrush, QColor, QPen, QFont
from ..core.models import Field, NodeType
SOURCE_ID_FIELD = 'sourceid'

class FieldItem(QGraphicsRectItem):
    """Поле внутри прямоугольника таблицы/сущности. Может иметь чекбокс (для источника)."""
    def __init__(self, field: Field, show_checkbox: bool = False, is_sourceid: bool = False, parent=None):
        super().__init__(parent)
        self.field = field
        self.show_checkbox = show_checkbox
        self.is_sourceid = is_sourceid
        width = 200
        self.setRect(0, 0, width, 22)
        if is_sourceid:
            self.setBrush(QBrush(QColor(200, 200, 200)))
            self.setPen(QPen(Qt.darkGray))
        else:
            self.setBrush(QBrush(QColor(240, 240, 240)))
            self.setPen(QPen(Qt.black))
        self.setFlag(QGraphicsItem.ItemIsSelectable)
        self.setAcceptHoverEvents(True)
        y_offset = 2
        if show_checkbox and not is_sourceid:
            self.checkbox = QCheckBox()
            self.checkbox.setChecked(True)
            self.proxy = QGraphicsProxyWidget(self)
            self.proxy.setWidget(self.checkbox)
            self.proxy.setPos(5, y_offset)
            text_x = 25
        else:
            self.checkbox = None
            text_x = 5 if not is_sourceid else 25
        label = f"{field.name}" if is_sourceid else field.name
        self.text = QGraphicsTextItem(label, self)
        self.text.setDefaultTextColor(Qt.black)
        self.text.setPos(text_x, y_offset)
    def hoverEnterEvent(self, event):
        if not self.is_sourceid:
            self.setBrush(QBrush(QColor(200, 230, 255)))
        super().hoverEnterEvent(event)
    def hoverLeaveEvent(self, event):
        if not self.is_sourceid:
            self.setBrush(QBrush(QColor(240, 240, 240)))
        super().hoverLeaveEvent(event)
    def is_checked(self) -> bool:
        if self.checkbox:
            return self.checkbox.isChecked()
        return False
    def set_checked(self, state: bool):
        if self.checkbox:
            self.checkbox.setChecked(state)
    def boundingRect(self):
        return self.rect()

class NodeItem(QGraphicsRectItem):
    """Прямоугольник таблицы/сущности с чекбоксом в заголовке (для выбора таблицы) и чекбоксами полей
(для источника)."""
    def __init__(self, node, parent=None):
        super().__init__(parent)
        self.node = node
        self.field_items = []
        self.setFlag(QGraphicsItem.ItemIsMovable)
        self.setFlag(QGraphicsItem.ItemIsSelectable)
        self.setFlag(QGraphicsItem.ItemSendsGeometryChanges)
        width = 240
        field_count = len(node.fields)
        height = 30 + field_count * 22
        self.setRect(0, 0, width, height)
        if node.node_type == NodeType.SOURCE_TABLE:
            bg_color = QColor(220, 255, 220)  # светло-зелёный
        else:
            bg_color = QColor(255, 255, 200)  # светло-жёлтый
        self.setBrush(QBrush(bg_color))
        self.setPen(QPen(Qt.black, 2))
        # Чекбокс заголовка (выбор всей таблицы)
        self.checkbox = QCheckBox()
        self.checkbox.setChecked(False)
        self.proxy = QGraphicsProxyWidget(self)
        self.proxy.setWidget(self.checkbox)
        self.proxy.setPos(5, 5)
        self.title = QGraphicsTextItem(node.name, self)
        self.title.setDefaultTextColor(Qt.darkBlue)
        self.title.setFont(QFont("Arial", 10, QFont.Bold))
        self.title.setPos(25, 5)
        # Поля
        y = 30
        show_checkbox = (node.node_type == NodeType.SOURCE_TABLE)
        for field in node.fields:
            is_sid = (field.name == SOURCE_ID_FIELD) and (node.node_type == NodeType.MOZART_ENTITY)
            fi = FieldItem(field, show_checkbox=show_checkbox, is_sourceid=is_sid, parent=self)
            fi.setPos(10, y)
            self.field_items.append(fi)
            y += 22
        self.setPos(node.x, node.y)
    def itemChange(self, change, value):
        if change == QGraphicsItem.ItemPositionChange:
            new_pos = value
            new_pos.setY(max(0, new_pos.y()))
            if self.scene() and hasattr(self.scene(), 'divider_x'):
                divider_x = self.scene().divider_x
                if self.node.node_type == NodeType.SOURCE_TABLE:
                    max_x = divider_x - self.rect().width() - 10
                    new_pos.setX(max(10, min(new_pos.x(), max_x)))
                else:
                    min_x = divider_x + 10
                    new_pos.setX(max(min_x, new_pos.x()))
            return new_pos
        return super().itemChange(change, value)
    def mouseReleaseEvent(self, event):
        """После перемещения вручную обновляем линии."""
        super().mouseReleaseEvent(event)
        # Обновляем все линии
        if self.scene() and hasattr(self.scene(), 'update_all_links'):
            self.scene().update_all_links()
```

```
    def update_position(self):
        self.node.x = self.pos().x()
        self.node.y = self.pos().y()
    def is_checked(self) -> bool:
        """Чекбокс заголовка."""
        return self.checkbox.isChecked()
    def set_checked(self, checked: bool):
        self.checkbox.setChecked(checked)
    def get_checked_fields(self):
        """Возвращает имена полей, отмеченных чекбоксами (только для источника)."""
        if self.node.node_type != NodeType.SOURCE_TABLE:
            return []
        return [fi.field.name for fi in self.field_items if fi.is_checked()]
    def set_all_checked(self, checked: bool):
        """Отметить/снять все поля (источник)."""
        for fi in self.field_items:
            if not fi.is_source:
                fi.set_checked(checked)
```

Файл: /home/sergey/Documents/configurate/mozart_import/graph/graphics_view.py (относ

```
# mozart_import/graph/graphics_view.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QGraphicsView
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QPainter, QWheelEvent
from .link_line import LinkLine

class GraphicsView(QGraphicsView):
    def __init__(self, scene, parent=None):
        super().__init__(scene, parent)
        self.setRenderHint(QPainter.Antialiasing)
        self.setDragMode(self.ScrollHandDrag)
        self.setTransformationAnchor(self.AnchorUnderMouse)
        self.setResizeAnchor(self.AnchorUnderMouse)
        self.setFocusPolicy(Qt.StrongFocus)  # чтобы получать события клавиш
        self.scale_factor = 1.0
    def wheelEvent(self, event: QWheelEvent):
        if event.modifiers() == Qt.ControlModifier:
            delta = event.angleDelta().y()
            if delta > 0:
                self.scale_view(1.25)
            elif delta < 0:
                self.scale_view(0.8)
        else:
            super().wheelEvent(event)
    def scale_view(self, factor):
        self.scale_factor *= factor
        self.scale(factor, factor)
    def keyPressEvent(self, event):
        """Обработка клавиши Delete для удаления выделенных линий."""
        if event.key() == Qt.Key_Delete:
            scene = self.scene()
            for item in scene.selectedItems():
                if isinstance(item, LinkLine):
                    # Вызываем метод удаления линии из сцены
                    if hasattr(scene, 'remove_link'):
                        scene.remove_link(item)
            event.accept()
        else:
            super().keyPressEvent(event)
```

Файл: /home/sergey/Documents/configurate/mozart_import/graph/link_line.py (относитель

```
# mozart_import/graph/link_line.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QGraphicsLineItem
from PyQt5.QtCore import QLineF, Qt
from PyQt5.QtGui import QPen, QColor
from ..core.models import LinkType

class LinkLine(QGraphicsLineItem):
    """
    Линия связи между двумя полями.
    Хранит полный набор маппингов для данной связи импорта.
    Теперь поддерживает выделение и удаление.
    """
    COLORS = {
        LinkType.FK_SOURCE: QColor(200, 30, 30),  # красный
        LinkType.FK_TARGET: QColor(100, 100, 140),  # сине-серый
        LinkType.COMPOSITION: QColor(30, 150, 30),  # зеленый
    }
    def __init__(self, start_item, start_field_item, end_item, end_field_item,
                 link_type: LinkType, mappings=None):
        super().__init__()
        self.start_item = start_item
        self.start_field_item = start_field_item
        self.end_item = end_item
        self.end_field_item = end_field_item
        self.link_type = link_type
        # Маппинги: список словарей {source_field, target_field, is_source_id}
        self.mappings = mappings or []
        # Если маппинги не заданы, создаём один на основе начального поля
        if not self.mappings and link_type == LinkType.IMPORT:
            self.mappings = [{
                'source_field': start_field_item.field.name,
                'target_field': end_field_item.field.name,
                'is_source_id': False
            }]
        color = self.COLORS.get(link_type, Qt.black)
        width = 3 if link_type == LinkType.IMPORT else 1
        style = Qt.SolidLine if link_type == LinkType.IMPORT else Qt.DashLine
        # Пера для обычного и выделенного состояния
        self._normal_pen = QPen(color, width, style)
        self._selected_pen = QPen(Qt.red, width + 1, style)
        self.setPen(self._normal_pen)
        self.setZValue(-1)
        # Разрешаем выделение линии
        self.setFlag(QGraphicsLineItem.ItemIsSelectable, True)
        self.setFlag(QGraphicsLineItem.ItemSendsGeometryChanges, True)
        self.update_position()
    def itemChange(self, change, value):
        """При выделении меняем цвет линии."""
        if change == QGraphicsLineItem. itemSelectedChange:
            if value:
                self.setPen(self._selected_pen)
            else:
                self.setPen(self._normal_pen)
        return super().itemChange(change, value)
    def update_position(self):
        if not self.start_field_item or not self.end_field_item:
            return
        sp = self.start_field_item.scenePos()
        sp.setX(sp.x() + self.start_field_item.boundingRect().width())
        sp.setY(sp.y() + self.start_field_item.boundingRect().height() / 2)
        ep = self.end_field_item.scenePos()
        ep.setX(ep.x())
        ep.setY(ep.y() + self.end_field_item.boundingRect().height() / 2)
        self.setLine(QLineF(sp, ep))
    def update_mappings_from_dialog(self, new_mappings):
        """Обновить маппинг после диалога."""
        self.mappings = new_mappings
```

Файл: /home/sergey/Documents/configurate/mozart_import/graph/relation_scene.py (относ

```
# mozart_import/graph/relation_scene.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import (QGraphicsScene, QGraphicsItem, QMenu, QAction,
                             QMessageBox, QDialog, QTableWidget, QTableWidgetItem,
                             QVBoxLayout, QHeaderView)
from PyQt5.QtCore import QPointF
from PyQt5.QtGui import QColor, QTransform
from .graphics.items import NodeItem, FieldItem, SOURCE_ID_FIELD
from .link_line import LinkLine
from ..core.models import Node, NodeType, LinkType

class RelationScene(QGraphicsScene):
    """Графическая сцена конструктора импорта с виртуальным разделителем."""
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setBackgroundBrush(QColor(245, 245, 245))
        scene_width = 3000
        scene_height = 1000
        self.setSceneRect(0, 0, scene_width, scene_height)
        self.divider_x = scene_width / 2
        self.nodes = {}
        self.links = []
        # Коллбэки
        self.on_edit_link = None
        self.on_preview_table = None
        self.on_create_entity = None
        self.on_create_fields_from_source = None
        self.on_set_source_refs = None
    def clear_scene(self):
        for node_item in list(self.nodes.values()):
            self.removeItem(node_item)
        for link in self.links:
            self.removeItem(link)
        self.nodes.clear()
        self.links.clear()
    def add_node(self, node: Node):
        item = NodeItem(node)
        if node.node_type == NodeType.SOURCE_TABLE:
            max_x = self.divider_x - item.rect().width() - 10
            if node.x > max_x:
                node.x = max_x
            if node.x < 10:
                node.x = 10
        else:
            if node.x < self.divider_x + 10:
                node.x = self.divider_x + 10
        item.setPos(node.x, node.y)
        self.addItem(item)
        self.nodes[node.id] = item
        return item
    def add_link(self, start_node_item, end_node_item, link_type: LinkType, mappings=None):
        if link_type == LinkType.IMPORT:
            if (start_node_item.node.node_type != NodeType.SOURCE_TABLE or
                end_node_item.node.node_type != NodeType.MOZART_ENTITY):
                return None
            if mappings:
                first_m = mappings[0]
                last_m = mappings[-1]
                src_fi = self._find_field_item(start_node_item, first_m['source_field'])
                tgt_fi = self._find_field_item(end_node_item, last_m.get('target_field'))
                if not src_fi or not tgt_fi:
                    if start_node_item.field_items and end_node_item.field_items:
                        src_fi = start_node_item.field_items[0]
                        tgt_fi = end_node_item.field_items[0]
                    else:
                        return None
                link = LinkLine(start_node_item, src_fi, end_node_item, tgt_fi, link_type, mappings=mappings)
                self.addItem(link)
                self.links.append(link)
                return link
            else:
                if start_node_item.field_items and end_node_item.field_items:
                    src_fi = start_node_item.field_items[0]
                    tgt_fi = end_node_item.field_items[0]
                    link = LinkLine(start_node_item, src_fi, end_node_item, tgt_fi, link_type)
                    self.addItem(link)
                    self.links.append(link)
                    return link
        elif link_type == LinkType.FK_SOURCE:
            if (start_node_item.node.node_type != NodeType.SOURCE_TABLE or
                end_node_item.node.node_type != NodeType.SOURCE_TABLE):
                return None
            if start_node_item.field_items and end_node_item.field_items:
                src_fi = start_node_item.field_items[0]
                tgt_fi = end_node_item.field_items[0]
                link = LinkLine(start_node_item, src_fi, end_node_item, tgt_fi, link_type)
                self.addItem(link)
                self.links.append(link)
                return link
        return None
    def remove_link(self, link_line: LinkLine):
        """Удаляет линию со сцены и из списка links."""
        if link_line in self.links:
            self.links.remove(link_line)
        self.removeItem(link_line)
    def _find_field_item(self, node_item, field_name):
        for fi in node_item.field_items:
            if fi.field.name == field_name:
                return fi
        return None
    # ---------- Контекстные меню ----------
    def contextMenuEvent(self, event):
        item = self.itemAt(event.scenePos(), QTransform())
        if isinstance(item, NodeItem):
            self._show_node_context_menu(item, event.screenPos())
        elif isinstance(item, LinkLine):
            self._show_link_context_menu(item, event.screenPos())
        else:
            self._show_scene_context_menu(event.screenPos())
    def _show_node_context_menu(self, node_item, screen_pos):
        menu = QMenu()
        if node_item.node.node_type == NodeType.SOURCE_TABLE:
            act_preview = QAction("Быстрый просмотр", None)
            act_preview.triggered.connect(lambda: self._emit_preview(node_item))
            menu.addAction(act_preview)
            act_create = QAction("Создать сущность Mozart", None)
            act_create.triggered.connect(lambda: self._emit_create_entity(node_item))
            menu.addAction(act_create)
            menu.addSeparator()
            act_refs = QAction("Ссылочные поля (связность)", None)
            act_refs.triggered.connect(lambda: self._emit_set_source_refs(node_item))
            menu.addAction(act_refs)
            menu.addSeparator()
            act_all = QAction("Выбрать все поля", None)
            act_all.triggered.connect(lambda: node_item.set_all_checked(True))
            menu.addAction(act_all)
            act_none = QAction("Снять все поля", None)
            act_none.triggered.connect(lambda: node_item.set_all_checked(False))
            menu.addAction(act_none)
        else:  # Mozart
            act_show_links = QAction("Показать связи импорта", None)
            act_show_links.triggered.connect(lambda: self._show_import_links_for(node_item))
            menu.addAction(act_show_links)
            act_fields = QAction("Создать поля из выделенной таблицы", None)
            act_fields.triggered.connect(lambda: self._emit_create_fields(node_item))
            menu.addAction(act_fields)
        menu.exec_(screen_pos)
```

```
    def _show_link_context_menu(self, link_line, screen_pos):
        menu = QMenu()
        act_edit = QAction("Настроить связь", None)
        act_edit.triggered.connect(lambda: self._emit_edit_link(link_line))
        menu.addAction(act_edit)
        act_del = QAction("Удалить связь", None)
        act_del.triggered.connect(lambda: self.remove_link(link_line))
        menu.addAction(act_del)
        menu.exec_(screen_pos)
    def _show_scene_context_menu(self, screen_pos):
        selected_source = self._get_single_checked(NodeType.SOURCE_TABLE)
        selected_mozart = self._get_single_checked(NodeType.MOZART_ENTITY)
        menu = QMenu()
        if selected_source and selected_mozart:
            act_link = QAction("Связать выбранные", None)
            act_link.triggered.connect(lambda: self._link_selected(selected_source, selected_mozart))
            menu.addAction(act_link)
        else:
            hint = "Отметьте чекбоксами по одной таблице источника и сущности Mozart"
            menu.addAction(hint)
        menu.addSeparator()
        act_refresh = QAction("Обновить все связи", None)
        act_refresh.triggered.connect(self.update_all_links)
        menu.addAction(act_refresh)
        menu.exec_(screen_pos)
    def _get_single_checked(self, node_type):
        checked = []
        for ni in self.nodes.values():
            if ni.node.node_type == node_type and ni.is_checked():
                checked.append(ni)
        return checked[0] if len(checked) == 1 else None
    def _link_selected(self, source_node_item, mozart_node_item):
        checked_source_fields = source_node_item.get_checked_fields()
        if not checked_source_fields:
            checked_source_fields = [fi.field.name for fi in source_node_item.field_items]
        target_fields = [fi.field.name for fi in mozart_node_item.field_items]
        if SOURCE_ID_FIELD not in target_fields:
            target_fields.append(SOURCE_ID_FIELD)
        existing_mappings = []
        for link in self.links:
            if (link.link_type == LinkType.IMPORT and
                link.start_item == source_node_item and
                link.end_item == mozart_node_item):
                existing_mappings = link.mappings
                break
        if not existing_mappings:
            for d in self.get_import_links_data():
                if (d['source_table'] == source_node_item.node.table_name and
                    d['target_entity_id'] == mozart_node_item.node.entity_id):
                    existing_mappings = d.get('mappings', [])
                    break
        if self.on_edit_link:
            source_refs = getattr(source_node_item.node, 'refs', {})
            self.on_edit_link(source_node_item, mozart_node_item, existing_mappings,
                              checked_source_fields, source_refs)
        else:
            QMessageBox.information(None, "Информация", "Обработчик настройки связи не назначен")
    def _show_import_links_for(self, mozart_node_item):
        entity_name = mozart_node_item.node.name
        relevant_links = []
        for link in self.links:
            if link.link_type == LinkType.IMPORT and link.end_item == mozart_node_item:
                relevant_links.append(link)
        if not relevant_links:
            QMessageBox.information(None, f"Связи для {entity_name}", "Нет назначенных импортируемых связей.")
            return
        dlg = QDialog()
        dlg.setWindowTitle(f"Связи импорта: {entity_name}")
        layout = QVBoxLayout(dlg)
        table = QTableWidget(0, 5)
        table.setHorizontalHeaderLabels(["Источник (таблица.поле)", "Поле Mozart", "Ключевое",
                                         "Ссылка?", "Цель (табл.поле)"])
        table.horizontalHeader().setStretchLastSection(True)
        for link in relevant_links:
            src_table = link.start_item.node.name
            for m in link.mappings:
                row = table.rowCount()
                table.insertRow(row)
                table.setItem(row, 0, QTableWidgetItem(f"{src_table}.{m['source_field']}"))
                table.setItem(row, 1, QTableWidgetItem(m['target_field']))
                is_key = m.get('is_source_id', False)
                table.setItem(row, 2, QTableWidgetItem(is_key))
                is_ref = m.get('is_reference', False)
                table.setItem(row, 3, QTableWidgetItem(is_ref))
                if m.get('is_reference'):
                    ref_info = f"{m.get('ref_source_table','')}.{m.get('ref_source_key_field','')}"
                else:
                    ref_info = ""
                table.setItem(row, 4, QTableWidgetItem(ref_info))
        layout.addWidget(table)
        dlg.exec_()
    def update_all_links(self):
        for link in self.links:
            link.update_position()
    def get_import_links_data(self):
        data = []
        for link in self.links:
            if link.link_type == LinkType.IMPORT:
                data.append({
                    'source_table': link.start_item.node.table_name,
                    'target_entity_id': link.end_item.node.entity_id,
                    'mappings': link.mappings
                })
        return data
    # Эмиттеры
    def _emit_preview(self, node_item):
        if self.on_preview_table:
            self.on_preview_table(node_item)
    def _emit_create_entity(self, node_item):
        if self.on_create_entity:
            self.on_create_entity(node_item)
    def _emit_create_fields(self, node_item):
        if self.on_create_fields_from_source:
            self.on_create_fields_from_source(node_item)
    def _emit_edit_link(self, link_line):
        if self.on_edit_link:
            source_node_item = link_line.start_item
            checked_fields = source_node_item.get_checked_fields()
            if not checked_fields:
                checked_fields = [fi.field.name for fi in source_node_item.field_items]
            source_refs = getattr(source_node_item.node, 'refs', {})
            self.on_edit_link(link_line.start_item, link_line.end_item,
                              link_line.mappings, checked_fields, source_refs)
    def _emit_set_source_refs(self, node_item):
        if self.on_set_source_refs:
            self.on_set_source_refs(node_item)
```

Файл: /home/sergey/Documents/configurate/mozart_import/matchers/__init__.py (относите

```
# mozart_import/matchers/__init__.py
from .name_matcher import NameMatcher
from .type_matcher import TypeMatcher
from .constraint_matcher import ConstraintMatcher
__all__ = [
    'NameMatcher',
    'TypeMatcher',
    'ConstraintMatcher'
```

]

Файл: /home/sergey/Documents/configurate/mozart_import/matchers/constraint_matcher.p

```
# mozart_import/matchers/constraint_matcher.py
from typing import List, Dict, Any

class ConstraintMatcher:
    """Сопоставление по ограничениям (первичные ключи, внешние ключи, уникальность)."""
    @staticmethod
    def is_primary_key_candidate(field_name: str) -> bool:
        """Проверяет, похоже ли имя поля на первичный ключ."""
        name_lower = field_name.lower()
        return name_lower in ('id', 'code', 'num', 'key', 'pk')
    @staticmethod
    def is_foreign_key_candidate(field_name: str) -> bool:
        """Проверяет, похоже ли имя поля на внешний ключ."""
        name_lower = field_name.lower()
        return name_lower.endswith('_id') or name_lower.startswith('id_') or name_lower in ('parentid', 'parent_id')
    @staticmethod
    def extract_referenced_table(field_name: str, current_table: str) -> str:
        """По имени внешнего ключа предполагает имя целевой таблицы."""
        name_lower = field_name.lower()
        if name_lower.endswith('_id'):
            target = name_lower[:-3]
        elif name_lower.startswith('id_'):
            target = name_lower[3:]
        else:
            target = name_lower
        if target == 'parent':
            target = current_table
        return target
```

Файл: /home/sergey/Documents/configurate/mozart_import/matchers/name_matcher.py (от

```
# mozart_import/matchers/name_matcher.py
from difflib import SequenceMatcher
from typing import Tuple, Optional

class NameMatcher:
    """Сопоставление по имени (таблиц, полей)."""
    @staticmethod
    def similarity(a: str, b: str) -> float:
        return SequenceMatcher(None, a.lower(), b.lower()).ratio()
    @staticmethod
    def best_match(name: str, candidates: list, threshold: float = 0.6) -> Tuple[Optional[str], float]:
        best = None
        best_score = 0.0
        for cand in candidates:
            score = NameMatcher.similarity(name, cand)
            if score > best_score:
                best_score = score
                best = cand
        if best_score >= threshold:
            return best, best_score
        return None, 0.0
```

Файл: /home/sergey/Documents/configurate/mozart_import/matchers/type_matcher.py (отн

```
# mozart_import/matchers/type_matcher.py
from ..core.models import FieldType

class TypeMatcher:
    """Сопоставление типов данных источника и Mozart."""
    # Маппинг типов источника -> коды типов Mozart (C, I, N, D, T, L, M, R)
    TYPE_MAPPING = {
        FieldType.STRING: 'C',
        FieldType.INTEGER: 'I',
        FieldType.FLOAT: 'N',
        FieldType.BOOLEAN: 'L',
        FieldType.DATE: 'D',
        FieldType.DATETIME: 'T',
        FieldType.TIME: 'T',
        FieldType.BINARY: 'F',
    }
    @classmethod
    def get_mozart_type(cls, source_type: FieldType) -> str:
        return cls.TYPE_MAPPING.get(source_type, 'C')
    @classmethod
    def is_compatible(cls, source_type: FieldType, mozart_type: str) -> bool:
        target = cls.get_mozart_type(source_type)
        if target == mozart_type:
            return True
        # Дополнительные совместимости
        if source_type == FieldType.INTEGER and mozart_type == 'N':
            return True
        if source_type == FieldType.FLOAT and mozart_type == 'N':
            return True
        if source_type == FieldType.STRING and mozart_type == 'C':
            return True
        return False
```

Файл: /home/sergey/Documents/configurate/mozart_import/utils/__init__.py (относительно

```
# mozart_import/utils/__init__.py
from .ext_helper import ExtHelper
from .log_manager import LogManager
__all__ = ['ExtHelper', 'LogManager']
```

Файл: /home/sergey/Documents/configurate/mozart_import/utils/ext_helper.py (относитель

```
# mozart_import/utils/ext_helper.py
from database import DatabaseService

class ExtHelper:
    """Вспомогательный класс для работы с ext-таблицами и полями."""
    def __init__(self, db: DatabaseService):
        self.db = db
    def get_ext_table_name(self, entity_alias: str) -> str:
        return f'ext.ext_{entity_alias.lower()}'
    def ensure_ext_table(self, entity_id: int) -> bool:
        """Проверяет существование ext-таблицы, создаёт при необходимости."""
        alias_res = self.db.execute_query("SELECT calias FROM meta.entitytypes WHERE id=%s", (entity_id,))
        if not alias_res:
            return False
        alias = alias_res[0][0].lower()
        ext_table = self.get_ext_table_name(alias)
        check = self.db.execute_query("SELECT to_regclass(%s)", (ext_table,))
        if check and check[0][0]:
            return True
        sql = f"""
            CREATE TABLE IF NOT EXISTS {ext_table} (
                id SERIAL PRIMARY KEY,
                versionid INTEGER NOT NULL REFERENCES data.entity_versions(id) ON DELETE CASCADE
        """)
        self.db.execute_query(sql, fetch=False)
        return True
```

```
    def ensure_field(self, entity_id: int, field_name: str, field_type: str = 'C',
                     calias: str = None, ref_entity_id: int = None, length: int = 255) -> int:
        """
        Создаёт поле в meta.fields (если его нет) и вызывает p_field_sync.
        Возвращает id поля.
        """
        if calias is None:
            calias = field_name.capitalize()
        # Проверяем существование
        existing = self.db.execute_query(
            "SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s",
            (entity_id, field_name)
        )
        if existing:
            return existing[0][0]
        res = self.db.execute_query(
            "INSERT INTO meta.fields
                (entitytypeid, cfieldname, cfieldtype, calias, nlength, lisindexed, ref_entitytypeid, status_id)
                VALUES (%s, %s, %s, %s, %s, %s, %s, 1)
                RETURNING id",
            (entity_id, field_name, field_type, calias, length, True, ref_entity_id)
        )
        field_id = res[0][0]
        self.db.execute_procedure("meta.p_field_sync", (field_id,))
        return field_id
```

Файл: /home/sergey/Documents/configurate/mozart_import/utils/field_creator.py (относите

```
# mozart_import/utils/field_creator.py
# -*- coding: utf-8 -*-
from database import DatabaseService

class FieldCreator:
    def __init__(self, db: DatabaseService):
        self.db = db
    def create_fields(self, entity_id: int, fields: list) -> list:
        created_ids = []
        for f in fields:
            exists = self.db.execute_query(
                "SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s",
                (entity_id, f['cfieldname'])
            )
            if exists:
                created_ids.append(exists[0][0])
                continue
            cfieldname = f['cfieldname'][:63]
            calias = f.get('calias', cfieldname.capitalize())[:50]
            cfieldtype = f.get('cfieldtype', 'C')
            nlength = f.get('nlength', 255)
            ndecimals = f.get('ndecimals', 0)
            lisindexed = f.get('lisindexed', True)
            ref_entitytypeid = f.get('ref_entitytypeid', None)
            res = self.db.execute_query(
                "INSERT INTO meta.fields
                    (entitytypeid, cfieldname, cfieldtype, calias,
                    nlength, ndecimals, lisindexed, ref_entitytypeid, status_id)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, 1)
                    RETURNING id",
                (entity_id, cfieldname, cfieldtype, calias,
                 nlength, ndecimals, lisindexed, ref_entitytypeid)
            )
            if res:
                field_id = res[0][0]
                self.db.execute_procedure("meta.p_field_sync", (field_id,))
                created_ids.append(field_id)
            else:
                raise Exception(f"Failed to create field {cfieldname}")
        return created_ids
```

Файл: /home/sergey/Documents/configurate/mozart_import/utils/log_manager.py (относите

```
# mozart_import/utils/log_manager.py
import logging
from datetime import datetime

class LogManager:
    """Простой логгер для импорта."""
    def __init__(self, name="mozart_import"):
        self.logger = logging.getLogger(name)
        self.logger.setLevel(logging.INFO)
        if not self.logger.handlers:
            handler = logging.StreamHandler()
            formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
            handler.setFormatter(formatter)
            self.logger.addHandler(handler)
    def info(self, msg):
        self.logger.info(msg)
    def error(self, msg):
        self.logger.error(msg)
    def warning(self, msg):
        self.logger.warning(msg)
    def debug(self, msg):
        self.logger.debug(msg)
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/__init__.py (относительн

```
# mozart_import/wizards/__init__.py
# -*- coding: utf-8 -*-
from .import_wizard import ImportWizard
from .source_page import SourcePage
from .schema_page import SchemaPage
from .relation_page import RelationPage
from .mapping_detail_page import MappingDetailPage
from .execute_page import ExecutePage

__all__ = [
    'ImportWizard',
    'SourcePage',
    'SchemaPage',
    'RelationPage',
    'MappingDetailPage',
    'ExecutePage'
]
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/execute_page.py (отно

```
# mozart_import/wizards/execute_page.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QWizardPage, QVBoxLayout, QProgressBar, QTextEdit, QPushButton,
QHBoxLayout, QMessageBox
from PyQt5.QtCore import QThread, pyqtSignal
from ..utils.translator import tr, TranslatorMixin
from ..core.models import ImportProfile, SourceType, MappingProposal, MatchedEntity, MatchedField
from ..core.profile_manager import ProfileManager
from ..core.import_engine import ImportEngine
from ..utils.field_creator import FieldCreator

class ImportThread(QThread):
    progress = pyqtSignal(int, int)
    finished = pyqtSignal(dict)
    error = pyqtSignal(str)
    def __init__(self, db, connector, profile):
```

```
        super().__init__()
        self.db = db
        self.connector = connector
        self.profile = profile
    def run(self):
        try:
            engine = ImportEngine(self.db, self.connector, self.profile)
            stats = engine.run(progress_callback=self.progress.emit)
            self.finished.emit(stats)
        except Exception as e:
            self.error.emit(str(e))

class ExecutePage(QWizardPage):
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = LocalTranslator()
        self.setTitle(self.translator.tr('execute_page_title'))
        self.setSubTitle(self.translator.tr('execute_page_subtitle'))
        layout = QVBoxLayout(self)
        self.progress = QProgressBar()
        layout.addWidget(self.progress)
        self.log = QTextEdit()
        self.log.setReadOnly(True)
        layout.addWidget(self.log)
        btn_layout = QHBoxLayout()
        self.btn_start = QPushButton(self.translator.tr('start_import_btn'))
        self.btn_start.clicked.connect(self.start_import)
        self.btn_start.setEnabled(False)
        btn_layout.addWidget(self.btn_start)
        layout.addLayout(btn_layout)
        self.import_thread = None
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('execute_page_title'))
        self.setSubTitle(self.translator.tr('execute_page_subtitle'))
        self.btn_start.setText(self.translator.tr('start_import_btn'))
    def initializePage(self):
        self.prepare_profile()
        self.btn_start.setEnabled(True)
    def prepare_profile(self):
        # Создаем недостающие сущности и поля
        creator = FieldCreator(self.wizard.db)
        if hasattr(self.wizard, 'pending_entities') and self.wizard.pending_entities:
            for node_item in self.wizard.pending_entities:
                res = self.wizard.db.execute_query(
                    """INSERT INTO meta_entitytypes (cname, calias, class_id, chasclass, lisversioned, isecuritystrategy, status_id)
                        VALUES (%s, %s, %s, true, 1, 1) RETURNING id""",
                    (entity_data['cname'], entity_data['calias'], entity_data['class_id'])
                )
                if res:
                    entity_id = res[0][0]
                    node_item.node.entity_id = entity_id
        # Строим MappingProposal
        source_nodes = {n.id: n for n in getattr(self.wizard, 'source_nodes', [])}
        target_nodes = {}
        if hasattr(self.wizard, 'mozart_nodes'):
            for n in self.wizard.mozart_nodes:
                target_nodes[n.id] = n
        if hasattr(self.wizard, 'pending_entities'):
            for node_item in self.wizard.pending_entities:
                target_nodes[node_item.node.id] = node_item.node
        links = getattr(self.wizard, 'links', [])
        target_groups = {}
        for link in links:
            target_id = link['target_node_id']
            if target_id not in target_groups:
                target_groups[target_id] = []
            target_groups[target_id].append(link)
        entities = []
        for target_id, link_list in target_groups.items():
            target_node = target_nodes.get(target_id)
            if not target_node:
                continue
            source_table_name = link_list[0]['source_node_id'].replace('src_', '')
            matched_fields = []
            for link in link_list:
                if link.get('target_field'):
                    is_ref = False
                    ref_entity_id = None
                    matched_fields.append(MatchedField(
                        source_field=link['source_field'],
                        target_field=link['target_field'],
                        confidence=1.0,
                        is_reference=is_ref,
                        ref_entitytype_id=ref_entity_id
                    ))
            entities.append(MatchedEntity(
                source_table=source_table_name,
                target_entity_id=target_node.entity_id,
                target_entity_name=target_node.name,
                confidence=1.0,
                fields=matched_fields,
                is_new=(target_node.entity_id is None)
            ))
        self.wizard.mapping_proposal = MappingProposal(
            source_schema=self.wizard.schema,
            entities=entities,
            unresolved_tables=[],
            suggestions_for_new_entities=[]
        )
        profile = ImportProfile(
            name=f"Import from {self.wizard.connector.source_path}",
            source_type=self.wizard.connector.source_type,
            source_path=self.wizard.connector.source_path,
            mapping_proposal=self.wizard.mapping_proposal,
            scene_json=getattr(self.wizard, 'import_scene_json', None)
        )
        self.wizard.profile = profile
        prof_mgr = ProfileManager(self.wizard.db)
        prof_mgr.save_profile(profile)
    def start_import(self):
        if not self.wizard.profile:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('profile_not_ready'))
            return
        self.btn_start.setEnabled(False)
        self.import_thread = ImportThread(self.wizard.db, self.wizard.connector, self.wizard.profile)
        self.import_thread.progress.connect(self.update_progress)
        self.import_thread.finished.connect(self.on_finished)
        self.import_thread.error.connect(self.on_error)
        self.import_thread.start()
    def update_progress(self, current, total):
        self.progress.setMaximum(total)
        self.progress.setValue(current)
        self.log.append(f"{self.translator.tr('processed_records')}: {current} / {total}")
    def on_finished(self, stats):
        self.log.append(f"{self.translator.tr('import_finished')}: {self.translator.tr('inserted')}: {stats['inserted']}, "
                        f"{self.translator.tr('updated')}: {stats['updated']}, {self.translator.tr('errors')}: {stats['errors']}")
        QMessageBox.information(self, self.translator.tr('import_finished_title'), self.translator.tr('import_finished'))
        self.btn_start.setEnabled(False)
    def on_error(self, msg):
        self.log.append(f"{self.translator.tr('error')}: {msg}")
        QMessageBox.critical(self, self.translator.tr('error'), msg)
        self.btn_start.setEnabled(True)
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/import_wizard.py (отно

```
# mozart_import/wizards/import_wizard.py
# -*- coding: utf-8 -*-
```

```
from PyQt5.QtWidgets import QWizard
from PyQt5.QtCore import QEvent
from lang.local_translator import LocalTranslator
from database.import_database_service
from .source_page import SourcePage
from .schema_page import SchemaPage
from .relation_page import RelationPage
from .mapping_detail_page import MappingDetailPage
from .execute_page import ExecutePage

class ImportWizard(QWizard):
    def __init__(self, db: DatabaseService, parent=None):
        super().__init__(parent)
        self.db = db
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('wizard_title'))
        self.resize(1200, 800)
        self.setWizardStyle(QWizard.ModernStyle)
        # Данные, передаваемые между страницами
        self.connector = None
        self.schema = None
        self.source_nodes = []
        self.links = []
        self.pending_entities = []
        self.import_links = []
        # Страницы
        self.source_page = SourcePage(self)
        self.schema_page = SchemaPage(self)
        self.relation_page = RelationPage(self)
        self.detail_page = MappingDetailPage(self)
        self.execute_page = ExecutePage(self)
        self.addPage(self.source_page)
        self.addPage(self.schema_page)
        self.addPage(self.relation_page)
        self.addPage(self.detail_page)
        self.addPage(self.execute_page)
        self.setButtonText(QWizard.FinishButton, self.translator.tr('wizard_finish'))
        self.setButtonText(QWizard.CancelButton, self.translator.tr('btn_cancel'))
    def retranslate_ui(self):
        self.setWindowTitle(self.translator.tr('wizard_title'))
        self.setButtonText(QWizard.FinishButton, self.translator.tr('wizard_finish'))
        self.setButtonText(QWizard.CancelButton, self.translator.tr('btn_cancel'))
        self.source_page.retranslate_ui()
        self.schema_page.retranslate_ui()
        self.relation_page.retranslate_ui()
        self.detail_page.retranslate_ui()
        self.execute_page.retranslate_ui()
    def changeEvent(self, event):
        if event.type() == QEvent.LanguageChange:
            self.retranslate_ui()
        super().changeEvent(event)
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/mapping_detail_page.p

```
# mozart_import/wizards/mapping_detail_page.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import (QWizardPage, QVBoxLayout, QHBoxLayout, QTableWidget,
                             QTableWidgetItem, QHeaderView, QPushButton, QMessageBox,
                             QComboBox, QCheckBox, QSpinBox)
from PyQt5.QtCore import Qt
from lang.local_translator import LocalTranslator
from ..utils.field_creator import FieldCreator

class MappingDetailPage(QWizardPage):
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = LocalTranslator()
        self.setTitle(self.translator.tr('detail_page_title'))
        self.setSubTitle(self.translator.tr('detail_page_subtitle'))
        layout = QVBoxLayout(self)
        self.table = QTableWidget(0, 8)
        self.table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_entity'),
            self.translator.tr('target_field'),
            self.translator.tr('mozart_type'),
            self.translator.tr('length'),
            self.translator.tr('source_id'),
            self.translator.tr('action')
        ])
        self.table.horizontalHeader().setStretchLastSection(True)
        layout.addWidget(self.table)
        btn_layout = QHBoxLayout()
        self.btn_apply = QPushButton(self.translator.tr('apply_mapping_btn'))
        self.btn_apply.clicked.connect(self.apply_mapping)
        self.btn_back = QPushButton(self.translator.tr('back_btn'))
        self.btn_back.clicked.connect(lambda: self.wizard.back())
        btn_layout.addWidget(self.btn_back)
        btn_layout.addWidget(self.btn_apply)
        layout.addLayout(btn_layout)
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('detail_page_title'))
        self.setSubTitle(self.translator.tr('detail_page_subtitle'))
        self.table.setHorizontalHeaderLabels([
            self.translator.tr('source_table'),
            self.translator.tr('source_field'),
            self.translator.tr('target_entity'),
            self.translator.tr('target_field'),
            self.translator.tr('mozart_type'),
            self.translator.tr('length'),
            self.translator.tr('source_id'),
            self.translator.tr('action')
        ])
        self.btn_apply.setText(self.translator.tr('apply_mapping_btn'))
        self.btn_back.setText(self.translator.tr('back_btn'))
    def initializePage(self):
        self.load_mappings()
    def load_mappings(self):
        links = getattr(self.wizard, 'links', [])
        source_nodes = {node.id: node for node in getattr(self.wizard, 'source_nodes', [])}
        target_nodes = {}
        if hasattr(self.wizard, 'pending_entities'):
            for node_item in self.wizard.pending_entities:
                target_nodes[node_item.node.id] = node_item.node
        self.table.setRowCount(len(links))
        for row, link in enumerate(links):
            source_node = source_nodes.get(link['source_node_id'])
            target_node = target_nodes.get(link['target_node_id'])
            self.table.setItem(row, 0, QTableWidgetItem(source_node.name if source_node else ''))
            self.table.setItem(row, 1, QTableWidgetItem(link['source_field'] ))
            self.table.setItem(row, 2, QTableWidgetItem(target_node.name if target_node else ''))
            combo_target = QComboBox()
            combo_target.addItem("", self.translator.tr('ignore_field') + " --", "")
            if target_node and target_node.entity_id:
                fields = self.wizard.db.execute_query(
                    "SELECT fieldname, calias FROM meta.fields WHERE entitytypeid=%s",
                    (target_node.entity_id,)
                )
                for fname, calias in fields:
                    combo_target.addItem(f"{calias} ({fname})", fname)
            combo_target.addItem("__NEW__", "++ " + self.translator.tr('create_new_field') + " ++", "__NEW__")
            if link['target_field']:
                idx = combo_target.findData(link['target_field'])
                if idx >= 0:
                    combo_target.setCurrentIndex(idx)
combo_target.currentIndexChanged.connect(lambda idx, r=row, cb=combo_target:
self.on_target_field_changed(r, cb))
            self.table.setCellWidget(row, 3, combo_target)
```

```
        type_combo = QComboBox()
        type_combo.addItems(["N", "I", "D", "T", "L", "M", "R"])
        type_combo.setEnabled(False)
        if link.get('target_field') and not self.field_exists(target_node, link['target_field']):
            type_combo.setEnabled(True)
        type_combo.setCurrentText(link.get('field_type', 'C'))
        self.table.setCellWidget(row, 4, type_combo)
        length_spin = QSpinBox()
        length_spin.setRange(0, 10000)
        length_spin.setValue(link.get('field_length', 255))
        length_spin.setEnabled(False)
        if link.get('target_field') and not self.field_exists(target_node, link['target_field']):
            length_spin.setEnabled(True)
        self.table.setCellWidget(row, 5, length_spin)
        chk = QCheckBox()
        chk.setChecked(link.get('is_source_id', False))
        chk.stateChanged.connect(lambda state, r=row: self.on_source_id_changed(r, state))
        self.table.setCellWidget(row, 6, chk)
        btn = QPushButton(self.translator.tr('delete_btn'))
        btn.clicked.connect(lambda checked, r=row: self.remove_link(r))
        self.table.setCellWidget(row, 7, btn)
    def field_exists(self, target_node, field_name):
        if not target_node or not target_node.entity_id:
            return False
        res = self.wizard.db.execute_query(
            "SELECT 1 FROM meta_fields WHERE entitytypeid=%s AND cfieldname=%s",
            (target_node.entity_id, field_name)
        )
        return bool(res)
    def on_target_field_changed(self, row, combo):
        if combo.currentData() == "__NEW__":
            self.create_new_field(row)
    def create_new_field(self, row):
        source_field = self.table.item(row, 1).text()
        from .wizard_dialogs import NewFieldDialog
        dlg = NewFieldDialog(self, self.wizard.db)
        # Предлагаем имя на основе исходного поля
        import re
        suggested = re.sub(r'[^a-zA-Z0-9_]', '_', source_field)
        if suggested[0].isdigit():
            suggested = 'f' + suggested
        dlg.name_edit.setText(suggested)
        if dlg.exec():
            data = dlg.get_data()
            links = getattr(self.wizard, 'links', [])
            if row < len(links):
                links[row]['target_field'] = data['cfieldname']
                links[row]['field_type'] = data['cfieldtype']
                links[row]['field_length'] = data['nlength']
            self.load_mappings()
    def on_source_id_changed(self, row, state):
        for r in range(self.table.rowCount()):
            if r != row:
                chk = self.table.cellWidget(r, 6)
                if chk:
                    chk.blockSignals(True)
                    chk.setChecked(False)
                    chk.blockSignals(False)
        links = getattr(self.wizard, 'links', [])
        if row < len(links):
            links[row]['is_source_id'] = (state == Qt.Checked)
    def remove_link(self, row):
        if hasattr(self.wizard, 'links'):
            self.wizard.links.pop(row)
        self.load_mappings()
    def apply_mapping(self):
        has_source_id = any(self.table.cellWidget(r, 6).isChecked() for r in range(self.table.rowCount()))
        if not has_source_id:
            reply = QMessageBox.question(self, self.translator.tr('warning'),
                                         self.translator.tr('no_source_id_warning'),
                                         QMessageBox.Yes | QMessageBox.No)
            if reply == QMessageBox.No:
                return
        links = getattr(self.wizard, 'links', [])
        for row in range(self.table.rowCount()):
            if row >= len(links):
                break
            combo = self.table.cellWidget(row, 3)
            target_field = combo.currentData() if combo else None
            if target_field and target_field != "__NEW__":
                links[row]['target_field'] = target_field
                links[row]['field_type'] = self.table.cellWidget(row, 4).currentText()
                links[row]['field_length'] = self.table.cellWidget(row, 5).value()
        self.wizard.import_links = links
        self.wizard.next()
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/relation_page.py (отно

```
# -*- coding: utf-8 -*-
import json
from PyQt6.QtWidgets import QWizardPage, QVBoxLayout, QPushButton, QMessageBox, QHBoxLayout, QDialog,
QVBoxLayout as QVBoxLayoutD, QTableWidget, QTableWidgetItem
from PyQt6.QtCore import Qt
from lang.local_translator import LocalTranslator
from ..graph.graphics_view import GraphicsView
from ..graph.relation_scene import RelationScene
from ..core.models import Node, NodeType, Field, DataType, LinkType
from .wizard_dialogs import NewEntityDialog

class RelationPage(QWizardPage):
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = LocalTranslator()
        self.setTitle(self.translator.tr('relation_page_title'))
        self.setSubTitle(self.translator.tr('relation_page_subtitle'))
        layout = QVBoxLayout(self)
        self.scene = RelationScene(self, relation_page=self)  # передаём себя
        self.view = GraphicsView(self.scene)
        layout.addWidget(self.view, 1)
        btn_layout = QHBoxLayout()
        self.btn_load_source = QPushButton(self.translator.tr('load_source_btn'))
        self.btn_load_source.clicked.connect(self.load_source)
        self.btn_load_mozart = QPushButton(self.translator.tr('load_mozart_btn'))
        self.btn_load_mozart.clicked.connect(self.load_existing_entities)
        self.btn_create_entity = QPushButton(self.translator.tr('create_entity_btn'))
        self.btn_create_entity.clicked.connect(self.create_entity)
        self.btn_save_mapping = QPushButton(self.translator.tr('save_mapping_btn'))
        self.btn_save_mapping.clicked.connect(self.save_mapping)
        btn_layout.addWidget(self.btn_load_source)
        btn_layout.addWidget(self.btn_load_mozart)
        btn_layout.addWidget(self.btn_create_entity)
        btn_layout.addWidget(self.btn_save_mapping)
        btn_layout.addStretch()
        layout.addLayout(btn_layout)
        self.source_nodes = []
        self.mozart_nodes = []
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('relation_page_title'))
        self.setSubTitle(self.translator.tr('relation_page_subtitle'))
        self.btn_load_source.setText(self.translator.tr('load_source_btn'))
        self.btn_load_mozart.setText(self.translator.tr('load_mozart_btn'))
        self.btn_create_entity.setText(self.translator.tr('create_entity_btn'))
        self.btn_save_mapping.setText(self.translator.tr('save_mapping_btn'))
    def clear_scene(self):
        self.scene.clear()
        self.source_nodes = []
        self.mozart_nodes = []
    def load_source(self):
        if not self.wizard.connector:
```

```
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('connector_not_ready'))
            return
        self.clear_scene()
        schema = self.wizard.connector.get_schema()
        if not schema.tables:
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('no_tables_found'))
            return
        x = 30
        for table in schema.tables:
            fields = []
            for f in table.fields:
                dt = DataType.STRING
                if f.field_type.name == 'INTEGER':
                    dt = DataType.INTEGER
                elif f.field_type.name == 'FLOAT':
                    dt = DataType.FLOAT
                elif f.field_type.name == 'BOOLEAN':
                    dt = DataType.BOOLEAN
                elif f.field_type.name == 'DATE':
                    dt = DataType.DATE
                elif f.field_type.name == 'DATETIME':
                    dt = DataType.DATETIME
                fields.append(Field(
                    name=f.name,
                    data_type=dt,
                    is_primary_key=f.is_primary_key,
                    is_foreign_key=f.is_foreign_key,
                    references=f.refers_to
                ))
            node = Node(
                id=f"src_{table.name}",
                name=table.name,
                node_type=NodeType.SOURCE_TABLE,
                fields=fields,
                x=x, y=y,
                table_name=table.name
            )
            self.scene.add_node(node)
            self.source_nodes.append(node)
            x += 250
            if x > 500:
                x = 30
                y += 250
        # Автоматические связи FK
        for node in self.source_nodes:
            for field in node.fields:
                if field.is_foreign_key and field.references:
                    target_table_name = field.references
                    for target_node in self.source_nodes:
                        if target_node.table_name == target_table_name:
                            target_field = None
                            for tf in target_node.fields:
                                if tf.is_primary_key:
                                    target_field = tf
                                    break
                            if target_field:
                                start_fi = self.find_field_item(node, field.name)
                                end_fi = self.find_field_item(target_node, target_field.name)
                                if start_fi and end_fi:
                                    self.scene.add_link(start_fi, end_fi, LinkType.FK_SOURCE)
QMessageBox.information(self, self.translator.tr('info'),
self.translator.tr('source_loaded').format(len(schema.tables)))
    def find_field_item(self, node_item, field_name):
        for fi in node_item.field_items:
            if fi.field.name == field_name:
                return fi
        return None
    def load_existing_entities(self):
        if not self.wizard.db:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('db_not_ready'))
            return
rows = self.wizard.db.execute_query("SELECT id, cname, calias, class_id FROM meta.entitytypes ORDER BY
cname")
        x = 600
        for eid, cname, calias, class_id in rows:
class_res = self.wizard.db.execute_query("SELECT calias FROM meta.entity_classes WHERE id=%s", (class_id,))
            class_alias = class_res[0][0] if class_res else "DICT"
            fields = []
frows = self.wizard.db.execute_query("SELECT cfieldname, cfieldtype FROM meta.fields WHERE
entitytypeid=%s", (eid,))
            for fname, ftype in frows:
                dt = DataType.STRING
                if ftype == 'INTEGER':
                    dt = DataType.INTEGER
                elif ftype == 'FLOAT':
                    dt = DataType.FLOAT
                elif ftype == 'BOOLEAN':
                    dt = DataType.BOOLEAN
                elif ftype == 'DATE':
                    dt = DataType.DATE
                elif ftype == 'DATETIME':
                    dt = DataType.DATETIME
                fields.append(Field(name=fname, data_type=dt))
            node = Node(
                id=f"moz_{eid}",
                name=f"{cname} ({calias})",
                node_type=NodeType.MOZART_ENTITY,
                fields=fields,
                x=x, y=y,
                entity_id=eid,
                class_alias=class_alias
            )
            self.scene.add_node(node)
            self.mozart_nodes.append(node)
            x += 250
            if x > 1200:
                x = 600
                y += 250
        # Связи композиции
comps = self.wizard.db.execute_query("SELECT parent_entityid, child_entityid, c_link_fieldname FROM
meta.entity_composition")
        for parent_id, child_id, link_field in comps:
            parent_node = None
            child_node = None
            for n in self.mozart_nodes:
                if n.entity_id == parent_id:
                    parent_node = n
                if n.entity_id == child_id:
                    child_node = n
            if parent_node and child_node:
                # Ищем поле в дочерней сущности с именем link_field
                child_field = None
                for f in child_node.fields:
                    if f.name == link_field:
                        child_field = f
                        break
                if child_field:
                    # Для родителя используем первое поле (или можно создать линию без привязки к полю)
                    parent_field = parent_node.fields[0] if parent_node.fields else None
                    if parent_field:
                        start_fi = self.find_field_item(parent_node, parent_field.name)
                        end_fi = self.find_field_item(child_node, child_field.name)
                        if start_fi and end_fi:
                            self.scene.add_link(start_fi, end_fi, LinkType.COMPOSITION)
QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('mozart_loaded').format(len(rows)))
    def create_entity(self):
        dlg = NewEntityDialog(self, self.wizard.db)
        if dlg.exec():
            data = dlg.get_data()
            node = Node(
                id=f"new_{data['calias']}",
                name=data['cname'],
                node_type=NodeType.MOZART_ENTITY,
                fields=[],
                x=600, y=300,
                class_alias="DICT"
            )
            self.scene.add_node(node)
            if not hasattr(self.wizard, 'pending_entities'):
                self.wizard.pending_entities = []
            self.wizard.pending_entities.append((node, data))
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('entity_added_to_scene'))
    def save_mapping(self):
```

```
        nodes_pos = {}
        for node_item in self.scene.nodes.values():
            nodes_pos[node_item.node.id] = {'x': node_item.node.x, 'y': node_item.node.y}
        links_data = []
        for link_line in self.scene.links:
            link_dict = {
                'source_node_id': link_line.start_item.node.id,
                'source_field': link_line.start_field_item.field.name,
                'target_node_id': link_line.end_item.node.id,
                'target_field': link_line.end_field_item.field.name,
                'link_type': link_line.link_type.value,
                'is_source_id': getattr(link_line, 'is_source_id', False)
            }
            links_data.append(link_dict)
        scene_data = {
            'nodes_pos': nodes_pos,
            'links': links_data
        }
        self.wizard.import_scene_json = json.dumps(scene_data, ensure_ascii=False)
        self.wizard.source_nodes = self.source_nodes
        self.wizard.mozart_nodes = self.mozart_nodes
        self.wizard.links = links_data
        self.wizard.next()
    def preview_table(self, node_item):
        if not self.wizard.connector:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('connector_not_ready'))
            return
        table_name = node_item.node.table_name
        sample = self.wizard.connector.get_table_sample(table_name, limit=20)
        if not sample:
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('no_data'))
            return
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('preview_table_title').format(table_name))
        layout = QVBoxLayout(dlg)
        table = QTableWidget()
        if sample:
            headers = list(sample[0].keys())
            table.setColumnCount(len(headers))
            table.setHorizontalHeaderLabels(headers)
            table.setRowCount(len(sample))
            for r, row in enumerate(sample):
                for c, col in enumerate(headers):
                    table.setItem(r, c, QTableWidgetItem(str(row.get(col, ''))))
        layout.addWidget(table)
        dlg.exec_()
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/schema_page.py (отно

```
# mozart_import/wizards/schema_page.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QWizardPage, QVBoxLayout, QLabel, QComboBox, QTableWidget,
QTableWidgetItem, QHeaderView
from lang.local_translator import LocalTranslator

class SchemaPage(QWizardPage):
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = LocalTranslator()
        self.setTitle(self.translator.tr('schema_page_title'))
        self.setSubTitle(self.translator.tr('schema_page_subtitle'))
        layout = QVBoxLayout(self)
        self.table_combo = QComboBox()
        self.table_combo.currentIndexChanged.connect(self.table_selected)
        layout.addWidget(QLabel(self.translator.tr('table_label')))
        layout.addWidget(self.table_combo)
        self.info_label = QLabel("")
        layout.addWidget(self.info_label)
        self.fields_table = QTableWidget(0, 2)
        self.fields_table.setHorizontalHeaderLabels(
            [self.translator.tr('field_name_header'), self.translator.tr('field_type_header')])
        self.fields_table.horizontalHeader().setStretchLastSection(True)
        layout.addWidget(self.fields_table)
        self.loading = QLabel(self.translator.tr('loading_message'))
        layout.addWidget(self.loading)
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('schema_page_title'))
        self.setSubTitle(self.translator.tr('schema_page_subtitle'))
        self.fields_table.setHorizontalHeaderLabels(
            [self.translator.tr('field_name_header'), self.translator.tr('field_type_header')])
    def initializePage(self):
        self.loading.show()
        self.wizard.schema = self.wizard.connector.get_schema()
        self.table_combo.clear()
        for t in self.wizard.schema.tables:
            self.table_combo.addItem(t.name, t.name)
        if self.wizard.schema.tables:
            self.table_selected(0)
        self.loading.hide()
    def table_selected(self, idx):
        table_name = self.table_combo.currentData()
        if not table_name:
            return
        for t in self.wizard.schema.tables:
            if t.name == table_name:
                self.current_table = t
                self.info_label.setText(
f"{self.translator.tr('rows_count_label')}: {t.row_count or self.translator.tr('unknown')},
{self.translator.tr('samples_label')}: {len(t.sample_rows or [])}")
                self.fields_table.setRowCount(len(t.fields))
                for r, f in enumerate(t.fields):
                    self.fields_table.setItem(r, 0, QTableWidgetItem(f.name))
                    self.fields_table.setItem(r, 1, QTableWidgetItem(f.field_type.name))
                break
    def validatePage(self):
        self.wizard.selected_table_name = self.table_combo.currentData()
        return True
# -*- coding: utf-8 -*-
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/source_page.py (относ

```
# -*- coding: utf-8 -*-
# mozart_import/wizards/source_page.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QWizardPage, QVBoxLayout, QHBoxLayout, QLabel, QComboBox, QLineEdit,
QPushButton, QFileDialog, QGroupBox, QMessageBox
from lang.local_translator import LocalTranslator
from core.models import SourceType, FileFormat
from ::connectors.file_connector import FileConnector
from ::connectors.db_connector import DBConnector
from ::connectors.url_connector import URLConnector

class SourcePage(QWizardPage):
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = LocalTranslator()
        self.setTitle(self.translator.tr('source_page_title'))
        self.setSubTitle(self.translator.tr('source_page_subtitle'))
        layout = QVBoxLayout(self)
        self.type_combo = QComboBox()
        self.type_combo.addItem(self.translator.tr('source_type_file'), SourceType.FILE)
        self.type_combo.addItem(self.translator.tr('source_type_db'), SourceType.DATABASE)
        self.type_combo.addItem(self.translator.tr('source_type_url'), SourceType.URL)
        self.type_combo.currentIndexChanged.connect(self.toggle_groups)
        layout.addWidget(QLabel(self.translator.tr('source_type_label')))
        layout.addWidget(self.type_combo)
        # Файл
        self.file_group = QGroupBox(self.translator.tr('file_group'))
```

```
        file_layout = QVBoxLayout(self.file_group)
        self.format_combo = QComboBox()
        for fmt in [FileFormat.XML, FileFormat.DBF, FileFormat.CSV, FileFormat.EXCEL]:
            self.format_combo.addItem(fmt.value, fmt)
        file_layout.addWidget(QLabel(self.translator.tr('file_format_label')))
        file_layout.addWidget(self.format_combo)
        self.path_edit = QLineEdit()
        self.browse_btn = QPushButton(self.translator.tr('browse_button'))
        self.browse_btn.clicked.connect(self.browse_file)
        path_layout = QHBoxLayout()
        path_layout.addWidget(self.path_edit)
        path_layout.addWidget(self.browse_btn)
        file_layout.addLayout(path_layout)
        layout.addWidget(self.file_group)
        # База данных
        self.db_group = QGroupBox(self.translator.tr('db_group'))
        db_layout = QVBoxLayout(self.db_group)
        self.db_type = QComboBox()
        self.db_type.addItems(["postgresql", "mysql", "mssql", "oracle"])
        db_layout.addWidget(QLabel(self.translator.tr('db_type_label')))
        db_layout.addWidget(self.db_type)
        self.db_host = QLineEdit("localhost")
        self.db_port = QLineEdit()
        self.db_name = QLineEdit()
        self.db_user = QLineEdit()
        self.db_pass = QLineEdit()
        self.db_pass.setEchoMode(QLineEdit.Password)
        db_layout.addWidget(QLabel(self.translator.tr('db_host_label')))
        db_layout.addWidget(self.db_host)
        db_layout.addWidget(QLabel(self.translator.tr('db_port_label')))
        db_layout.addWidget(self.db_port)
        db_layout.addWidget(QLabel(self.translator.tr('db_name_label')))
        db_layout.addWidget(self.db_name)
        db_layout.addWidget(QLabel(self.translator.tr('db_user_label')))
        db_layout.addWidget(self.db_user)
        db_layout.addWidget(QLabel(self.translator.tr('db_pass_label')))
        db_layout.addWidget(self.db_pass)
        layout.addWidget(self.db_group)
        # URL
        self.url_group = QGroupBox(self.translator.tr('url_group'))
        url_layout = QVBoxLayout(self.url_group)
        self.url_edit = QLineEdit()
        url_layout.addWidget(QLabel(self.translator.tr('url_label')))
        url_layout.addWidget(self.url_edit)
        self.method_combo = QComboBox()
        self.method_combo.addItems(["GET", "POST"])
        url_layout.addWidget(QLabel(self.translator.tr('http_method_label')))
        url_layout.addWidget(self.method_combo)
        layout.addWidget(self.url_group)
        layout.addStretch()
        self.toggle_groups()
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('source_page_title'))
        self.setSubTitle(self.translator.tr('source_page_subtitle'))
        self.type_combo.setItemText(0, self.translator.tr('source_type_file'))
        self.type_combo.setItemText(1, self.translator.tr('source_type_db'))
        self.type_combo.setItemText(2, self.translator.tr('source_type_url'))
        self.file_group.setTitle(self.translator.tr('file_group'))
        self.browse_btn.setText(self.translator.tr('browse_button'))
        self.db_group.setTitle(self.translator.tr('db_group'))
        self.url_group.setTitle(self.translator.tr('url_group'))
    def toggle_groups(self):
        st = self.type_combo.currentData()
        self.file_group.setVisible(st == SourceType.FILE)
        self.db_group.setVisible(st == SourceType.DATABASE)
        self.url_group.setVisible(st == SourceType.URL)
    def browse_file(self):
        path = QFileDialog.getOpenFileName(self, self.translator.tr('select_file_dialog'))
        if path:
            self.path_edit.setText(path)
    def validatePage(self):
        st = self.type_combo.currentData()
        try:
            if st == SourceType.FILE:
                self.wizard.connector = FileConnector(self.path_edit.text(), self.format_combo.currentData())
            elif st == SourceType.DATABASE:
                params = {
                    'db_type': self.db_type.currentText(),
                    'host': self.db_host.text(),
                    'port': self.db_port.text() or None,
                    'database': self.db_name.text(),
                    'user': self.db_user.text(),
                    'password': self.db_pass.text()
                }
                self.wizard.connector = DBConnector("", **params)
            else:
                self.wizard.connector = URLConnector(self.url_edit.text())
            self.wizard.connector.connect()
            return True
        except Exception as e:
            QMessageBox.critical(self, self.translator.tr('error'), str(e))
            return False
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/wizard_dialogs.py (отно

```
# mozart_import/wizards/wizard_dialogs.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QFormLayout, QLineEdit, QComboBox, QSpinBox,
QDialogButtonBox
from lang.local_translator import LocalTranslator

class NewFieldDialog(QDialog):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('new_field_dialog_title'))
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.name_edit = QLineEdit()
        form.addRow(self.translator.tr('field_name_latin'), self.name_edit)
        self.alias_edit = QLineEdit()
        form.addRow(self.translator.tr('field_alias'), self.alias_edit)
        self.type_combo = QComboBox()
        self.type_combo.addItems(["N", "I", "D", "T", "L", "M", "R"])
        form.addRow(self.translator.tr('mozart_type'), self.type_combo)
        self.length_edit = QSpinBox()
        self.length_edit.setRange(0, 10000)
        self.length_edit.setValue(255)
        form.addRow(self.translator.tr('length_for_string'), self.length_edit)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self.accept)
        btn_box.rejected.connect(self.reject)
        self.name_edit.textChanged.connect(lambda: self.name_edit.setText(self.name_edit.text()[:63]))
        self.alias_edit.textChanged.connect(lambda: self.alias_edit.setText(self.alias_edit.text()[:50]))
        layout.addWidget(btn_box)
    def get_data(self):
        return {
            'cfieldname': self.name_edit.text(),
            'calias': self.alias_edit.text() or self.name_edit.text(),
            'cfieldtype': self.type_combo.currentText(),
            'nlength': self.length_edit.value()
        }

class NewEntityDialog(QDialog):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db
        self.translator = LocalTranslator()
```

```
        self.setWindowTitle(self.translator.tr('create_entity_dialog_title'))
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.name_edit = QLineEdit()
        form.addRow(self.translator.tr('entity_name_cname'), self.name_edit)
        self.alias_edit = QLineEdit()
        form.addRow(self.translator.tr('entity_alias_calias'), self.alias_edit)
        self.class_combo = QComboBox()
        classes = self.db.execute_query("SELECT id, cname FROM meta.entity_classes ORDER BY cname")
        for cid, cname in classes:
            self.class_combo.addItem(cname, cid)
        form.addRow(self.translator.tr('entity_class'), self.class_combo)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self.accept)
        btn_box.rejected.connect(self.reject)
        layout.addWidget(btn_box)
    def get_data(self):
        return {
            'cname': self.name_edit.text(),
            'calias': self.alias_edit.text(),
            'class_id': self.class_combo.currentData()
        }
```

Файл: /home/sergey/Documents/configurate/mozart_import/wizards/wizard_pages.py (отно

```
# mozart_import/wizards/wizard_pages.py (полностью, с новым MappingPage и предпросмотром)
from PyQt5.QtWidgets import QWizardPage, QVBoxLayout, QHBoxLayout, QLabel, QComboBox,
                            QLineEdit, QPushButton, QFileDialog, QTableWidget, QTableWidgetItem,
                            QHeaderView, QProgressBar, QMessageBox, QGroupBox, QTextEdit,
                            QCheckBox, QSpinBox, QFrame, QTabWidget, QWidget)
from PyQt5.QtCore import Qt, QVariant
from lang.local_translator import LocalTranslator
from database.import_database_service
from PyQt5.QtWidgets import QStyledItemDelegate
from ..core.models.import_source_type, FileFormat, MatchedField, MatchedEntity, MappingProposal
from ..connectors.file_connector.import_file_connector
from ..connectors.db_connector.import_db_connector
from ..connectors.http_connector.import_http_connector
from ..utils.field_creator import FieldCreator
class ComboBoxDelegate(QStyledItemDelegate):
    pass  # не используется, т.к. мы вставляем виджеты напрямую
class SourcePage(QWizardPage):
    # (без изменений, как в предыдущей версии)
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = wizard.translator
        self.setTitle(self.translator.tr('source_page_title'))
        self.setSubTitle(self.translator.tr('source_page_subtitle'))
        layout = QVBoxLayout(self)
        self.type_combo = QComboBox()
        self.type_combo.addItem(self.translator.tr('source_type_file'), SourceType.FILE)
        self.type_combo.addItem(self.translator.tr('source_type_db'), SourceType.DATABASE)
        self.type_combo.addItem(self.translator.tr('source_type_url'), SourceType.URL)
        self.type_combo.currentIndexChanged.connect(self.toggle_groups)
        layout.addWidget(QLabel(self.translator.tr('source_type_label')))
        layout.addWidget(self.type_combo)
        self.file_group = QGroupBox(self.translator.tr('file_group'))
        file_layout = QVBoxLayout(self.file_group)
        self.format_combo = QComboBox()
        for fmt in [FileFormat.JSON, FileFormat.XML, FileFormat.DBF, FileFormat.CSV, FileFormat.EXCEL]:
            self.format_combo.addItem(fmt.value, fmt)
        file_layout.addWidget(QLabel(self.translator.tr('file_format_label')))
        file_layout.addWidget(self.format_combo)
        self.path_edit = QLineEdit()
        self.browse_btn = QPushButton(self.translator.tr('browse_button'))
        self.browse_btn.clicked.connect(self.browse_file)
        path_layout = QHBoxLayout()
        path_layout.addWidget(self.path_edit)
        path_layout.addWidget(self.browse_btn)
        file_layout.addLayout(path_layout)
        layout.addWidget(self.file_group)
        self.db_group = QGroupBox(self.translator.tr('db_group'))
        db_layout = QVBoxLayout(self.db_group)
        self.db_type = QComboBox()
        self.db_type.addItems(["postgresql", "mysql", "mssql", "oracle"])
        db_layout.addWidget(QLabel(self.translator.tr('db_type_label')))
        db_layout.addWidget(self.db_type)
        self.db_host = QLineEdit("localhost")
        self.db_port = QLineEdit()
        self.db_name = QLineEdit()
        self.db_user = QLineEdit()
        self.db_pass = QLineEdit()
        self.db_pass.setEchoMode(QLineEdit.Password)
        db_layout.addWidget(QLabel(self.translator.tr('db_host_label')))
        db_layout.addWidget(self.db_host)
        db_layout.addWidget(QLabel(self.translator.tr('db_port_label')))
        db_layout.addWidget(self.db_port)
        db_layout.addWidget(QLabel(self.translator.tr('db_name_label')))
        db_layout.addWidget(self.db_name)
        db_layout.addWidget(QLabel(self.translator.tr('db_user_label')))
        db_layout.addWidget(self.db_user)
        db_layout.addWidget(QLabel(self.translator.tr('db_pass_label')))
        db_layout.addWidget(self.db_pass)
        layout.addWidget(self.db_group)
        self.url_group = QGroupBox(self.translator.tr('url_group'))
        url_layout = QVBoxLayout(self.url_group)
        self.url_edit = QLineEdit()
        url_layout.addWidget(QLabel(self.translator.tr('url_label')))
        url_layout.addWidget(self.url_edit)
        self.method_combo = QComboBox()
        self.method_combo.addItems(["GET", "POST"])
        url_layout.addWidget(QLabel(self.translator.tr('http_method_label')))
        url_layout.addWidget(self.method_combo)
        layout.addWidget(self.url_group)
        layout.addStretch()
        self.toggle_groups()
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('source_page_title'))
        self.setSubTitle(self.translator.tr('source_page_subtitle'))
        self.type_combo.setItemText(0, self.translator.tr('source_type_file'))
        self.type_combo.setItemText(1, self.translator.tr('source_type_db'))
        self.type_combo.setItemText(2, self.translator.tr('source_type_url'))
        self.file_group.setTitle(self.translator.tr('file_group'))
        self.browse_btn.setText(self.translator.tr('browse_button'))
        self.db_group.setTitle(self.translator.tr('db_group'))
        self.url_group.setTitle(self.translator.tr('url_group'))
    def toggle_groups(self):
        st = self.type_combo.currentData()
        self.file_group.setVisible(st == SourceType.FILE)
        self.db_group.setVisible(st == SourceType.DATABASE)
        self.url_group.setVisible(st == SourceType.URL)
    def browse_file(self):
        path = QFileDialog.getOpenFileName(self, self.translator.tr('select_file_dialog'))
        if path:
            self.path_edit.setText(path)
    def validatePage(self):
        st = self.type_combo.currentData()
        try:
            if st == SourceType.FILE:
                self.wizard.connector = FileConnector(self.path_edit.text(), self.format_combo.currentData())
            elif st == SourceType.DATABASE:
                params = {
                    'db_type': self.db_type.currentText(),
                    'host': self.db_host.text(),
                    'port': self.db_port.text() or None,
                    'database': self.db_name.text(),
                    'user': self.db_user.text(),
                    'password': self.db_pass.text()
```

```
            }
            self.wizard.connector = DBConnector("", **params)
        else:
            self.wizard.connector = URLConnector(self.url_edit.text())
        self.wizard.connector.connect()
        return True
    except Exception as e:
        QMessageBox.critical(self, self.translator.tr('error'), str(e))
        return False

class SchemaPage(QWizardPage):
    # ... (без изменений)
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = wizard.translator
        self.setTitle(self.translator.tr('schema_page_title'))
        self.setSubTitle(self.translator.tr('schema_page_subtitle'))
        layout = QVBoxLayout(self)
        self.table_combo = QComboBox()
        self.table_combo.currentIndexChanged.connect(self.table_selected)
        layout.addWidget(QLabel(self.translator.tr('table_label')))
        layout.addWidget(self.table_combo)
        self.info_label = QLabel("")
        layout.addWidget(self.info_label)
        self.fields_table = QTableWidget(0, 2)
self.fields_table.setHorizontalHeaderLabels([self.translator.tr('field_name_header'),
self.translator.tr('field_type_header')])
        layout.addWidget(self.fields_table)
        self.loading = QLabel(self.translator.tr('loading_message'))
        layout.addWidget(self.loading)
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('schema_page_title'))
        self.setSubTitle(self.translator.tr('schema_page_subtitle'))
self.fields_table.setHorizontalHeaderLabels([self.translator.tr('field_name_header'),
self.translator.tr('field_type_header')])
    def initializePage(self):
        self.loading.show()
        self.wizard.schema = self.wizard.connector.get_schema()
        self.table_combo.clear()
        for t in self.wizard.schema.tables:
            self.table_combo.addItem(t.name, t.name)
        if self.wizard.schema.tables:
            self.table_selected(0)
        self.loading.hide()
    def table_selected(self, idx):
        table_name = self.table_combo.currentData()
        if not table_name:
            return
        for t in self.wizard.schema.tables:
            if t.name == table_name:
                self.info_label.setText(f"{self.translator.tr('rows_count_label')}: {t.row_count or self.translator.tr('unknown')},
{self.translator.tr('samples_label')}: {len(t.sample_rows or [])}")
                self.fields_table.setRowCount(len(t.fields))
                for i, f in enumerate(t.fields):
                    self.fields_table.setItem(i, 0, QTableWidgetItem(f.name))
                    self.fields_table.setItem(i, 1, QTableWidgetItem(f.field_type.name))
                break
    def validatePage(self):
        self.wizard.selected_table_name = self.table_combo.currentData()
        return True

class MappingPage(QWizardPage):
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = wizard.translator
        self.setTitle(self.translator.tr('mapping_page_title'))
        self.setSubTitle(self.translator.tr('mapping_page_subtitle'))
        self.current_entity_id = None
        self.pending_fields = {}  # {row: {name, type, length, alias, ref}}
        self.init_ui()
    def init_ui(self):
        layout = QVBoxLayout(self)
        # Верхняя панель: выбор/создание сущности
        top_layout = QHBoxLayout()
        self.entity_combo = QComboBox()
        self.entity_combo.setEditable(True)
        self.entity_combo.addItem(self.translator.tr('create_new_entity'), None)
        self.entity_combo.currentIndexChanged.connect(self.on_entity_selected)
        top_layout.addWidget(QLabel(self.translator.tr('target_entity_label')))
        top_layout.addWidget(self.entity_combo)
        self.btn_create_entity = QPushButton(self.translator.tr('create_entity_button'))
        self.btn_create_entity.clicked.connect(self.create_new_entity)
        top_layout.addWidget(self.btn_create_entity)
        layout.addLayout(top_layout)
        # Таблица маппинга (исходные поля -> целевые поля)
        self.mapping_table = QTableWidget(0, 5)
        self.mapping_table.setHorizontalHeaderLabels([
            self.translator.tr('source_field_header'),
            self.translator.tr('target_field_header'),
            self.translator.tr('type_header'),
            self.translator.tr('length_header'),
            self.translator.tr('ref_header')
        ])
        self.mapping_table.horizontalHeader().setStretchLastSection(True)
        self.mapping_table.setItemDelegateForColumn(1, ComboBoxDelegate(self))  # Кастомный делегат
        layout.addWidget(self.mapping_table)
        # Кнопки действий
        btn_layout = QHBoxLayout()
        self.btn_apply_structure = QPushButton(self.translator.tr('apply_structure_button'))
        self.btn_apply_structure.clicked.connect(self.apply_structure)
        self.btn_import_data = QPushButton(self.translator.tr('import_data_button'))
        self.btn_import_data.clicked.connect(self.import_data)
        self.btn_import_data.setEnabled(False)
        btn_layout.addWidget(self.btn_apply_structure)
        btn_layout.addWidget(self.btn_import_data)
        layout.addLayout(btn_layout)
        # Вкладка предпросмотра данных
        self.preview_tab = QTabWidget()
        self.preview_table = QTableWidget()
        self.preview_tab.addTab(self.preview_table, self.translator.tr('data_preview_tab'))
        layout.addWidget(self.preview_tab)
        self.loading = QLabel(self.translator.tr('loading_message'))
        layout.addWidget(self.loading)
        self.refresh_entity_list()
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('mapping_page_title'))
        self.setSubTitle(self.translator.tr('mapping_page_subtitle'))
        self.entity_combo.setItemText(0, self.translator.tr('create_new_entity'))
        self.btn_create_entity.setText(self.translator.tr('create_entity_button'))
        self.btn_apply_structure.setText(self.translator.tr('apply_structure_button'))
        self.btn_import_data.setText(self.translator.tr('import_data_button'))
        self.preview_tab.setTabText(0, self.translator.tr('data_preview_tab'))
        self.mapping_table.setHorizontalHeaderLabels([
            self.translator.tr('source_field_header'),
            self.translator.tr('target_field_header'),
            self.translator.tr('type_header'),
            self.translator.tr('length_header'),
            self.translator.tr('ref_header')
        ])
    def refresh_entity_list(self):
        self.entity_combo.clear()
        self.entity_combo.addItem(self.translator.tr('create_new_entity'), None)
        rows = self.wizard.db.execute_query("SELECT id, cname, calias FROM meta.entitytypes ORDER BY cname")
        for eid, cname, calias in rows:
            self.entity_combo.addItem(f"{cname} ({calias})", eid)
    def on_entity_selected(self):
        self.current_entity_id = self.entity_combo.currentData()
        self.load_mapping_table()
```

```
    def load_mapping_table(self):
        if not self.wizard.schema:
            return
        source_table = None
        for t in self.wizard.schema.tables:
            if t.name == self.wizard.selected_table_name:
                source_table = t
                break
        if not source_table:
            self.mapping_table.setRowCount(0)
            return
        self.mapping_table.setRowCount(len(source_table.fields))
        for row, sf in enumerate(source_table.fields):
            self.mapping_table.setItem(row, 0, QTableWidgetItem(sf.name))
            # В колонке 1 — комбобокс с существующими полями + опция создания нового
            combo = QComboBox()
            combo.addItem("-- " + self.translator.tr('ignore_field') + " --", "")
            if self.current_entity_id:
                fields = self.wizard.db.execute_query(
                    "SELECT fieldname, callas FROM meta.fields WHERE entitytypeid=%s",
                    (self.current_entity_id,)
                )
                for fname, callas in fields:
                    combo.addItem(f"{callas} ({fname})", fname)
            combo.addItem("++ " + self.translator.tr('create_new_field') + " ++", "__NEW__")
            combo.currentIndexChanged.connect(lambda idx, r=row: self.on_new_field_selected(r, combo))
            self.mapping_table.setCellWidget(row, 1, combo)
            self.mapping_table.setItem(row, 2, QTableWidgetItem(sf.field_type.name))
            self.mapping_table.setItem(row, 3, QTableWidgetItem(""))
            self.mapping_table.setItem(row, 4, QTableWidgetItem(""))
            # Если есть pending поля для этой строки, восстановить
            if row in self.pending_fields:
                self.show_pending_editor(row, self.pending_fields[row])
        # Загрузить предпросмотр данных
        self.load_preview(source_table)
    def on_new_field_selected(self, row, combo):
        if combo.currentData() == "__NEW__":
            combo.blockSignals(True)
            combo.setCurrentIndex(0)
            combo.blockSignals(False)
            # Показываем редактор в строке
            self.show_pending_editor(row)
    def show_pending_editor(self, row, pending=None):
        # Создаем виджеты для ввода параметров нового поля
        # Временно сохраним в self.pending_fields и отобразим в дополнительных колонках
        # Для простоты используем QLineEdit в ячейках колонок 2-4
        name_edit = QLineEdit()
        if pending and pending.get('name'):
            name_edit.setText(pending['name'])
        else:
            source_field = self.mapping_table.item(row, 0).text()
            name_edit.setText(self.to_field_name(source_field))
        name_edit.textChanged.connect(lambda t, r=row: self.update_pending(r, 'name', t))
        self.mapping_table.setCellWidget(row, 2, name_edit)
        type_combo = QComboBox()
        type_combo.addItems(["I", "N", "J", "D", "T", "L", "M", "R"])
        if pending and pending.get('type'):
            idx = type_combo.findText(pending['type'])
            if idx >= 0:
                type_combo.setCurrentIndex(idx)
type_combo.currentIndexChanged.connect(lambda idx, r=row: self.update_pending(r, 'type',
type_combo.currentText()))
        self.mapping_table.setCellWidget(row, 3, type_combo)
        length_spin = QSpinBox()
        length_spin.setRange(1, 10000)
        length_spin.setValue(pending.get('length', 255) if pending else 255)
        length_spin.valueChanged.connect(lambda v, r=row: self.update_pending(r, 'length', v))
        self.mapping_table.setCellWidget(row, 4, length_spin)
        # Сохраняем pending
        self.pending_fields[row] = {
            'name': name_edit.text(),
            'type': type_combo.currentText(),
            'length': length_spin.value(),
            'alias': name_edit.text()
        }
    def update_pending(self, row, key, value):
        if row not in self.pending_fields:
            self.pending_fields[row] = {}
        self.pending_fields[row][key] = value
    def to_field_name(self, source_field):
        # Преобразует имя исходного поля в допустимое имя поля MySQL/CamelCase
        import re
        name = re.sub(r'[^a-zA-Z0-9_]', '_', source_field)
        if name[0].isdigit():
            name = 'f' + name
        return name.lower()
    def apply_structure(self):
        if not self.current_entity_id:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('select_entity_first'))
            return
        # Собираем все новые поля из pending_fields
        fields_to_create = []
        for row, pending in self.pending_fields.items():
            if not pending.get('name'):
                continue
            fields_to_create.append({
                'fieldname': pending['name'],
                'fieldtype': pending.get('type', 'C'),
                'callas': pending.get('alias', pending['name']),
                'length': pending.get('length', 255),
                'isindexed': True
            })
        if not fields_to_create:
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('no_fields_to_create'))
            return
        # Транзакция через FieldCreator
        creator = FieldCreator(self.wizard.db)
        try:
            created_ids = creator.create_fields(self.current_entity_id, fields_to_create)
QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('structure_created', len(created_ids)))
            self.btn_import_data.setEnabled(True)
            # Обновить таблицу маппинга: убрать редакторы, загрузить существующие поля
            self.load_mapping_table()
        except Exception as e:
            QMessageBox.critical(self, self.translator.tr('error'), str(e))
    def import_data(self):
        # Собрать маппинг для импорта
        matched_fields = []
        for row in range(self.mapping_table.rowCount()):
            source_field = self.mapping_table.item(row, 0).text()
            combo = self.mapping_table.cellWidget(row, 1)
            target_field = combo.currentData()
            if target_field and target_field != "__NEW__":
                matched_fields.append(MatchedField(
                    source_field=source_field,
                    target_field=target_field,
                    confidence=1.0,
                    is_reference=False
                ))
        # Создаём MatchedEntity
        matched_entity = MatchedEntity(
            source_table=self.wizard.selected_table_name,
            target_entity_id=self.current_entity_id,
            target_entity_name="",
            confidence=1.0,
            fields=matched_fields,
            is_new=False
        )
        self.wizard.matching_proposal = MappingProposal(
            source_schema=self.wizard.schema,
            entities=[matched_entity],
            unresolved_tables=[],
            suggestions_for_new_entities=[]
        )
        # Переходим на страницу импорта (ExecutePage)
        self.wizard.next()
    def load_preview(self, source_table):
        if not source_table.sample_rows:
            self.preview_table.setRowCount(0)
```

```
            self.preview_table.setColumnCount(0)
            return
        sample = source_table.sample_rows
        if not sample:
            return
        headers = list(sample[0].keys())
        self.preview_table.setColumnCount(len(headers))
        self.preview_table.setHorizontalHeaderLabels(headers)
        self.preview_table.setRowCount(len(sample))
        for r, row in enumerate(sample):
            for c, col in enumerate(headers):
                val = row.get(col, '')
                self.preview_table.setItem(r, c, QTableWidgetItem(str(val)))
        self.preview_table.resizeColumnsToContents()
    def create_new_entity(self):
        from ui.dialogs import NewEntityDialog
        dlg = NewEntityDialog(self, self.wizard.db)
        if dlg.exec():
            data = dlg.get_data()
            res = self.wizard.db.execute_query(
"""INSERT INTO meta_entitytypes (cname, calias, class_id, chaseclass, lisversioned, isecuritystrategy, status_id)
                VALUES (%s, %s, %s, true, 1, 1) RETURNING id""",
                (data['cname'], data['calias'], data['class_id'])
            )
            if res:
                entity_id = res[0][0]
                QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('entity_created'))
                self.refresh_entity_list()
                idx = self.entity_combo.findData(entity_id)
                if idx >= 0:
                    self.entity_combo.setCurrentIndex(idx)
                self.current_entity_id = entity_id
                self.load_mapping_table()

class ExecutePage(QWizardPage):
    # ... (без изменений, как в предыдущей версии)
    def __init__(self, wizard):
        super().__init__(wizard)
        self.wizard = wizard
        self.translator = wizard.translator
        self.setTitle(self.translator.tr('execute_page_title'))
        self.setSubTitle(self.translator.tr('execute_page_subtitle'))
        layout = QVBoxLayout(self)
        self.progress = QProgressBar()
        layout.addWidget(self.progress)
        self.log = QTextEdit()
        self.log.setReadOnly(True)
        layout.addWidget(self.log)
        self.import_thread = None
    def retranslate_ui(self):
        self.setTitle(self.translator.tr('execute_page_title'))
        self.setSubTitle(self.translator.tr('execute_page_subtitle'))
    def initializePage(self):
        from ..core.models import ImportProfile
        from ..core.profile_manager import ProfileManager
        profile = ImportProfile(
            name=f"Import {self.wizard.connector.source_path}",
            source_type=self.wizard.connector.source_type,
            source_path=self.wizard.connector.source_path,
            mapping_proposal=self.wizard.matching_proposal
        )
        prof_mgr = ProfileManager(self.wizard.db)
        prof_mgr.save_profile(profile)
        self.wizard.profile = profile
        from ..core.import_engine import ImportEngine
        from ..import_wizard.import_thread import ImportThread
        self.import_thread = ImportThread(self.wizard.db, self.wizard.connector, profile)
        self.import_thread.progress.connect(self.update_progress)
        self.import_thread.finished.connect(self.on_finished)
        self.import_thread.error.connect(self.on_error)
        self.import_thread.start()
    def update_progress(self, current, total):
        self.progress.setMaximum(total)
        self.progress.setValue(current)
        self.log.append(f"{self.translator.tr('processed_records')}: {current} / {total}")
    def on_finished(self, stats):
self.log.append(f"{self.translator.tr('import_finished')}: {self.translator.tr('inserted')}: {stats['inserted']}, {self.translator.tr('updated')}: {stats['updated']}, {self.translator.tr('errors')}: {stats['errors']}")
        QMessageBox.information(self, self.translator.tr('import_finished_title'), self.translator.tr('import_finished'))
    def on_error(self, msg):
        self.log.append(f"{self.translator.tr('error')}: {msg}")
        QMessageBox.critical(self, self.translator.tr('error'), msg)
```

Файл: /home/sergey/Documents/configurate/openDBF.py (относительно: openDBF.py)

```
import sys
import fcntl
import re
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QPushButton, QTableView, QFileDialog,
                             QMessageBox, QLabel, QLineEdit, QHeaderView)
from PyQt5.QtCore import Qt, QAbstractTableModel, QModelIndex, pyqtSignal, QSortFilterProxyModel
from PyQt5.QtGui import QFont
from dbfread import DBF
import dbf

class DBFTableModel(QAbstractTableModel):
    """Модель для отображения данных DBF с поддержкой фильтрации."""
    def __init__(self, data, headers):
        super().__init__()
        self._data = data              # список записей (словарей)
        self._headers = headers        # список имен полей
        self._filtered_rows = list(range(len(data)))  # индексы видимых строк
        self._filters = {}             # словарь {col: pattern}
    def rowCount(self, parent=QModelIndex()):
        return len(self._filtered_rows)
    def columnCount(self, parent=QModelIndex()):
        return len(self._headers)
    def data(self, index, role=Qt.DisplayRole):
        if not index.isValid():
            return None
        if role == Qt.DisplayRole:
            row = self._filtered_rows[index.row()]
            col = index.column()
            value = self._data[row][self._headers[col]]
            return str(value) if value is not None else ""
        return None
    def headerData(self, section, orientation, role=Qt.DisplayRole):
        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
            return self._headers[section]
        return None
    def setFilter(self, col, pattern):
        """Устанавливает фильтр для колонки."""
        if pattern:
            self._filters[col] = pattern
        else:
            self._filters.pop(col, None)
        self.updateFilter()
    def clearFilters(self):
        self._filters.clear()
        self.updateFilter()
    def updateFilter(self):
        """Пересчитывает список видимых строк на основе фильтров."""
        self.beginResetModel()
        self._filtered_rows = []
        for row, record in enumerate(self._data):
```

```
        match = True
        for col, pattern in self._filters.items():
            value = str(record[self._headers[col]]) if record[self._headers[col]] is not None else ""
            # Простая проверка на вхождение или regex
            regex = re.escape(pattern).replace(r'\?', '.').replace(r'\*', '.*') + '$'
            if not re.match(regex, value):
                match = False
                break
        if match:
            self._filtered_rows.append(row)
    self.endResetModel()

class DBFViewer(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Просмотрщик DBF")
        self.resize(900, 700)
        central = QWidget()
        self.setCentralWidget(central)
        main_layout = QVBoxLayout(central)
        # Верхняя панель
        top_layout = QHBoxLayout()
        self.btn_open = QPushButton("Открыть DBF файл")
        self.btn_open.clicked.connect(self.open_file)
        top_layout.addWidget(self.btn_open)
        self.btn_apply = QPushButton("Применить фильтр")
        self.btn_apply.clicked.connect(self.apply_filters)
        top_layout.addWidget(self.btn_apply)
        self.btn_clear = QPushButton("Сбросить фильтры")
        self.btn_clear.clicked.connect(self.clear_filters)
        top_layout.addWidget(self.btn_clear)
        self.info_label = QLabel("Файл не открыт")
        top_layout.addStretch()
        top_layout.addWidget(self.info_label)
        main_layout.addLayout(top_layout)
        # Панель фильтров (горизонтальные QLineEdit)
        self.filter_layout = QHBoxLayout()
        main_layout.addLayout(self.filter_layout)
        # Таблица
        self.table_view = QTableView()
        font = self.table_view.font()
        font.setPointSize(20)
        self.table_view.setFont(font)
        self.table_view.horizontalHeader().setStretchLastSection(True)
        self.table_view.setSortingEnabled(False)
        main_layout.addWidget(self.table_view)
        self.model = None
        self.filter_edits = []
    def open_file(self):
        file_path, _ = QFileDialog.getOpenFileName(
            self, "Выберите DBF файл", "", "DBF files (*.DBF);;All files (*)"
        )
        if not file_path:
            return
        try:
            # Определяем кодировку
            encodings = ['cp866', 'utf-8', 'windows-1251']
            dbf = None
            for enc in encodings:
                try:
                    dbf = DBF(file_path, encoding=enc)
                    # Проверяем, что читается первая запись
                    next(iter(dbf))
                    break
                except (UnicodeDecodeError, StopIteration):
                    continue
            if dbf is None:
                dbf = DBF(file_path)
            # Читаем все записи (для больших файлов осторожно, но КЛАДР влезет)
            data = list(dbf)
            headers = [field.name for field in dbf.fields]
            # Создаём модель
            self.model = DBFTableModel(data, headers)
            self.table_view.setModel(self.model)
            # Создаём поля ввода для фильтров
            self._create_filter_inputs(headers)
            self.info_label.setText(f"Файл: {file_path.split('/')[-1]} | Всего записей: {len(data)}")
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось прочитать файл:\n{str(e)}")
    def _create_filter_inputs(self, headers):
        # Очищаем старые фильтры
        self._clear_filter_layout()
        self.filter_edits = []
        for col, header in enumerate(headers):
            line_edit = QLineEdit()
            line_edit.setPlaceholderText(header)
            line_edit.returnPressed.connect(self.apply_filters)
            self.filter_layout.addWidget(line_edit)
            self.filter_edits.append((col, line_edit))
    def _clear_filter_layout(self):
        for i in reversed(range(self.filter_layout.count())):
            widget = self.filter_layout.itemAt(i).widget()
            if widget:
                widget.deleteLater()
    def apply_filters(self):
        if self.model is None:
            return
        for col, line_edit in self.filter_edits:
            text = line_edit.text().strip()
            self.model.setFilter(col, text if text else None)
    def clear_filters(self):
        if self.model is None:
            return
        self.model.clearFilters()
        for _, line_edit in self.filter_edits:
            line_edit.clear()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    viewer = DBFViewer()
    viewer.show()
    sys.exit(app.exec_())
```

Файл: /home/sergey/Documents/configurate/sqllite/data_browser_dialog.py (относительно

```
from PyQt5.QtWidgets import (
    QDialog, QVBoxLayout, QHBoxLayout, QTableWidget, QTableWidgetItem,
    QPushButton, QMessageBox, QHeaderView, QAbstractItemView
)
from database import DatabaseManager
class DataBrowserDialog(QDialog):
    """Просмотр и редактирование строк таблицы или представления."""
    def __init__(self, db: DatabaseManager, table_name: str, parent=None):
        super().__init__(parent)
        self.db = db
        self.table_name = table_name
        self.setWindowTitle(f"Данные: {table_name}")
        self.resize(900, 600)
        self.init_ui()
        self._load_data()
    def init_ui(self):
        layout = QVBoxLayout(self)
        self.data_table = QTableWidget()
        self.data_table.setEditTriggers(QAbstractItemView.AllEditTriggers)
        self.data_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
```

```
        layout.addWidget(self.data_table)
        btn_layout = QHBoxLayout()
        self.add_btn = QPushButton("Добавить строку")
        self.add_btn.clicked.connect(self._add_row)
        self.del_btn = QPushButton("Удалить строку")
        self.del_btn.clicked.connect(self._delete_row)
        self.save_btn = QPushButton("Сохранить")
        self.save_btn.clicked.connect(self._save_changes)
        btn_layout.addWidget(self.add_btn)
        btn_layout.addWidget(self.del_btn)
        btn_layout.addStretch()
        btn_layout.addWidget(self.save_btn)
        layout.addLayout(btn_layout)
    def _load_data(self):
        cursor = self.db.execute(f"SELECT * FROM {self.table_name}")
        columns = [desc[0] for desc in cursor.description]
        self.data_table.setColumnCount(len(columns))
        self.data_table.setHorizontalHeaderLabels(columns)
        self.data_table.setRowCount(0)
        for row_data in cursor.fetchall():
            row_idx = self.data_table.rowCount()
            self.data_table.insertRow(row_idx)
            for col_idx, value in enumerate(row_data):
                txt = str(value) if value is not None else ""
                self.data_table.setItem(row_idx, col_idx, QTableWidgetItem(txt))
    def _add_row(self):
        self.data_table.insertRow(self.data_table.rowCount())
    def _delete_row(self):
        row = self.data_table.currentRow()
        if row >= 0:
            self.data_table.removeRow(row)
    def _save_changes(self):
        reply = QMessageBox.question(self, "Подтверждение",
                                     "Перезаписать все данные таблицы?\n"
                                     "(Таблица строки будут удалены и заменены текущими.)",
                                     QMessageBox.Yes | QMessageBox.No)
        if reply != QMessageBox.Yes:
            return
        try:
            self._perform_save()
        except Exception as e:
            err_msg = str(e)
            if "FOREIGN KEY constraint failed" in err_msg:
                reply = QMessageBox.warning(self, "Ошибка сохранения",
                                            f"Операция нарушает целостность внешних ключей:\n{err_msg}\n\n"
                                            "Разрешить временно отключить внешние ключи?",
                                            QMessageBox.Yes | QMessageBox.No)
                if reply == QMessageBox.Yes:
                    try:
                        self.db.execute("PRAGMA foreign_keys = OFF")
                        self._perform_save()
                        self.db.execute("PRAGMA foreign_keys = ON")
                        QMessageBox.information(self, "Готово", "Изменения сохранены с отключенными FK.")
                    except Exception as e2:
                        self.db.execute("PRAGMA foreign_keys = ON")
                        QMessageBox.critical(self, "Ошибка", str(e2))
                    self.db.rollback()
                else:
                    self.db.rollback()
            else:
                QMessageBox.critical(self, "Ошибка", err_msg)
                self.db.rollback()
    def _perform_save(self):
        """Выполняет DELETE + INSERT для полной перезаписи таблицы."""
        self.db.execute(f"DELETE FROM {self.table_name}")
        cols = [self.data_table.horizontalHeaderItem(i).text()
                for i in range(self.data_table.columnCount())]
        placeholders = ", ".join("?" for _ in cols)
        sql = f"INSERT INTO {self.table_name} ({', '.join(cols)}) VALUES ({placeholders})"
        for row in range(self.data_table.rowCount()):
            values = []
            for col in range(self.data_table.columnCount()):
                item = self.data_table.item(row, col)
                val = item.text() if item else ""
                # Попытка преобразования в число
                try:
                    if '.' in val:
                        val = float(val)
                    else:
                        val = int(val)
                except ValueError:
                    pass
                values.append(val)
            self.db.execute(sql, values)
        self.db.commit()
        QMessageBox.information(self, "Готово", "Данные успешно сохранены.")
```

Файл: /home/sergey/Documents/configurate/sqllite/database.py (относительно: sqllite/data

```
import sqlite3
import os
class DatabaseManager:
    """Управление соединением с SQLite и выполнение операций."""
    def __init__(self):
        self.connection = None
        self.path = None
    def open(self, db_path: str):
        """Открыть существующую БД или создать новую."""
        if self.connection:
            self.connection.close()
        self.path = db_path
        self.connection = sqlite3.connect(db_path)
        self.connection.execute("PRAGMA foreign_keys = ON")
        self.connection.row_factory = sqlite3.Row  # для доступа по именам столбцов
    def close(self):
        if self.connection:
            self.connection.close()
            self.connection = None
    def execute(self, sql: str, params=None):
        """Выполнить запрос, возвращает курсор."""
        cursor = self.connection.cursor()
        if params:
            cursor.execute(sql, params)
        else:
            cursor.execute(sql)
        return cursor
    def commit(self):
        self.connection.commit()
    def rollback(self):
        self.connection.rollback()
    def get_tables(self):
        """Список пользовательских таблиц."""
cursor = self.execute("SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite_%'")
        return [row['name'] for row in cursor.fetchall()]
    def get_views(self):
        """Список представлений и их определений."""
        cursor = self.execute("SELECT name, sql FROM sqlite_master WHERE type='view'")
        return [{'name': row['name'], 'sql': row['sql']} for row in cursor.fetchall()]
    def get_table_schema(self, table_name: str):
        """Вся информация о структуре таблицы."""
        cursor = self.execute(f"PRAGMA table_info('{table_name}')")
        columns = [dict(row) for row in cursor.fetchall()]
        # Получаем внешние ключи
        cursor = self.execute(f"PRAGMA foreign_key_list('{table_name}')")
        foreign_keys = [dict(row) for row in cursor.fetchall()]
        return {
            'name': table_name,
```

```
        'columns': columns,
        'foreign_keys': foreign_keys
    }
def table_exists(self, table_name: str) -> bool:
    cursor = self.execute("SELECT name FROM sqlite_master WHERE type='table' AND name=?", (table_name,))
    return bool(cursor.fetchone())
```

Файл: /home/sergey/Documents/configurate/sqllite/main.py (относительно: sqllite/main.py

```
import sys
import os
from PyQt5.QtWidgets import QApplication
from main_window import MainWindow
DEFAULT_DIR = "/home/sergey/Documents/configurate/sqllite"
def main():
    os.makedirs(DEFAULT_DIR, exist_ok=True)
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
if __name__ == "__main__":
    main()
```

Файл: /home/sergey/Documents/configurate/sqllite/main_window.py (относительно: sqllite

```
import os
from PyQt5.QtWidgets import QMainWindow, QTabWidget, QMenuBar, QAction, QFileDialog, QStatusBar
from database import DatabaseManager
from widgets.table_list_widget import TableListWidget
from widgets.view_list_widget import ViewListWidget
DEFAULT_DIR = "/home/sergey/Documents/configurate/sqllite"
class MainWindow(QMainWindow):
    """Главное окно приложения."""
    def __init__(self):
        super().__init__()
        self.db = DatabaseManager()
        self.setWindowTitle("SQLite Manager")
        self.resize(1100, 750)
        self._setup_menu()
        self._setup_ui()
        self.statusBar().showMessage("Готов. Откройте или создайте базу данных.")
    def _setup_menu(self):
        menu_bar = self.menuBar()
        file_menu = menu_bar.addMenu("Файл")
        open_act = QAction("Открыть...", self)
        open_act.triggered.connect(self._open_db)
        file_menu.addAction(open_act)
        create_act = QAction("Создать...", self)
        create_act.triggered.connect(self._create_db)
        file_menu.addAction(create_act)
        file_menu.addSeparator()
        exit_act = QAction("Выход", self)
        exit_act.triggered.connect(self.close)
        file_menu.addAction(exit_act)
    def _setup_ui(self):
        self.tabs = QTabWidget()
        self.table_widget = TableListWidget(self.db)
        self.view_widget = ViewListWidget(self.db)
        self.tabs.addTab(self.table_widget, "Таблицы")
        self.tabs.addTab(self.view_widget, "Представления")
        self.setCentralWidget(self.tabs)
    def _open_db(self):
        path, _ = QFileDialog.getOpenFileName(
            self, "Открыть базу данных", DEFAULT_DIR,
            "SQLite файлы (*.db *.sqlite *.sqlite3);;Все файлы (*)"
        )
        if path:
            self.db.open(path)
            self._after_connect()
    def _create_db(self):
        path, _ = QFileDialog.getSaveFileName(
            self, "Создать базу данных", DEFAULT_DIR,
            "SQLite файлы (*.db *.sqlite *.sqlite3)"
        )
        if path:
            if not os.path.exists(path):
                open(path, 'a').close()
            self.db.open(path)
            self._after_connect()
    def _after_connect(self):
        self.table_widget.refresh()
        self.view_widget.refresh()
        self.statusBar().showMessage(f"База данных: {self.db.path}")
    def closeEvent(self, event):
        self.db.close()
        event.accept()
```

Файл: /home/sergey/Documents/configurate/sqllite/models.py (относительно: sqllite/mode

```
from dataclasses import dataclass, field
from typing import List, Optional
@dataclass
class ColumnDef:
    """Определение столбца таблицы."""
    name: str
    col_type: str = "TEXT"    # тип как строка (INTEGER, TEXT, ...)
    pk: bool = False          # часть первичного ключа
    not_null: bool = False
    unique: bool = False
    default: Optional[str] = None
@dataclass
class ForeignKey:
    """Внешний ключ."""
    from_column: str
    to_table: str
    to_column: str
@dataclass
class TableSchema:
    """Полное описание таблицы."""
    name: str
    columns: List[ColumnDef] = field(default_factory=list)
    foreign_keys: List[ForeignKey] = field(default_factory=list)
    def add_column(self, col: ColumnDef):
        self.columns.append(col)
    def add_foreign_key(self, fk: ForeignKey):
        self.foreign_keys.append(fk)
    def generate_create_sql(self) -> str:
        """Сгенерировать SQL CREATE TABLE."""
        cols_sql = []
        pk_cols = []
        for col in self.columns:
            part = f"{col.name} {col.col_type}"
            if col.not_null:
                part += " NOT NULL"
            if col.unique:
                part += " UNIQUE"
            if col.default is not None:
                part += f" DEFAULT {col.default}"  # упрощённо, без кавычек для текста
            if col.pk:
```

```
            pk_cols.append(col.name)
        cols_sql.append(part)
    # FK
    for fk in self.foreign_keys:
        cols_sql.append(
            f"FOREIGN KEY ({fk.from_column}) REFERENCES {fk.to_table}({fk.to_column})"
        )
    if pk_cols:
        cols_sql.append(f"PRIMARY KEY ({', '.join(pk_cols)})")
    sql = f"CREATE TABLE {self.name} (\n  " + ",\n  ".join(cols_sql) + "\n)"
    return sql
```

Файл: /home/sergey/Documents/configurate/sqllite/table_editor_dialog.py (относительно: s

```
from PyQt5.QtWidgets import (
    QDialog, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QTableWidget,
    QTableWidgetItem, QPushButton, QComboBox, QCheckBox, QWidget,
    QGroupBox, QDialogButtonBox, QMessageBox, QHeaderView
)
from PyQt5.QtCore import Qt
from models import ColumnDef, ForeignKey, TableSchema
from database import DatabaseManager
SQLITE_TYPES = [
    "INTEGER", "TEXT", "REAL", "BLOB", "NUMERIC",
    "DATE", "DATETIME", "TIMESTAMP", "BOOLEAN",
    "VARCHAR(255)", "CHAR(50)", "DECIMAL(10,2)",
]
class TableEditorDialog(QDialog):
    """Диалог создания/редактирования структуры таблицы"""
    def __init__(self, db: DatabaseManager, table_name: str = None, parent=None):
        super().__init__(parent)
        self.db = db
        self.original_name = table_name
        self.schema = TableSchema(name=table_name or "")
        self.setWindowTitle("Редактировать таблицу" if table_name else "Создать таблицу")
        self.resize(800, 550)
        self.init_ui()
        if table_name:
            self._load_existing_schema()
    def _init_ui(self):
        layout = QVBoxLayout(self)
        # Название
        name_layout = QHBoxLayout()
        name_layout.addWidget(QLabel("Название таблицы:"))
        self.name_edit = QLineEdit()
        name_layout.addWidget(self.name_edit)
        layout.addLayout(name_layout)
        # Таблица столбцов
        self.col_table = QTableWidget(0, 5)
        self.col_table.setHorizontalHeaderLabels(["Имя", "Тип", "PK", "Not Null", "Unique"])
        self.col_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        layout.addWidget(self.col_table)
        # Кнопки столбцов
        col_btn_layout = QHBoxLayout()
        btn_add_col = QPushButton("Добавить столбец")
        btn_add_col.clicked.connect(self._add_column_row)
        btn_remove_col = QPushButton("Удалить столбец")
        btn_remove_col.clicked.connect(self._remove_column_row)
        col_btn_layout.addWidget(btn_add_col)
        col_btn_layout.addWidget(btn_remove_col)
        layout.addLayout(col_btn_layout)
        # Внешние ключи
        fk_group = QGroupBox("Внешние ключи")
        fk_layout = QVBoxLayout()
        self.fk_table = QTableWidget(0, 2)
        self.fk_table.setHorizontalHeaderLabels(["Столбец", "Ссылка (таблица.столбец)"])
        self.fk_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        fk_layout.addWidget(self.fk_table)
        fk_btn_layout = QHBoxLayout()
        btn_add_fk = QPushButton("Добавить FK")
        btn_add_fk.clicked.connect(self._add_fk_row)
        btn_remove_fk = QPushButton("Удалить FK")
        btn_remove_fk.clicked.connect(self._remove_fk_row)
        fk_btn_layout.addWidget(btn_add_fk)
        fk_btn_layout.addWidget(btn_remove_fk)
        fk_layout.addLayout(fk_btn_layout)
        fk_group.setLayout(fk_layout)
        layout.addWidget(fk_group)
        # OK/Cancel
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self._on_save)
        btn_box.rejected.connect(self.reject)
        layout.addWidget(btn_box)
        self.name_edit.editingFinished.connect(self._update_schema_name)
    def _add_column_row(self):
        row = self.col_table.rowCount()
        self.col_table.insertRow(row)
        self.col_table.setItem(row, 0, QTableWidgetItem(""))
        type_combo = QComboBox()
        type_combo.addItems(SQLITE_TYPES)
        self.col_table.setCellWidget(row, 1, type_combo)
        for col_idx in [2, 3, 4]:
            w = QWidget()
            lay = QHBoxLayout()
            lay.setAlignment(Qt.AlignCenter)
            cb = QCheckBox()
            lay.addWidget(cb)
            w.setLayout(lay)
            self.col_table.setCellWidget(row, col_idx, w)
    def _remove_column_row(self):
        row = self.col_table.currentRow()
        if row >= 0:
            self.col_table.removeRow(row)
    def _add_fk_row(self):
        row = self.fk_table.rowCount()
        self.fk_table.insertRow(row)
        self.fk_table.setItem(row, 0, QTableWidgetItem(""))
        ref_combo = QComboBox()
        ref_combo.setEditable(True)
        tables = self.db.get_tables()
        for t in tables:
            schema = self.db.get_table_schema(t)
            for col in schema.columns:
                ref_combo.addItem(f"{t}.{col['name']}")
        self.fk_table.setCellWidget(row, 1, ref_combo)
    def _remove_fk_row(self):
        row = self.fk_table.currentRow()
        if row >= 0:
            self.fk_table.removeRow(row)
    def _load_existing_schema(self):
        info = self.db.get_table_schema(self.original_name)
        self.name_edit.setText(info['name'])
        for col in info['columns']:
            self._add_column_row()
            row = self.col_table.rowCount() - 1
            self.col_table.item(row, 0).setText(col['name'])
            type_combo = self.col_table.cellWidget(row, 1)
            if col['type']:
                type_combo.setCurrentText(col['type'].upper())
            pk_cb = self.col_table.cellWidget(row, 2).findChild(QCheckBox)
            pk_cb.setChecked(bool(col['pk']))
            nn_cb = self.col_table.cellWidget(row, 3).findChild(QCheckBox)
            nn_cb.setChecked(bool(col['notnull']))
            un_cb = self.col_table.cellWidget(row, 4).findChild(QCheckBox)
```

```
            un_cb.setChecked(False)  # уникальность не загружаем, можно доработать
        for fk in info['foreign_keys']:
            self.add_fk_row()
            row = self.fk_table.rowCount() - 1
            self.fk_table.item(row, 0).setText(fk['from'])
            ref_combo = self.fk_table.cellWidget(row, 1)
            ref_str = f"{fk['table']}.{fk['to']}"
            ref_combo.setCurrentText(ref_str)
    def build_schema(self) -> TableSchema:
        schema = TableSchema(name=self.name_edit.text().strip())
        for row in range(self.col_table.rowCount()):
            name_item = self.col_table.item(row, 0)
            if not name_item or not name_item.text().strip():
                continue
            name = name_item.text().strip()
            type_combo = self.col_table.cellWidget(row, 1)
            col_type = type_combo.currentText()
            pk = self.col_table.cellWidget(row, 2).findChild(QCheckBox).isChecked()
            nn = self.col_table.cellWidget(row, 3).findChild(QCheckBox).isChecked()
            un = self.col_table.cellWidget(row, 4).findChild(QCheckBox).isChecked()
            schema.add_column(Column(name, col_type, pk, nn, un))
        for row in range(self.fk_table.rowCount()):
            from_item = self.fk_table.item(row, 0)
            ref_widget = self.fk_table.cellWidget(row, 1)
            if from_item and ref_widget:
                from_col = from_item.text().strip()
                ref_text = ref_widget.currentText().strip()
                if from_col and '.' in ref_text:
                    to_table, to_col = ref_text.split('.', 1)
                    schema.add_foreign_key(ForeignKey(from_col, to_table, to_col))
        return schema
    def _on_save(self):
        schema = self._build_schema()
        if not schema.name:
            QMessageBox.warning(self, "Ошибка", "Имя таблицы не может быть пустым")
            return
        if not schema.columns:
            QMessageBox.warning(self, "Ошибка", "Добавьте хотя бы один столбец")
            return
        # Новая таблица
        if not self.original_name:
            try:
                self.db.execute(schema.generate_create_sql())
                self.db.commit()
                self.accept()
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", str(e))
                self.db.rollback()
            return
        # Редактирование существующей
        old_name = self.original_name
        new_name = schema.name
        try:
            # Создаём таблицу с новой структурой
            self.db.execute(schema.generate_create_sql())
            # Переносим данные из старой таблицы
            old_cols = [c['name'] for c in self.db.get_table_schema(old_name)['columns']]
            new_cols = [c.name for c in schema.columns]
            common = [c for c in old_cols if c in new_cols]
            if common:
                cols = ', '.join(common)
                self.db.execute(f"INSERT INTO {new_name} ({cols}) SELECT {cols} FROM {old_name}")
            # Удаляем старую таблицу, временно отключив FK
            self.db.execute("PRAGMA foreign_keys = OFF")
            self.db.execute(f"DROP TABLE {old_name}")
            self.db.execute("PRAGMA foreign_keys = ON")
            self.db.commit()
            self.accept()
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", str(e))
            # Попытка подчистить новую таблицу, если она была создана
            try:
                self.db.execute("PRAGMA foreign_keys = OFF")
                if self.db.table_exists(new_name):
                    self.db.execute(f"DROP TABLE IF EXISTS {new_name}")
                self.db.execute("PRAGMA foreign_keys = ON")
            except:
                pass
            self.db.rollback()
    def _update_schema_name(self):
        self.schema.name = self.name_edit.text().strip()
```

Файл: /home/sergey/Documents/configurate/sqllite/view_dialog.py (относительно: sqllite/v

```
from PyQt5.QtWidgets import (
    QDialog, QVBoxLayout, QLabel, QLineEdit, QDialogButtonBox, QMessageBox
)
from database import DatabaseManager
class ViewDialog(QDialog):
    """Создание или редактирование представления."""
    def __init__(self, db: DatabaseManager, view_name=None, view_sql=None, parent=None):
        super().__init__(parent)
        self.db = db
        self.original_name = view_name
        self.setWindowTitle("Редактировать представление" if view_name else "Создать представление")
        self.resize(600, 300)
        layout = QVBoxLayout(self)
        layout.addWidget(QLabel("Имя представления:"))
        self.name_edit = QLineEdit()
        layout.addWidget(self.name_edit)
        layout.addWidget(QLabel("SQL-запрос:"))
        self.sql_edit = QLineEdit()
        layout.addWidget(self.sql_edit)
        if view_sql:
            self.name_edit.setText(view_name)
            self.sql_edit.setText(view_sql)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self._save)
        btn_box.rejected.connect(self.reject)
        layout.addWidget(btn_box)
    def _save(self):
        name = self.name_edit.text().strip()
        sql = self.sql_edit.text().strip()
        if not name or not sql:
            QMessageBox.warning(self, "Ошибка", "Заполните имя и запрос")
            return
        try:
            if self.original_name:
                self.db.execute(f"DROP VIEW IF EXISTS {self.original_name}")
            self.db.execute(f"CREATE VIEW {name} AS {sql}")
            self.db.commit()
            self.accept()
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", str(e))
            self.db.rollback()
```

Файл: /home/sergey/Documents/configurate/sqllite/widgets/__init__.py (относительно: sqlli

```
# widgets/__init__.py
from .table_list_widget import TableListWidget
from .view_list_widget import ViewListWidget
```

Файл: /home/sergey/Documents/configurate/sqllite/widgets/table_list_widget.py (относител

```
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QListWidget, QMessageBox
from database import DatabaseManager
from table_editor_dialog import TableEditorDialog
from data_browser_dialog import DataBrowserDialog
class TableListWidget(QWidget):
    """Вкладка со списком таблиц и кнопками управления."""
    def __init__(self, db: DatabaseManager, parent=None):
```

```
        super().__init__(parent)
        self.db = db
        self._init_ui()
    def _init_ui(self):
        layout = QVBoxLayout(self)
        self.list_widget = QListWidget()
        layout.addWidget(self.list_widget)
        btn_layout = QHBoxLayout()
        self.add_btn = QPushButton("Добавить")
        self.edit_btn = QPushButton("Редактировать")
        self.del_btn = QPushButton("Удалить")
        self.browse_btn = QPushButton("Browse")
        btn_layout.addWidget(self.add_btn)
        btn_layout.addWidget(self.edit_btn)
        btn_layout.addWidget(self.del_btn)
        btn_layout.addWidget(self.browse_btn)
        layout.addLayout(btn_layout)
        self.add_btn.clicked.connect(self._add_table)
        self.edit_btn.clicked.connect(self._edit_table)
        self.del_btn.clicked.connect(self._delete_table)
        self.browse_btn.clicked.connect(self._browse_table)
    def refresh(self):
        self.list_widget.clear()
        tables = self.db.get_tables()
        self.list_widget.addItems(tables)
    def _add_table(self):
        dlg = TableEditorDialog(self.db, parent=self)
        if dlg.exec_():
            self.refresh()
    def _edit_table(self):
        item = self.list_widget.currentItem()
        if not item: return
        name = item.text()
        dlg = TableEditorDialog(self.db, name, parent=self)
        if dlg.exec_():
            self.refresh()
    def _delete_table(self):
        item = self.list_widget.currentItem()
        if not item: return
        name = item.text()
        if QMessageBox.question(self, "Удалить", f"Удалить таблицу {name}?") == QMessageBox.Yes:
            try:
                self.db.execute(f"DROP TABLE IF EXISTS {name}")
                self.db.commit()
                self.refresh()
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", str(e))
    def _browse_table(self):
        item = self.list_widget.currentItem()
        if not item: return
        name = item.text()
        dlg = DataBrowserDialog(self.db, name, parent=self)
        dlg.exec_()
```

Файл: /home/sergey/Documents/configurate/sqllite/widgets/view_list_widget.py (относител

```
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QListWidget, QMessageBox
from database.manager import DatabaseManager
from view_dialog.main_view_dialog import ViewDialog
from data_browser_dialog.main_data_browser_dialog import DataBrowserDialog
from PyQt5.QtWidgets import QAbstractItemView
class ViewListWidget(QWidget):
    """Вкладка со списком представлений и кнопками."""
    def __init__(self, db: DatabaseManager, parent=None):
        super().__init__(parent)
        self.db = db
        self._init_ui()
    def _init_ui(self):
        layout = QVBoxLayout(self)
        self.list_widget = QListWidget()
        layout.addWidget(self.list_widget)
        btn_layout = QHBoxLayout()
        self.add_btn = QPushButton("Добавить")
        self.del_btn = QPushButton("Удалить")
        self.browse_btn = QPushButton("Browse")
        btn_layout.addWidget(self.add_btn)
        btn_layout.addWidget(self.del_btn)
        btn_layout.addWidget(self.browse_btn)
        layout.addLayout(btn_layout)
        self.add_btn.clicked.connect(self._add_view)
        self.del_btn.clicked.connect(self._delete_view)
        self.browse_btn.clicked.connect(self._browse_view)
    def refresh(self):
        self.list_widget.clear()
        views = self.db.get_views()
        for v in views:
            self.list_widget.addItem(v['name'])
    def _add_view(self):
        dlg = ViewDialog(self.db, parent=self)
        if dlg.exec_():
            self.refresh()
    def _delete_view(self):
        item = self.list_widget.currentItem()
        if not item: return
        name = item.text()
        if QMessageBox.question(self, "Удалить", f"Удалить представление {name}?") == QMessageBox.Yes:
            try:
                self.db.execute(f"DROP VIEW IF EXISTS {name}")
                self.db.commit()
                self.refresh()
            except Exception as e:
                QMessageBox.critical(self, "Ошибка", str(e))
    def _browse_view(self):
        item = self.list_widget.currentItem()
        if not item: return
        name = item.text()
        dlg = DataBrowserDialog(self.db, name, parent=self)
        dlg.data_table.setEditTriggers(QAbstractItemView.NoEditTriggers)
        dlg.add_btn.setVisible(False)
        dlg.del_btn.setVisible(False)
        dlg.save_btn.setVisible(False)
        dlg.exec_()
        self.refresh()
```

Файл: /home/sergey/Documents/configurate/tabs/__init__.py (относительно: tabs/__init__.p

```
from .balance_units_tab import BalanceUnitsTab
from .users_roles_tab import UsersRolesTab
from .entities_tab import EntitiesTab
from .documents_tab import DocumentsTab
from .data_view_tab import DataViewTab
from .relations_tab import RelationsTab
__all__ = [
    'BalanceUnitsTab',
    'UsersRolesTab',
    'EntitiesTab',
    'DocumentsTab',
    'DataViewTab',
    'RelationsTab',
]
```

Файл: /home/sergey/Documents/configurate/tabs/balance_units_tab.py (относительно: tab

```
# tabs/balance_units_tab.py
# tabs/balance_units_tab.py
from PyQt5.QtWidgets import QTreeWidget, QTreeWidgetItem, QMessageBox, QDialog, QFormLayout,
QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QCheckBox
from PyQt5.QtCore import Qt
from base_tab import BaseTab
from crud_buttons import CrudButtons
from widgets import ReferenceEditor
from lang.local_translator import LocalTranslator

class BalanceUnitEditDialog(QDialog):
    def __init__(self, parent=None, data=None, db=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_edit_balance_unit'))
        self.data = data or {}
        self.db = db
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(self.data.get("cname", ""))
        self.edit_alias = QLineEdit(self.data.get("calias", ""))
        self.ref_parent = ReferenceEditor(
            table_name='auth.balanceunits',
            display_field='cname',
            db=self.db
        )
        if self.data.get("parentid"):
            sql = "SELECT cname FROM auth.balanceunits WHERE id = %s"
            res = self.db.execute_query(sql, (self.data["parentid"],))
            if res:
                self.ref_parent.set_value(self.data["parentid"], res[0][0])
        self.chk_is_bu = QCheckBox(self.translator.tr('label_is_balance_unit'))
        self.chk_is_bu.setChecked(self.data.get("lisbalanceunit", True))
        form.addRow(self.translator.tr('field_name'), self.edit_name)
        form.addRow(self.translator.tr('field_alias'), self.edit_alias)
        form.addRow(self.translator.tr('field_parent'), self.ref_parent)
        form.addRow(self.chk_is_bu)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))

        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "cname": self.edit_name.text(),
            "calias": self.edit_alias.text(),
            "parentid": self.ref_parent.get_value(),
            "lisbalanceunit": self.chk_is_bu.isChecked()
        }

class BalanceUnitsTab(BaseTab):
    def __init__(self, parent=None, db=None):
        super().__init__(parent, db)
        self.crud = CrudButtons(self)
        self.layout.addWidget(self.crud)
        self.tree = QTreeWidget()
        self.tree.setColumnWidth(0, 50)
        self.tree.setColumnWidth(1, 200)
        self.tree.setColumnWidth(2, 100)
        self.tree.itemDoubleClicked.connect(self.edit_item)
        self.layout.addWidget(self.tree)
        self.crud.add_clicked.connect(self.add_item)
        self.crud.edit_clicked.connect(self.edit_item)
        self.crud.delete_clicked.connect(self.delete_item)
        self.retranslate_ui()   # устанавливаем заголовки при старте
        self.load_data()
    def retranslate_ui(self):
        self.tree.setHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_name'),
            self.translator.tr('field_alias'),
            self.translator.tr('field_active')
        ])
        if hasattr(self, 'crud'):
            self.crud.retranslate_ui()

    def load_data(self):
        sql = "SELECT id, parentid, level, data FROM get_hierarchy_iterative(%s, %s, %s)"
        rows = self.db.execute_query(sql, ('auth.balanceunits', 'id', 'parentid'))
        self.tree.clear()
        items = {}
        for row in rows:
            node_id, parent_id, level, data = row
            name = data.get('cname', '')
            alias = data.get('calias', '')
            is_active = data.get('lisbalanceunit', False)
            item = QTreeWidgetItem()
            item.setText(0, str(node_id))
            item.setText(1, name)
            item.setText(2, alias if alias else "")
            item.setText(3, "Да" if is_active else "Нет")
            item.setData(0, Qt.UserRole, node_id)
            items[node_id] = item
            if parent_id is None:
                self.tree.addTopLevelItem(item)
            else:
                if parent_id in items:
                    items[parent_id].addChild(item)
                else:
                    self.tree.addTopLevelItem(item)
        self.tree.expandAll()
    def get_selected_id(self):
        item = self.tree.currentItem()
        return item.data(0, Qt.UserRole) if item else None
    def add_item(self):
        dlg = BalanceUnitEditDialog(self, db=self.db)
        if dlg.exec_():
            data = dlg.get_data()
            sql = "INSERT INTO auth.balanceunits (cname, calias, parentid, lisbalanceunit)
VALUES (%s, %s, %s, %s)"
self.db.execute_query(sql, (data["cname"], data["calias"], data["parentid"], data["lisbalanceunit"]), fetch=False)
            self.load_data()
    def edit_item(self):
        item_id = self.get_selected_id()
        if not item_id:
            self.show_warning("Выберите запись для редактирования")
        sql = "SELECT cname, calias, parentid, lisbalanceunit FROM auth.balanceunits WHERE id=%s"
        row = self.db.execute_query(sql, (item_id,))
        if not row:
            return
        data = {
            "cname": row[0][0],
            "calias": row[0][1],
            "parentid": row[0][2],
            "lisbalanceunit": row[0][3]
```

```
        }
        dlg = BalanceUnitEditDialog(self, data, db=self.db)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = """UPDATE auth.balanceunits SET cname=%s, calias=%s, parentid=%s, lisbalanceunit=%s WHERE
id=%s"""
            self.db.execute_query(sql, (new_data["cname"], new_data["calias"], new_data["parentid"],
new_data["lisbalanceunit"], item_id), fetch=False)
            self.load_data()
    def delete_item(self):
        item_id = self.get_selected_id()
        if not item_id:
            self.show_warning("Выберите запись для удаления")
            return
        if QMessageBox.question(self, "Подтверждение", "Удалить выбранную запись?") == QMessageBox.Yes:
            sql = "DELETE FROM auth.balanceunits WHERE id=%s"
            self.db.execute_query(sql, (item_id,), fetch=False)
            self.load_data()
```

Файл: /home/sergey/Documents/configurate/tabs/base_tab.py (относительно: tabs/base_ta

```
# tabs/base_tab.py
# tabs/base_tab.py
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QMessageBox
from database.main_db_service import DatabaseService
from lang.local_translator import LocalTranslator

class BaseTab(QWidget):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.translator = LocalTranslator()
        self.layout = QVBoxLayout(self)
        self.layout.setContentsMargins(5, 5, 5, 5)
    def show_error(self, message):
        QMessageBox.critical(self, self.translator.tr('error'), message)
    def show_info(self, message):
        QMessageBox.information(self, self.translator.tr('info'), message)
    def show_warning(self, message):
        QMessageBox.warning(self, self.translator.tr('warning'), message)
    def refresh(self):
        pass
    def retranslate_ui(self):
        """Переопределяется в подклассах."""
        pass
```

Файл: /home/sergey/Documents/configurate/tabs/data_view_tab.py (относительно: tabs/da

```
from PyQt5.QtWidgets import QLabel, QComboBox, QPushButton, QTableWidget, QTableWidgetItem,
QVBoxLayout, QHBoxLayout, QHeaderView
from PyQt5.QtCore import Qt
from .base_tab import BaseTab

class DataViewTab(BaseTab):
    def __init__(self, parent=None, db=None):
        super().__init__(parent, db)
        h_layout = QHBoxLayout()
        h_layout.addWidget(QLabel("Тип сущности:"))
        self.combo_entity = QComboBox()
        self.combo_entity.currentIndexChanged.connect(self.on_entity_changed)
        h_layout.addWidget(self.combo_entity)
        self.btn_refresh = QPushButton("Обновить")
        self.btn_refresh.clicked.connect(self.load_data)
        h_layout.addWidget(self.btn_refresh)
        h_layout.addStretch()
        self.layout.addLayout(h_layout)
        self.table = QTableWidget()
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        self.table.horizontalHeader().setStretchLastSection(True)
        self.table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.layout.addWidget(self.table)
        self.load_entity_types()
    def load_entity_types(self):
        sql = "SELECT id, cname, calias FROM meta.entitytypes ORDER BY cname"
        rows = self.db.execute_query(sql)
        self.combo_entity.clear()
        self.combo_entity.addItem("-- Выберите сущность --", None)
        for eid, name, alias in rows:
            self.combo_entity.addItem(f"{name} ({alias})", eid)
    def on_entity_changed(self, index):
        if index > 0:
            self.load_data()
    def load_data(self):
        entity_id = self.combo_entity.currentData()
        if not entity_id:
            return
        sql = "SELECT calias FROM meta.entitytypes WHERE id=%s"
        res = self.db.execute_query(sql, (entity_id,))
        if not res:
            return
        alias = res[0][0]
        ext_table = f"ext.ext_{alias.lower()}"
        sql = "SELECT to_regclass(%s)"
        exists = self.db.execute_query(sql, (ext_table,))
        if not exists or not exists[0][0]:
            self.table.setRowCount(0)
            self.table.setColumnCount(0)
            self.show_info(f"Таблица {ext_table} не найдена")
            return
        sql_fields = """SELECT cfieldname, calias FROM meta.fields
                        WHERE entitytypeid = %s ORDER BY isorder"""
        fields = self.db.execute_query(sql_fields, (entity_id,))
        if not fields:
            self.table.setRowCount(0)
            self.table.setColumnCount(0)
            self.show_info("У сущности нет полей")
            return
        field_names = [f[0] for f in fields]
        field_aliases = [f[1] for f in fields]
        select_cols = ["v.id as version_id", "v.cname as entity_name"] + [f"e.{fn}" for fn in field_names]
        sql = f"""
            SELECT {', '.join(select_cols)}
            FROM data.entity_instances i
            JOIN data.entity_versions v ON i.id = v.instanceid AND v.cversionstatus = 'Active'
            JOIN {ext_table} e ON v.id = e.versionid
            WHERE i.entitytypeid = %s
            LIMIT 1000
        """
        rows = self.db.execute_query(sql, (entity_id,))
        headers = ["ID версии", "Наименование"] + field_aliases
        self.table.setColumnCount(len(headers))
        self.table.setHorizontalHeaderLabels(headers)
        self.table.setRowCount(0)
        for r in rows:
            pos = self.table.rowCount()
            self.table.insertRow(pos)
            for c, val in enumerate(r):
                item = QTableWidgetItem(str(val) if val is not None else "")
                self.table.setItem(pos, c, item)
```

Файл: /home/sergey/Documents/configurate/tabs/documents_tab.py (относительно: tabs/d

```
# tabs/documents_tab.py
from PyQt5.QtWidgets import (QWidget, QTabWidget, QTreeWidget, QTreeWidgetItem, QTableWidget,
                             QTableWidgetItem, QPushButton, QHBoxLayout, QVBoxLayout,
                             QMessageBox, QDialog, QFormLayout, QLineEdit,
                             QComboBox, QCheckBox, QLabel, QHeaderView)
from PyQt5.QtCore import Qt
from base_tab import BaseTab
from crud_buttons import CrudButtons
from composition_manager import CompositionManager
from lang.local_translator import LocalTranslator

class DocumentTypeEditDialog(QDialog):
    def __init__(self, parent=None, data=None, db=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_document_type_edit'))
        self.data = data or {}
        self.db = db
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(self.data.get("cname", ""))
        self.edit_alias = QLineEdit(self.data.get("calias", ""))
        self.chk_dependent = QCheckBox(self.translator.tr('label_dependent'))
        self.chk_dependent.setChecked(self.data.get("lisdependent", False))
        self.chk_hierarchy = QCheckBox(self.translator.tr('label_hierarchical'))
        self.chk_hierarchy.setChecked(self.data.get("lishierarchy", False))
        self.chk_versioned = QCheckBox(self.translator.tr('label_versioned'))
        self.chk_versioned.setChecked(self.data.get("lisversioned", True))
        # Определяем class_id для типа 'DOC'
        doc_class = self.db.execute_query("SELECT id FROM meta.entity_classes WHERE calias = 'DOC'")
        self.class_id = doc_class[0][0] if doc_class else None
        form.addRow(self.translator.tr('field_name'), self.edit_name)
        form.addRow(self.translator.tr('field_alias'), self.edit_alias)
        form.addRow(self.chk_dependent)
        form.addRow(self.chk_hierarchy)
        form.addRow(self.chk_versioned)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "cname": self.edit_name.text(),
            "calias": self.edit_alias.text(),
            "lisdependent": self.chk_dependent.isChecked(),
            "lishierarchy": self.chk_hierarchy.isChecked(),
            "lisversioned": self.chk_versioned.isChecked(),
            "class_id": self.class_id
        }

class CompositionRoleEditDialog(QDialog):
    def __init__(self, parent=None, data=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_composition_role_edit'))
        self.data = data or {}
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(self.data.get("cname", ""))
        self.edit_alias = QLineEdit(self.data.get("calias", ""))
        self.edit_comment = QLineEdit(self.data.get("mcomment", ""))
        form.addRow(self.translator.tr('field_name'), self.edit_name)
        form.addRow(self.translator.tr('field_alias'), self.edit_alias)
        form.addRow(self.translator.tr('field_comment'), self.edit_comment)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "cname": self.edit_name.text(),
            "calias": self.edit_alias.text(),
            "mcomment": self.edit_comment.text()
        }

class CompositionEditDialog(QDialog):
    def __init__(self, parent=None, data=None, doc_types=None, child_types=None, roles=None, db=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_composition_edit'))
        self.data = data or {}
        self.db = db
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.combo_parent = QComboBox()
        for tid, tname in doc_types or []:
            self.combo_parent.addItem(f"{tname} ({tid})", tid)
        self.combo_child = QComboBox()
        for tid, tname in child_types or []:
            self.combo_child.addItem(f"{tname} ({tid})", tid)
        self.combo_role = QComboBox()
        for rid, rname in roles or []:
            self.combo_role.addItem(f"{rname} ({rid})", rid)
        self.edit_link_field = QLineEdit(self.data.get("c_link_field_name", ""))
        self.edit_sort = QLineEdit(str(self.data.get("isortorder", 10)))
        if data:
            idx = self.combo_parent.findData(data.get("parent_entityid"))
            if idx >= 0: self.combo_parent.setCurrentIndex(idx)
            idx = self.combo_child.findData(data.get("child_entityid"))
            if idx >= 0: self.combo_child.setCurrentIndex(idx)
            idx = self.combo_role.findData(data.get("role_id"))
            if idx >= 0: self.combo_role.setCurrentIndex(idx)
        form.addRow(self.translator.tr('field_parent_document'), self.combo_parent)
        form.addRow(self.translator.tr('field_child_entity'), self.combo_child)
        form.addRow(self.translator.tr('field_role'), self.combo_role)
        form.addRow(self.translator.tr('field_link_field'), self.edit_link_field)
        form.addRow(self.translator.tr('field_order'), self.edit_sort)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "parent_entityid": self.combo_parent.currentData(),
            "child_entityid": self.combo_child.currentData(),
            "role_id": self.combo_role.currentData(),
            "c_link_fieldname": self.edit_link_field.text(),
            "isortorder": int(self.edit_sort.text()) if self.edit_sort.text().isdigit() else 10
        }

class DocumentsTab(BaseTab):
    def __init__(self, parent=None, db=None):
        super().__init__(parent, db)
```

```
    self.translator = LocalTranslator()
    self.composition_manager = CompositionManager(db=self.db, parent_widget=self)
    self.tabs = QTabWidget()
    self.layout.addWidget(self.tabs)
    self.doc_types_tab = self._create_doc_types_tab()
    self.tabs.addTab(self.doc_types_tab, self.translator.tr('tab_document_types'))
    self.roles_tab = self._create_roles_tab()
    self.tabs.addTab(self.roles_tab, self.translator.tr('tab_composition_roles'))
    self.composition_tab = self._create_composition_tab()
    self.tabs.addTab(self.composition_tab, self.translator.tr('tab_master_detail'))
    self.retranslate_ui()
def retranslate_ui(self):
    self.tabs.setTabText(0, self.translator.tr('tab_document_types'))
    self.tabs.setTabText(1, self.translator.tr('tab_composition_roles'))
    self.tabs.setTabText(2, self.translator.tr('tab_master_detail'))
    if hasattr(self, '_doc_crud'):
        self.doc_crud.retranslate_ui()
        self.role_crud.retranslate_ui()
        self.comp_crud.retranslate_ui()
    self._update_doc_tree_headers()
    self._update_role_table_headers()
    self._update_comp_table_headers()
    self._load_doc_types()
    self._load_roles()
    self._load_compositions()
def _update_doc_tree_headers(self):
    self.doc_tree.setHeaderLabels([
        self.translator.tr('field_id'),
        self.translator.tr('field_name'),
        self.translator.tr('field_alias'),
        self.translator.tr('label_dependent'),
        self.translator.tr('label_hierarchical'),
        self.translator.tr('label_versioned')
    ])
def _update_role_table_headers(self):
    self.role_table.setHorizontalHeaderLabels([
        self.translator.tr('field_id'),
        self.translator.tr('field_alias'),
        self.translator.tr('field_name')
    ])
def _update_comp_table_headers(self):
    self.comp_table.setHorizontalHeaderLabels([
        self.translator.tr('field_id'),
        self.translator.tr('field_master'),
        self.translator.tr('field_detail'),
        self.translator.tr('field_role'),
        self.translator.tr('field_link_field')
    ])
def _create_doc_types_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)
    self.doc_crud = CrudButtons(self)
    layout.addWidget(self.doc_crud)
    self.doc_tree = QTreeWidget()
    self.doc_tree.setHeaderLabels(["ID", self.translator.tr('field_name'), self.translator.tr('field_alias'),
                                   self.translator.tr('label_dependent'), self.translator.tr('label_hierarchical'),
                                   self.translator.tr('label_versioned')])
    self.doc_tree.itemDoubleClicked.connect(self.edit_doc_type)
    layout.addWidget(self.doc_tree)
    self.doc_crud.add_clicked.connect(self.add_doc_type)
    self.doc_crud.edit_clicked.connect(self.edit_doc_type)
    self.doc_crud.delete_clicked.connect(self.delete_doc_type)
    self._load_doc_types()
    return tab
def _create_roles_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)
    self.role_crud = CrudButtons(self)
    layout.addWidget(self.role_crud)
    self.role_table = QTableWidget()
    self.role_table.setColumnCount(3)
    self.role_table.setHorizontalHeaderLabels([self.translator.tr('field_id'),
                                               self.translator.tr('field_alias'),
                                               self.translator.tr('field_name')])
    self.role_table.horizontalHeader().setStretchLastSection(True)
    self.role_table.setSelectionBehavior(QTableWidget.SelectRows)
    self.role_table.setEditTriggers(QTableWidget.NoEditTriggers)
    self.role_table.itemDoubleClicked.connect(self.edit_role)
    layout.addWidget(self.role_table)
    self.role_crud.add_clicked.connect(self.add_role)
    self.role_crud.edit_clicked.connect(self.edit_role)
    self.role_crud.delete_clicked.connect(self.delete_role)
    self._load_roles()
    return tab
def _create_composition_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)
    self.comp_crud = CrudButtons(self)
    layout.addWidget(self.comp_crud)
    self.comp_table = QTableWidget()
    self.comp_table.setColumnCount(5)
    self.comp_table.setHorizontalHeaderLabels([self.translator.tr('field_id'),
                                               self.translator.tr('field_master'),
                                               self.translator.tr('field_detail'),
                                               self.translator.tr('field_role'),
                                               self.translator.tr('field_link_field')])
    self.comp_table.horizontalHeader().setStretchLastSection(True)
    self.comp_table.setSelectionBehavior(QTableWidget.SelectRows)
    self.comp_table.setEditTriggers(QTableWidget.NoEditTriggers)
    self.comp_table.itemDoubleClicked.connect(self.edit_composition)
    layout.addWidget(self.comp_table)
    self.comp_crud.add_clicked.connect(self.add_composition)
    self.comp_crud.edit_clicked.connect(self.edit_composition)
    self.comp_crud.delete_clicked.connect(self.delete_composition)
    self._load_compositions()
    return tab
def _load_doc_types(self):
    sql = """SELECT id, cname, calias, lisdependent, lishierarchy, lisversioned
             FROM meta.entitytypes
             WHERE class_id = (SELECT id FROM meta.entity_classes WHERE calias = 'DOC')
             ORDER BY cname ;"""
    rows = self.db.execute_query(sql)
    self.doc_tree.clear()
    for row in rows:
        item = QTreeWidgetItem(self.doc_tree)
        item.setText(0, str(row[0]))
        item.setText(1, row[1])
        item.setText(2, row[2])
        item.setText(3, self.translator.tr('yes') if row[3] else self.translator.tr('no'))
        item.setText(4, self.translator.tr('yes') if row[4] else self.translator.tr('no'))
        item.setText(5, self.translator.tr('yes') if row[5] else self.translator.tr('no'))
        item.setData(0, Qt.UserRole, row[0])
def add_doc_type(self):
    dlg = DocumentTypeEditDialog(self, db=self.db)
    if dlg.exec():
        data = dlg.get_data()
        sql = """INSERT INTO meta.entitytypes
                 (cname, calias, chaseclass, lisdependent, lishierarchy, lisversioned, class_id)
                 VALUES (%s, %s, DOC, %s, %s, %s, %s)"""
        self.db.execute_query(sql, (data["cname"], data["calias"],
                                    data["lisdependent"], data["lishierarchy"],
                                    data["lisversioned"], data["class_id"]), fetch=False)
        self._load_doc_types()
def edit_doc_type(self):
    item = self.doc_tree.currentItem()
    if not item:
        self.show_warning(self.translator.tr('warning_select_document_type'))
        return
    eid = item.data(0, Qt.UserRole)
    sql = """SELECT cname, calias, lisdependent, lishierarchy, lisversioned
             FROM meta.entitytypes WHERE id = %s"""
    row = self.db.execute_query(sql, (eid,))
    if not row:
        return
    data = {
        "cname": row[0][0],
        "calias": row[0][1],
```

```
            "lisdependent": row[0][2],
            "lishierarchy": row[0][3],
            "lisversioned": row[0][4]
        }
        dlg = DocumentTypeEditDialog(self, data, db=self.db)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = "UPDATE meta.entitytypes SET
                cname=%s, calias=%s, lisdependent=%s, lishierarchy=%s, lisversioned=%s
                WHERE id=%s"
            self.db.execute_query(sql, (new_data["cname"], new_data["calias"],
                                        new_data["lisdependent"], new_data["lishierarchy"],
                                        new_data["lisversioned"], eid), fetch=False)
            self._load_doc_types()
    def delete_doc_type(self):
        item = self.doc_tree.currentItem()
        if not item:
            self.show_warning(self.translator.tr('warning_select_document_type'))
            return
        eid = item.data(0, Qt.UserRole)
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('confirm_delete_document_type')) == QMessageBox.Yes:
            sql = "DELETE FROM meta.entitytypes WHERE id=%s"
            self.db.execute_query(sql, (eid,), fetch=False)
            self._load_doc_types()
    def _load_roles(self):
        sql = "SELECT id, calias, cname FROM meta.composition_roles ORDER BY id"
        rows = self.db.execute_query(sql)
        self.role_table.setRowCount(0)
        for row in rows:
            pos = self.role_table.rowCount()
            self.role_table.insertRow(pos)
            for col, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                if col == 0:
                    item.setData(Qt.UserRole, val)
                self.role_table.setItem(pos, col, item)
    def add_role(self):
        dlg = CompositionRoleEditDialog(self)
        if dlg.exec():
            data = dlg.get_data()
            sql = "INSERT INTO meta.composition_roles (calias, cname, mcomment) VALUES (%s, %s, %s)"
            self.db.execute_query(sql, (data["calias"], data["cname"], data["mcomment"]), fetch=False)
            self._load_roles()
    def edit_role(self):
        row = self.role_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_role'))
            return
        item = self.role_table.item(row, 0)
        rid = item.data(Qt.UserRole)
        sql = "SELECT calias, cname, mcomment FROM meta.composition_roles WHERE id=%s"
        row_data = self.db.execute_query(sql, (rid,))
        if not row_data:
            return
        data = {
            "calias": row_data[0][0],
            "cname": row_data[0][1],
            "mcomment": row_data[0][2]
        }
        dlg = CompositionRoleEditDialog(self, data)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = "UPDATE meta.composition_roles SET calias=%s, cname=%s, mcomment=%s WHERE id=%s"
self.db.execute_query(sql, (new_data["calias"], new_data["cname"], new_data["mcomment"], rid), fetch=False)
            self._load_roles()
    def delete_role(self):
        row = self.role_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_role'))
            return
        item = self.role_table.item(row, 0)
        rid = item.data(Qt.UserRole)
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('confirm_delete_role')) == QMessageBox.Yes:
            sql = "DELETE FROM meta.composition_roles WHERE id=%s"
            self.db.execute_query(sql, (rid,), fetch=False)
            self._load_roles()
    def _load_compositions(self):
        sql = "SELECT c.id, p.cname, ch.cname, r.cname, c.c_link_fieldname
                FROM meta.entity_composition c
                JOIN meta.entitytypes p ON c.parent_entityid = p.id
                JOIN meta.entitytypes ch ON c.child_entityid = ch.id
                JOIN meta.composition_roles r ON c.role_id = r.id
                ORDER BY c.id"
        rows = self.db.execute_query(sql)
        self.comp_table.setRowCount(0)
        for row in rows:
            pos = self.comp_table.rowCount()
            self.comp_table.insertRow(pos)
            for col, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                if col == 0:
                    item.setData(Qt.UserRole, row[0])
                self.comp_table.setItem(pos, col, item)
    def get_doc_types_list(self):
        sql = "SELECT id, cname FROM meta.entitytypes
                WHERE class_id = (SELECT id FROM meta.entity_classes WHERE calias = 'DOC')
                ORDER BY cname"
        return self.db.execute_query(sql)
    def get_child_types_list(self):
        sql = "SELECT id, cname FROM meta.entitytypes ORDER BY cname"
        return self.db.execute_query(sql)
    def get_roles_list(self):
        sql = "SELECT id, cname FROM meta.composition_roles ORDER BY cname"
        return self.db.execute_query(sql)
    def add_composition(self):
        doc_types = self.get_doc_types_list()
        child_types = self.get_child_types_list()
        roles = self.get_roles_list()
        dlg = CompositionEditDialog(self, doc_types=doc_types, child_types=child_types, roles=roles, db=self.db)
        if dlg.exec():
            data = dlg.get_data()
            success = self.composition_manager.add_composition(
                parent_entityid=data["parent_entityid"],
                child_entityid=data["child_entityid"],
                role_id=data["role_id"],
                link_field=data["c_link_fieldname"],
                sortorder=data["isortorder"]
            )
            if success:
                self._load_compositions()
    def edit_composition(self):
        row = self.comp_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_composition'))
            return
        item = self.comp_table.item(row, 0)
        comp_id = item.data(Qt.UserRole)
        sql = "SELECT parent_entityid, child_entityid, role_id, c_link_fieldname, isortorder
                FROM meta.entity_composition WHERE id=%s"
        row_data = self.db.execute_query(sql, (comp_id,))
        if not row_data:
            return
        data = {
            "parent_entityid": row_data[0][0],
            "child_entityid": row_data[0][1],
            "role_id": row_data[0][2],
            "c_link_fieldname": row_data[0][3],
            "isortorder": row_data[0][4]
        }
        doc_types = self.get_doc_types_list()
        child_types = self.get_child_types_list()
        roles = self.get_roles_list()
        dlg = CompositionEditDialog(self, data, doc_types, child_types, roles, db=self.db)
        if dlg.exec():
            new_data = dlg.get_data()
            success = self.composition_manager.update_composition(
                comp_id=comp_id,
                parent_entityid=new_data["parent_entityid"],
```

```
            child_entityid=new_data["child_entityid"],
            role_id=new_data["role_id"],
            link_field=new_data["link_field"],
            sortorder=new_data["isortorder"]
        )
        if success:
            self._load_compositions()
    def delete_composition(self):
        row = self.comp_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_composition'))
            return
        item = self.comp_table.item(row, 0)
        comp_id = item.data(Qt.UserRole)
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
            self.translator.tr('confirm_delete_composition')) == QMessageBox.Yes:
            self.composition_manager.delete_composition(comp_id)
            self._load_compositions()
```

Файл: /home/sergey/Documents/configurate/tabs/entities_tab.py (относительно: tabs/entit

```
# tabs/entities_tab.py
from PyQt5.QtWidgets import (QLabel, QWidget, QVBoxLayout, QHBoxLayout, QSplitter,
                             QTreeWidget, QTreeWidgetItem, QMessageBox, QHeaderView,
                             QDialog, QFormLayout, QLineEdit, QComboBox, QCheckBox,
                             QPushButton, QTextEdit, QAbstractItemView, QTabWidget)
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QBrush
from .base_tab import BaseTab
from .crud_buttons import CrudButtons
from widgets.reference_editor import ReferenceEditor
from .field_edit_dialog import FieldEditDialog
from .entity_subtabs import FieldsTab, BUVISibilityTab, CompositionTab, RoutesTab
from lang.local_translator import LocalTranslator
from .entity_subtabs import FieldsTab, BUVISibilityTab, CompositionTab, RoutesTab, ImportTab
ADDRESS_FIELDS = [
    ("full_address", "C", "Полный адрес", 500, True),
    ("house_id", "C", "Идентификатор ГАР", 36, True),
    ("house", "C", "Дом/строение", 50, False),
    ("building", "C", "Корпус/литера", 50, False),
    ("flat", "C", "Квартира/офис", 50, False),
    ("country", "C", "Страна", 100, False),
    ("region", "C", "Регион", 100, False),
    ("city", "C", "Город/населенный пункт", 150, False),
    ("locality", "C", "Населенный пункт", 150, False),
    ("street", "C", "Улица", 150, False),
    ("additional", "C", "Дополнительно", 100, False),
]

class EntityTypeEditDialog(QDialog):
    def __init__(self, parent=None, data=None, db=None, read_only_mode=False):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_entity_type_edit'))
        self.data = data or {}
        self.db = db
        self.read_only_mode = read_only_mode
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(self.data.get("cname", ""))
        self.edit_alias = QLineEdit(self.data.get("calias", ""))
        self.edit_comment = QTextEdit()
        self.edit_comment.setPlainText(self.data.get("mcomment", ""))
        self.edit_comment.setMaximumHeight(100)
        self.ref_class = ReferenceEditor(
            table_name='meta.entity_classes',
            display_field='cname',
            db=self.db
        )
        if self.data.get("class_id"):
            sql = "SELECT cname FROM meta.entity_classes WHERE id = %s"
            res = self.db.execute_query(sql, (self.data["class_id"],))
            if res:
                self.ref_class.set_value(self.data["class_id"], res[0][0])
        self.edit_baseclass = QLineEdit(self.data.get("cbaseclass", ""))
        self.chk_dependent = QCheckBox(self.translator.tr('label_dependent'))
        self.chk_dependent.setChecked(self.data.get("isdependent", False))
        self.chk_hierarchy = QCheckBox(self.translator.tr('label_hierarchical'))
        self.chk_hierarchy.setChecked(self.data.get("ishierarchy", False))
        self.chk_versioned = QCheckBox(self.translator.tr('label_versioned'))
        self.chk_versioned.setChecked(self.data.get("isversioned", True))
        self.edit_security = QLineEdit(str(self.data.get("isecuritystrategy", 1)))
        self.combo_status = QComboBox()
        statuses = self.db.execute_query("SELECT id, name FROM meta.object_statuses ORDER BY id")
        for sid, sname in statuses:
            self.combo_status.addItem(sname, sid)
        current_status = self.data.get("status_id")
        if current_status:
            idx = self.combo_status.findData(current_status)
            if idx >= 0:
                self.combo_status.setCurrentIndex(idx)
        form.addRow(self.translator.tr('field_system_name'), self.edit_name)
        form.addRow(self.translator.tr('field_alias'), self.edit_alias)
        form.addRow(self.translator.tr('field_description'), self.edit_comment)
        form.addRow(self.translator.tr('field_entity_class'), self.ref_class)
        form.addRow(self.translator.tr('field_baseclass_legacy'), self.edit_baseclass)
        form.addRow(self.chk_dependent)
        form.addRow(self.chk_hierarchy)
        form.addRow(self.chk_versioned)
        form.addRow(self.translator.tr('field_security_strategy'), self.edit_security)
        form.addRow(self.translator.tr('field_status'), self.combo_status)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
        if self.read_only_mode or current_status == 2:
            self.edit_name.setEnabled(False)
            self.edit_alias.setEnabled(False)
            self.ref_class.setReadOnly(True)
            self.edit_baseclass.setEnabled(False)
            self.chk_dependent.setEnabled(False)
            self.chk_hierarchy.setEnabled(False)
            self.chk_versioned.setEnabled(False)
            self.edit_security.setEnabled(False)
            self.edit_comment.setEnabled(True)
            self.combo_status.setEnabled(True)
    def get_data(self):
        return {
            "cname": self.edit_name.text(),
            "calias": self.edit_alias.text(),
            "mcomment": self.edit_comment.toPlainText(),
            "cbaseclass": self.edit_baseclass.text(),
            "isdependent": self.chk_dependent.isChecked(),
            "ishierarchy": self.chk_hierarchy.isChecked(),
            "isversioned": self.chk_versioned.isChecked(),
            "isecuritystrategy": int(self.edit_security.text()) if self.edit_security.text().isdigit() else 1,
            "class_id": self.ref_class.get_value(),
            "status_id": self.combo_status.currentData()
        }

class EntitiesTab(BaseTab):
    def __init__(self, parent=None, db=None):
        super().__init__(parent, db)
        self.translator = LocalTranslator()
        splitter = QSplitter(Qt.Horizontal)
        self.layout.addWidget(splitter)
        # Левая панель: список сущностей
```

```
        left_w = QWidget()
        left_v = QVBoxLayout(left_w)
        self.left_label = QLabel("<b>" + self.translator.tr('entity_types') + "</b>")
        left_v.addWidget(self.left_label)
        self.ent_crud = CrudButtons(self)
        left_v.addWidget(self.ent_crud)
        self.tree_entities = QTreeWidget()
self.tree_entities.setHeaderLabels([self.translator.tr('field_system_name'), self.translator.tr('field_alias')])
        self.tree_entities.setColumnWidth(0, 30)
        self.tree_entities.itemClicked.connect(self.on_entity_selected)
        left_v.addWidget(self.tree_entities)
        splitter.addWidget(left_w)
        # Правая панель: закладки
        right_w = QWidget()
        right_layout = QVBoxLayout(right_w)
        right_layout.setContentsMargins(0, 0, 0, 0)
        self.tabs = QTabWidget()
        right_layout.addWidget(self.tabs)
        self.fields_tab = FieldsTab(self, db=self.db)
        self.tabs.addTab(self.fields_tab, self.translator.tr('tab_fields'))
        self.bu_tab = BUVISibilityTab(self, db=self.db)
        self.tabs.addTab(self.bu_tab, self.translator.tr('tab_bu_visibility'))
        self.composition_tab = CompositionTab(self, db=self.db)
        self.tabs.addTab(self.composition_tab, self.translator.tr('tab_composition'))
        self.routes_tab = RoutesTab(self, db=self.db)
        self.tabs.addTab(self.routes_tab, self.translator.tr('tab_routes'))
        self.import_tab = ImportTab(self, db=self.db)
        self.tabs.addTab(self.import_tab, self.translator.tr('tab_import'))
        splitter.addWidget(right_w)
        splitter.setSizes([400, 600])
        self.ent_crud.add_clicked.connect(self.add_entity)
        self.ent_crud.edit_clicked.connect(self.edit_entity)
        self.ent_crud.delete_clicked.connect(self.delete_selected_entities)
        self.current_entity_id = None
        self.refresh()
        self.retranslate_ui()
    def retranslate_ui(self):
        # Обновляем заголовок левой панели
        self.left_label.setText("<b>" + self.translator.tr('entity_types') + "</b>")
        # Заголовки дерева
self.tree_entities.setHeaderLabels([self.translator.tr('field_system_name'), self.translator.tr('field_alias')])
        # Заголовки вкладок
        self.tabs.setTabText(0, self.translator.tr('tab_fields'))
        self.tabs.setTabText(1, self.translator.tr('tab_bu_visibility'))
        self.tabs.setTabText(2, self.translator.tr('tab_composition'))
        self.tabs.setTabText(3, self.translator.tr('tab_routes'))
        # Кнопки CRUD
        self.ent_crud.retranslate_ui()
        # Передать дочерним вкладкам
        self.fields_tab.retranslate_ui()
        self.bu_tab.retranslate_ui()
        self.composition_tab.retranslate_ui()
        self.routes_tab.retranslate_ui()
        if hasattr(self, 'import_tab'):
            self.import_tab.retranslate_ui()
        self.tabs.setTabText(4, self.translator.tr('tab_import'))
    def refresh(self):
        self._load_entities()
        self._update_right_panels(None)
    def _load_entities(self):
        sql = "SELECT id, cname, calias, status_id FROM meta.entitytypes ORDER BY cname"
        rows = self.db.execute_query(sql)
        self.tree_entities.clear()
        for eid, cname, calias, status_id in rows:
            item = QTreeWidgetItem(self.tree_entities)
            item.setFlags(item.flags() | Qt.ItemIsUserCheckable)
            item.setCheckState(0, Qt.Unchecked)
            if status_id == 2:
                for col in (1, 2):
                    item.setForeground(col, QBrush(Qt.gray))
            item.setText(1, cname)
            item.setText(2, calias)
            item.setData(0, Qt.UserRole, eid)
            item.setData(0, Qt.UserRole + 1, status_id)
    def on_entity_selected(self, item):
        self.current_entity_id = item.data(0, Qt.UserRole)
        self._update_right_panels(self.current_entity_id)
    def _update_right_panels(self, entity_id):
        self.fields_tab.set_entity_id(entity_id)
        self.bu_tab.set_entity_id(entity_id)
        self.composition_tab.set_entity_id(entity_id)
        self.routes_tab.set_entity_id(entity_id)
        self.import_tab.set_entity_id(entity_id)
    def add_entity(self):
        dlg = EntityTypeEditDialog(self, db=self.db)
        if dlg.exec():
            data = dlg.get_data()
            sql = """INSERT INTO meta.entitytypes
(cname, calias, mcomment, cbaseclass, lisdependent, lishierarchy, lisversioned, isecuritystrategy, class_id,
status_id)     VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
            self.db.execute_query(sql, (data["cname"], data["calias"], data["mcomment"], data["cbaseclass"],
                                        data["lisdependent"], data["lishierarchy"],
                                        data["lisversioned"], data["isecuritystrategy"],
                                        data["class_id"], data["status_id"]), fetch=False)
            self._load_entities()
            if data["class_id"] is not None:
                class_alias = self.db.execute_query(
                    "SELECT calias FROM meta.entity_classes WHERE id = %s", (data["class_id"],)
                )
                if class_alias and class_alias[0][0] == 'ADDRESS':
                    self._add_address_fields(data["calias"])
    def _add_address_fields(self, entity_alias):
        res = self.db.execute_query("SELECT id FROM meta.entitytypes WHERE calias = %s", (entity_alias,))
        if not res:
            return
        entity_id = res[0][0]
        for fname, ftype, alias, length, indexed in ADDRESS_FIELDS:
            exists = self.db.execute_query(
                "SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s", (entity_id, fname)
            )
            if not exists:
                res = self.db.execute_query("""
                    INSERT INTO meta.fields
                        (entitytypeid, cfieldname, cfieldtype, calias, nlength, lisindexed, status_id)
                    VALUES (%s, %s, %s, %s, %s, %s, 1) RETURNING id
                """, (entity_id, fname, ftype, alias, length, indexed))
                if res:
                    field_id = res[0][0]
                    try:
                        self.db.execute_procedure("meta.p_field_sync", (field_id,))
                    except Exception as e:
                        self.show_error("Ошибка синхронизации поля {fname}: {e}")
        if self.current_entity_id == entity_id:
            self.fields_tab.load_fields()
            self._update_right_panels(entity_id)
    def edit_entity(self):
        item = self.tree_entities.currentItem()
        if not item:
            self.show_warning(self.translator.tr('warning_select_entity'))
            return
        eid = item.data(0, Qt.UserRole)
        status_id = item.data(0, Qt.UserRole + 1)
sql = """SELECT cname, calias, mcomment, cbaseclass, lisdependent, lishierarchy, lisversioned,
isecuritystrategy, class_id, status_id
        FROM meta.entitytypes WHERE id=%s"""
        row = self.db.execute_query(sql, (eid,))
        if not row:
            return
        old_data = {
            "cname": row[0][0],
```

```
            "calias": row[0][1],
            "mcomment": row[0][2],
            "cbaseclass": row[0][3],
            "lisdependent": row[0][4],
            "lishierarchy": row[0][5],
            "lisversioned": row[0][6],
            "securitystrategy": row[0][7],
            "class_id": row[0][8],
            "status_id": row[0][9]
        }
        read_only = (status_id == 2)
        dlg = EntityTypeEditDialog(self, old_data, db=self.db, read_only_mode=read_only)
        if not dlg.exec_():
            return
        new_data = dlg.get_data()
        old_cname = old_data["cname"].strip().lower()
        new_cname = new_data["cname"].strip().lower()
        if old_cname != new_cname and status_id != 2:
            reply = QMessageBox.question(
                self,
                self.translator.tr('confirm_rename_title'),
                self.translator.tr('confirm_rename_message').format(old_cname, new_cname),
                QMessageBox.Yes | QMessageBox.No
            )
            if reply == QMessageBox.No:
                return
            check_sql = """
                SELECT table_name
                FROM information_schema.tables
                WHERE table_schema='ext' AND table_name=%s"""
            old_table = f"ext_{old_cname}"
            table_exists = self.db.execute_query(check_sql, (old_table,))
            if table_exists:
                rename_sql = f"ALTER TABLE ext.{old_table} RENAME TO ext_{new_cname}"
                try:
                    self.db.execute_query(rename_sql, fetch=False)
                    self.show_info(self.translator.tr('info_table_renamed').format(old_table, new_cname))
                except Exception as e:
                    self.show_error(self.translator.tr('error_rename_table').format(e))
                    return
            else:
                self.show_warning(self.translator.tr('warning_table_not_found').format(old_table))
        elif old_cname != new_cname and status_id == 2:
            self.show_error(self.translator.tr('error_rename_prod'))
            return
        sql = """UPDATE meta.entitytypes SET
                cname=%s, calias=%s, mcomment=%s, cbaseclass=%s, lisdependent=%s,
                lishierarchy=%s, lisversioned=%s, isecuritystrategy=%s, class_id=%s, status_id=%s
                WHERE id=%s"""
        self.db.execute_query(sql, (new_data["cname"], new_data["calias"], new_data["mcomment"],
new_data["cbaseclass"],
                                    new_data["lisdependent"], new_data["lishierarchy"],
                                    new_data["lisversioned"], new_data["isecuritystrategy"],
                                    new_data["class_id"], new_data["status_id"], eid), fetch=False)
        self.load_entities()
        if self.current_entity_id == eid:
            self._update_right_panels(eid)
    def delete_selected_entities(self):
        items = self.tree_entities.selectedItems()
        if not items:
            self.show_warning(self.translator.tr('warning_select_entity'))
            return
        dev_ids = []
        prod_names = []
        for item in items:
            eid = item.data(0, Qt.UserRole)
            status_id = item.data(0, Qt.UserRole + 1)
            if status_id == 1:
                dev_ids.append(eid)
            else:
                prod_names.append(item.text(1))
        if prod_names:
            self.show_error(self.translator.tr('error_prod_entity_delete').format(', '.join(prod_names)))
            return
        if not dev_ids:
            return
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('confirm_delete_entities').format(len(dev_ids)),
                                QMessageBox.Yes | QMessageBox.No) == QMessageBox.Yes:
            for eid in dev_ids:
                self.db.execute_query("DELETE FROM meta.entitytypes WHERE id=%s", (eid,), fetch=False)
            self.load_entities()
            if self.current_entity_id in dev_ids:
                self._update_right_panels(None)
                self.current_entity_id = None
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/__init__.py (относительно

```
# tabs/entity_subtabs/__init__.py
from .fields_tab import FieldsTab
from .bu_visibility_tab import BUVISibilityTab
from .composition_tab import CompositionTab
from .routes_tab import RoutesTab
from .import_tab import ImportTab   # раскомментируем

__all__ = [
    'FieldsTab',
    'BUVISibilityTab',
    'CompositionTab',
    'RoutesTab',
    'ImportTab',
]
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/bu_visibility_tab.py (относ

```
# tabs/entity_subtabs/bu_visibility_tab.py
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QTableWidget, QTableWidgetItem,
                            QHeaderView, QMessageBox, QComboBox, QDialog,
                            QFormLayout, QDialogButtonBox)
from PyQt5.QtCore import Qt
from database import DatabaseService
from crud_buttons import CrudButtons
from lang.local_translator import LocalTranslator

class BUVISibilityTab(QWidget):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.translator = LocalTranslator()
        self.current_entity_id = None
        layout = QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        self.crud = CrudButtons(self, with_add=True, with_edit=False, with_delete=True)
        layout.addWidget(self.crud)
        self.table = QTableWidget(0, 4)
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        self.table.setEditTriggers(QTableWidget.NoEditTriggers)
        layout.addWidget(self.table)
        self.crud.add_clicked.connect(self.add_visibility)
        self.crud.delete_clicked.connect(self.delete_selected)
        self.retranslate_ui()
    def retranslate_ui(self):
        headers = [
            self.translator.tr('field_id'),
            self.translator.tr('field_balance_unit'),
            self.translator.tr('field_policy'),
        ]
        self.table.setHorizontalHeaderLabels(headers)
        self.table.horizontalHeader().setStretchLastSection(True)
        if hasattr(self, 'crud'):
```

```
        self.crud.retranslate_ui()
        self.load_data()
    def set_entity_id(self, entity_id):
        self.current_entity_id = entity_id
        self.load_data()
    def load_data(self):
        if not self.current_entity_id:
            self.table.setRowCount(0)
            return
        sql = """
            SELECT id, balanceunit_id, visibility_policy
            FROM meta.entitytype_visibility
            WHERE entitytype_id = %s
            ORDER BY id
        """
        rows = self.db.execute_query(sql, (self.current_entity_id,))
        self.table.setRowCount(0)
        policy_map = {
            'I': self.translator.tr('policy_individual'),
            'A': self.translator.tr('policy_all_visible'),
            'N': self.translator.tr('policy_all_hidden')
        }
        for r_idx, (vid, buid, policy) in enumerate(rows):
            self.table.insertRow(r_idx)
            # ID
            self.table.setItem(r_idx, 0, QTableWidgetItem(str(vid)))
            # Балансовая единица
            bu_name = self.get_bu_name(buid)
            self.table.setItem(r_idx, 1, QTableWidgetItem(f"{bu_name} ({buid})"))
            # Политика
            self.table.setItem(r_idx, 2, QTableWidgetItem(policy_map.get(policy, policy)))
            self.table.item(r_idx, 0).setData(Qt.UserRole, vid)
    def get_bu_name(self, buid):
        res = self.db.execute_query("SELECT cname FROM auth.balanceunits WHERE id=%s", (buid,))
        return res[0][0] if res else str(buid)
    def get_bus_list(self):
        return self.db.execute_query("SELECT id, cname FROM auth.balanceunits ORDER BY cname")
    def add_visibility(self):
        if not self.current_entity_id:
            QMessageBox.warning(self, self.translator.tr('error'), self.translator.tr('warning_select_entity'))
            return
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('title_add_visibility'))
        layout = QVBoxLayout(dlg)
        form = QFormLayout()
        bu_combo = QComboBox()
        for buid, buname in self.get_bus_list():
            bu_combo.addItem(f"{buname} ({buid})", buid)
        form.addRow(self.translator.tr('field_balance_unit'), bu_combo)
        policy_combo = QComboBox()
        policy_combo.addItem(self.translator.tr('policy_individual'), 'I')
        policy_combo.addItem(self.translator.tr('policy_all_visible'), 'A')
        policy_combo.addItem(self.translator.tr('policy_all_hidden'), 'N')
        form.addRow(self.translator.tr('field_policy'), policy_combo)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.button(QDialogButtonBox.Ok).setText(self.translator.tr('btn_ok'))
        btn_box.button(QDialogButtonBox.Cancel).setText(self.translator.tr('btn_cancel'))
        btn_box.accepted.connect(dlg.accept)
        btn_box.rejected.connect(dlg.reject)
        layout.addWidget(btn_box)
        if dlg.exec_() == QDialog.Accepted:
            buid = bu_combo.currentData()
            policy = policy_combo.currentData()
            check = self.db.execute_query(
                "SELECT id FROM meta.entitytype_visibility WHERE entitytype_id=%s AND balanceunit_id=%s",
                (self.current_entity_id, buid))
            if check:
                QMessageBox.warning(self, self.translator.tr('error'), self.translator.tr('error_visibility_exists'))
                return
sql = "INSERT INTO meta.entitytype_visibility (entitytype_id, balanceunit_id, visibility_policy) VALUES (%s, %s,
%s)"
            self.db.execute_query(sql, (self.current_entity_id, buid, policy), fetch=False)
            self.load_data()
    def delete_selected(self):
        selected = self.table.selectedItems()
        if not selected:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('warning_select_record'))
            return
        rows = set(item.row() for item in selected)
        ids = []
        for row in rows:
            vid_item = self.table.item(row, 0)
            if vid_item:
                ids.append(int(vid_item.text()))
        if not ids:
            return
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('confirm_delete_records').format(ids),
                                QMessageBox.Yes | QMessageBox.No) == QMessageBox.Yes:
            for vid in ids:
                self.db.execute_query("DELETE FROM meta.entitytype_visibility WHERE id=%s", (vid,), fetch=False)
            self.load_data()
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/composition_tab.py (относ

```
# tabs/entity_subtabs/composition_tab.py
from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QTableWidget, QTableWidgetItem,
                             QHeaderView, QMessageBox, QTabWidget, QDialog,
                             QFormLayout, QComboBox, QLineEdit, QDialogButtonBox, QLabel)
from PyQt5.QtCore import Qt
from database import DatabaseService
from crud_buttons import CrudButtons
from composition_manager import CompositionManager
from lang.local_translator import LocalTranslator

class CompositionTab(QWidget):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.translator = LocalTranslator()
        self.composition_manager = CompositionManager(db=self.db, parent_widget=self)
        self.current_entity_id = None
        layout = QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        self.tabs = QTabWidget()
        layout.addWidget(self.tabs)
        self.parent_tab = self._create_parent_tab()
        self.tabs.addTab(self.parent_tab, self.translator.tr('tab_as_parent'))
        self.child_tab = self._create_child_tab()
        self.tabs.addTab(self.child_tab, self.translator.tr('tab_as_child'))
        self.retranslate_ui()
    def retranslate_ui(self):
        self.tabs.setTabText(0, self.translator.tr('tab_as_parent'))
        self.tabs.setTabText(1, self.translator.tr('tab_as_child'))
        # Обновляем заголовки таблиц
        self.parent_table.setHorizontalHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_detail'),
            self.translator.tr('field_role'),
            self.translator.tr('field_link_field'),
            self.translator.tr('field_order'),
        ])
        self.child_table.setHorizontalHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_master'),
            self.translator.tr('field_role'),
            self.translator.tr('field_link_field'),
            self.translator.tr('field_order'),
        ])
        self.parent_crud.retranslate_ui()
```

```
        self.child_crud.retranslate_ui()
    def _create_parent_tab(self):
        tab = QWidget()
        layout = QVBoxLayout(tab)
        layout.setContentsMargins(0, 0, 0, 0)
        self.parent_crud = CrudButtons(self, with_add=True, with_edit=True, with_delete=True)
        layout.addWidget(self.parent_crud)
        self.parent_table = QTableWidget(0, 5)
        self.parent_table.setHorizontalHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_detail'),
            self.translator.tr('field_role'),
            self.translator.tr('field_link_field'),
            self.translator.tr('field_order'),
        ])
        self.parent_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.parent_table.setSelectionBehavior(QTableWidget.SelectRows)
        self.parent_table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.parent_table.itemDoubleClicked.connect(lambda: self.edit_composition('parent'))
        layout.addWidget(self.parent_table)
        self.parent_crud.add_clicked.connect(lambda: self.add_composition('parent'))
        self.parent_crud.edit_clicked.connect(lambda: self.edit_composition('parent'))
        self.parent_crud.delete_clicked.connect(lambda: self.delete_composition('parent'))
        return tab
    def _create_child_tab(self):
        tab = QWidget()
        layout = QVBoxLayout(tab)
        layout.setContentsMargins(0, 0, 0, 0)
        self.child_crud = CrudButtons(self, with_add=False, with_edit=True, with_delete=True)
        layout.addWidget(self.child_crud)
        self.child_table = QTableWidget(0, 5)
        self.child_table.setHorizontalHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_master'),
            self.translator.tr('field_role'),
            self.translator.tr('field_link_field'),
            self.translator.tr('field_order'),
        ])
        self.child_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.child_table.setSelectionBehavior(QTableWidget.SelectRows)
        self.child_table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.child_table.itemDoubleClicked.connect(lambda: self.edit_composition('child'))
        layout.addWidget(self.child_table)
        self.child_crud.edit_clicked.connect(lambda: self.edit_composition('child'))
        self.child_crud.delete_clicked.connect(lambda: self.delete_composition('child'))
        return tab
    def set_entity_id(self, entity_id):
        self.current_entity_id = entity_id
        self.load_parent()
        self.load_child()
    def load_parent(self):
        if not self.current_entity_id:
            self.parent_table.setRowCount(0)
            return
        sql = """
            SELECT c.id, et.cname, cr.cname, c.c_link_fieldname, c.isortorder
            FROM meta.entity_composition c
            JOIN meta.entitytypes et ON c.child_entityid = et.id
            JOIN meta.composition_roles cr ON c.role_id = cr.id
            WHERE c.parent_entityid = %s
        """ORDER BY c.id
        rows = self.db.execute_query(sql, (self.current_entity_id,))
        self.parent_table.setRowCount(0)
        for r_idx, (cid, child, role, field, order) in enumerate(rows):
            self.parent_table.insertRow(r_idx)
            self.parent_table.setItem(r_idx, 0, QTableWidgetItem(str(cid)))
            self.parent_table.setItem(r_idx, 1, QTableWidgetItem(child))
            self.parent_table.setItem(r_idx, 2, QTableWidgetItem(role))
            self.parent_table.setItem(r_idx, 3, QTableWidgetItem(field))
            self.parent_table.setItem(r_idx, 4, QTableWidgetItem(str(order)))
            self.parent_table.item(r_idx, 0).setData(Qt.UserRole, cid)
    def load_child(self):
        if not self.current_entity_id:
            self.child_table.setRowCount(0)
            return
        sql = """
            SELECT c.id, et.cname, cr.cname, c.c_link_fieldname, c.isortorder
            FROM meta.entity_composition c
            JOIN meta.entitytypes et ON c.parent_entityid = et.id
            JOIN meta.composition_roles cr ON c.role_id = cr.id
            WHERE c.child_entityid = %s
        """ORDER BY c.id
        rows = self.db.execute_query(sql, (self.current_entity_id,))
        self.child_table.setRowCount(0)
        for r_idx, (cid, master, role, field, order) in enumerate(rows):
            self.child_table.insertRow(r_idx)
            self.child_table.setItem(r_idx, 0, QTableWidgetItem(str(cid)))
            self.child_table.setItem(r_idx, 1, QTableWidgetItem(master))
            self.child_table.setItem(r_idx, 2, QTableWidgetItem(role))
            self.child_table.setItem(r_idx, 3, QTableWidgetItem(field))
            self.child_table.setItem(r_idx, 4, QTableWidgetItem(str(order)))
            self.child_table.item(r_idx, 0).setData(Qt.UserRole, cid)
    def add_composition(self, role):
        if role == 'parent':
            if not self.current_entity_id:
                QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('warning_select_entity'))
                return
            dlg = QDialog(self)
            dlg.setWindowTitle(self.translator.tr('title_add_composition'))
            layout = QVBoxLayout(dlg)
            form = QFormLayout()
entities = self.db.execute_query("SELECT id, cname FROM meta.entitytypes WHERE id != %s ORDER BY
cname",
                                 (self.current_entity_id,))
            child_combo = QComboBox()
            for eid, ename in entities:
                child_combo.addItem(f"{ename} ({eid})", eid)
            form.addRow(self.translator.tr('field_detail'), child_combo)
            roles = self.db.execute_query("SELECT id, cname FROM meta.composition_roles ORDER BY cname")
            role_combo = QComboBox()
            for rid, rname in roles:
                role_combo.addItem(f"{rname} ({rid})", rid)
            form.addRow(self.translator.tr('field_role'), role_combo)
            field_edit = QLineEdit()
            form.addRow(self.translator.tr('field_link_field'), field_edit)
            order_edit = QLineEdit("10")
            form.addRow(self.translator.tr('field_order'), order_edit)
            layout.addLayout(form)
            btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
            btn_box.accepted.connect(dlg.accept)
            btn_box.rejected.connect(dlg.reject)
            layout.addWidget(btn_box)
            if dlg.exec() == QDialog.Accepted:
                child_id = child_combo.currentData()
                role_id = role_combo.currentData()
                field = field_edit.text()
                order = int(order_edit.text()) if order_edit.text().isdigit() else 10
                # Используем CompositionManager
                success = self.composition_manager.add_composition(
                    parent_entityid=self.current_entity_id,
                    child_entityid=child_id,
                    role_id=role_id,
                    link_field=field,
                    sortorder=order
                )
                if success:
                    self.load_parent()
        else:
            QMessageBox.information(self, self.translator.tr('info'), self.translator.tr('info_add_as_child_not_supported'))
    def edit_composition(self, mode):
        table = self.parent_table if mode == 'parent' else self.child_table
        selected = table.selectedItems()
        if not selected:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('warning_select_record'))
            return
        row = selected[0].row()
        cid = table.item(row, 0).data(Qt.UserRole)
        sql = """SELECT parent_entityid, child_entityid, role_id, c_link_fieldname, isortorder
                 FROM meta.entity_composition WHERE id=%s"""
        row_data = self.db.execute_query(sql, (cid,))
        if not row_data:
            return
```

```
        parent, child, role, field, order = row_data[0]
        dlg = QDialog(self)
        dlg.setWindowTitle(self.translator.tr('title_edit_composition'))
        layout = QVBoxLayout(dlg)
        form = QFormLayout()
        if mode == 'parent':
            entities = self.db.execute_query("SELECT id, cname FROM meta.entitytypes WHERE id != %s ORDER BY cname",
                                             (self.current_entity_id,))
            child_combo = QComboBox()
            for eid, ename in entities:
                child_combo.addItem(f"{ename} ({eid})", eid)
            idx = child_combo.findData(child)
            if idx >= 0: child_combo.setCurrentIndex(idx)
            form.addRow(self.translator.tr('field_detail'), child_combo)
        else:
            parent_name = self.db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (parent,))[0][0]
            form.addRow(self.translator.tr('field_master'), QLabel(f"{parent_name} ({parent})"))
        roles = self.db.execute_query("SELECT id, cname FROM meta.composition_roles ORDER BY cname")
        role_combo = QComboBox()
        for rid, rname in roles:
            role_combo.addItem(f"{rname} ({rid})", rid)
        idx = role_combo.findData(role)
        if idx >= 0: role_combo.setCurrentIndex(idx)
        form.addRow(self.translator.tr('field_role'), role_combo)
        field_edit = QLineEdit(field)
        form.addRow(self.translator.tr('field_link_field'), field_edit)
        order_edit = QLineEdit(str(order))
        form.addRow(self.translator.tr('field_order'), order_edit)
        layout.addLayout(form)
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(dlg.accept)
        btn_box.rejected.connect(dlg.reject)
        layout.addWidget(btn_box)
        if dlg.exec_() == QDialog.Accepted:
            new_role = role_combo.currentData()
            new_field = field_edit.text()
            new_order = int(order_edit.text()) if order_edit.text().isdigit() else 10
            if mode == 'parent':
                new_child = child_combo.currentData()
                # Используем CompositionManager
                success = self.composition_manager.update_composition(
                    comp_id=cid,
                    parent_entityid=self.current_entity_id,
                    child_entityid=new_child,
                    role_id=new_role,
                    link_field=new_field,
                    sortorder=new_order
                )
                if success:
                    self.load_parent()
            else:
                success = self.composition_manager.update_composition(
                    comp_id=cid,
                    parent_entityid=parent,
                    child_entityid=self.current_entity_id,
                    role_id=new_role,
                    link_field=new_field,
                    sortorder=new_order
                )
                if success:
                    self.load_child()
    def delete_composition(self, mode):
        table = self.parent_table if mode == 'parent' else self.child_table
        selected = table.selectedItems()
        if not selected:
            QMessageBox.warning(self, self.translator.tr('warning'), self.translator.tr('warning_select_record'))
            return
        rows = set(item.row() for item in selected)
        ids = []
        for row in rows:
            cid_item = table.item(row, 0)
            if cid_item:
                ids.append(cid_item.data(Qt.UserRole))
        if not ids:
            return
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('confirm_delete_records').%(len(ids)),
                                QMessageBox.Yes | QMessageBox.No) == QMessageBox.Yes:
            for cid in ids:
                self.composition_manager.delete_composition(cid)
            self.load_parent()
            self.load_child()
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/fields_tab.py (относитель

```
# tabs/entity_subtabs/fields_tab.py
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QTableWidget, QTableWidgetItem,
                            QHeaderView, QMessageBox
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QBrush
from database.database_service
from crud_buttons import CrudButtons
from ide_field_edit_dialog import IdeFieldEditDialog
from lang.local_translator import LocalTranslator

class FieldsTab(QWidget):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.translator = LocalTranslator()
        self.current_entity_id = None
        layout = QVBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        self.crud = CrudButtons(self)
        layout.addWidget(self.crud)
        self.table = QTableWidget(0, 9)
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        self.table.setSelectionMode(QTableWidget.ExtendedSelection)
        self.table.setEditTriggers(QTableWidget.NoEditTriggers)
        layout.addWidget(self.table)
        self.crud.add_clicked.connect(self.add_field)
        self.crud.edit_clicked.connect(self.edit_field)
        self.crud.delete_clicked.connect(self.delete_selected_fields)
        self.retranslate_ui()
    def retranslate_ui(self):
        headers = [
            # чекбокс
            self.translator.tr('field_name'),
            self.translator.tr('field_type'),
            self.translator.tr('field_alias'),
            self.translator.tr('field_description'),
            self.translator.tr('field_length'),
            self.translator.tr('field_decimals'),
            self.translator.tr('field_index'),
            self.translator.tr('field_ref_type')
        ]
        self.table.setHorizontalHeaderLabels(headers)
        self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        if hasattr(self.crud):
            self.crud.retranslate_ui()
    def set_entity_id(self, entity_id):
        self.current_entity_id = entity_id
        self.load_fields()
    def load_fields(self):
        if not self.current_entity_id:
            self.table.setRowCount(0)
            return
        sql = """
            SELECT f.cfieldname, t.cname, f.calias, f.mcomment,
                   f.cdecimals, f.nfieldlength, f.lusindex,
                   CASE WHEN f.ref_entitytypeid IS NOT NULL
                        THEN (SELECT cname FROM meta.entitytypes WHERE id = f.ref_entitytypeid)
                        ELSE NULL END as ref_name,
                   f.cfieldtype, f.ref_entitytypeid, f.status_id
            FROM meta.fields f
            JOIN meta.field_types t ON f.cfieldtype = t.ccode
            WHERE f.entitytypeid = %s ORDER BY f.isortorder
```

```
        rows = self.db.execute_query(sql, (self.current_entity_id,))
        self.table.setRowCount(0)
        for r_idx, row in enumerate(rows):
            self.table.insertRow(r_idx)
            # Колонка 0 - lexoks
            chk_item = QTableWidgetItem()
            chk_item.setFlags(Qt.ItemIsUserCheckable | Qt.ItemIsEnabled)
            if row[10] == 2:  # PROD
                chk_item.setFlags(chk_item.flags() & ~Qt.ItemIsEnabled)
                chk_item.setBackground(QBrush(Qt.lightGray))
            chk_item.setCheckState(Qt.Unchecked)
            chk_item.setData(Qt.UserRole, row[10])
            self.table.setItem(r_idx, 0, chk_item)
            # Остальные колонки (1..8)
            for c_idx in range(8):
                if c_idx == 7:  # колонка RefID – показываем имя сущности
                    val = row[7] if row[7] else ""
                else:
                    val = row[c_idx]
                item = QTableWidgetItem(str(val) if val is not None else "")
                if c_idx == 1:  # ТИП
                    item.setData(Qt.UserRole, row[8])
                self.table.setItem(r_idx, c_idx + 1, item)
    def add_field(self):
        if not self.current_entity_id:
            QMessageBox.warning(self, self.translator.tr('error'), self.translator.tr('warning_select_entity'))
            return
        dlg = FieldEditDialog(db=self.db)
        if dlg.exec_():
            data = dlg.get_data()
            self._save_field(data)
    def edit_field(self):
        selected = self.table.selectedItems()
        if not selected:
            QMessageBox.warning(self, self.translator.tr('error'), self.translator.tr('warning_select_field'))
            return
        row = selected[0].row()
        old_name = self.table.item(row, 1).text()
        type_code = self.table.item(row, 2).data(Qt.UserRole)
        data = [
            old_name,
            type_code,
            self.table.item(row, 3).text(),
            self.table.item(row, 4).text(),
            self.table.item(row, 5).text(),
            self.table.item(row, 6).text(),
            self.table.item(row, 7).text(),
            self.table.item(row, 8).text()
        ]
        chk_item = self.table.item(row, 0)
        status_id = chk_item.data(Qt.UserRole) if chk_item else 1
        read_only = (status_id == 2)
        dlg = FieldEditDialog(data, status_id=status_id, read_only_mode=read_only, db=self.db)
        if dlg.exec_():
            new_data = dlg.get_data()
            self._save_field(new_data, is_update=True, old_name=old_name)
    def delete_selected_fields(self):
        selected_rows = set()
        prod_names = []
        dev_ids = []
        for item in self.table.selectedItems():
            row = item.row()
            if row not in selected_rows:
                selected_rows.add(row)
                chk_item = self.table.item(row, 0)
                status_id = chk_item.data(Qt.UserRole) if chk_item else 1
                fname = self.table.item(row, 1).text()
                if status_id == 1:
                    dev_ids.append((self.current_entity_id, fname))
                else:
                    prod_names.append(fname)
        if prod_names:
            QMessageBox.critical(self, self.translator.tr('error'),
                                 self.translator.tr('error_prod_field_delete') + "\n" + ", ".join(prod_names))
            return
        if not dev_ids:
            return
        if QMessageBox.question(self, self.translator.tr('confirm_delete'),
                                self.translator.tr('confirm_delete_fields') % len(dev_ids),
                                QMessageBox.Yes | QMessageBox.No) == QMessageBox.Yes:
            for eid, fname in dev_ids:
                self.db.execute_query("DELETE FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s",
                                      (eid, fname), fetch=False)
            self.load_fields()
    def _save_field(self, field_data, is_update=False, old_name=None):
        entitytypeid = self.current_entity_id
        if not entitytypeid:
            QMessageBox.warning(self, self.translator.tr('error'), self.translator.tr('warning_select_entity'))
            return
        cfieldname = field_data[0]
        cfieldtype = field_data[1]
        calias = field_data[2]
        mcomment = field_data[3]
        nlength = field_data[4]
        ndecimals = field_data[5]
        lsindexed = field_data[6]
        ref_entitytypeid = field_data[7]
        status_id = field_data[8]
        try:
            if is_update:
# Проверка, не изменилось ли имя поля? Если изменилось – нужно переименовать колонку в ext-таблице
                if old_name and old_name != cfieldname:
                    # Получаем имя таблицы расширения
                    sql_table = "SELECT cname FROM meta.entitytypes WHERE id = %s"
                    res = self.db.execute_query(sql_table, (entitytypeid,))
                    if res:
                        ext_table = f"ext.ext_{res[0][0]}"
                        # Переименовываем колонку
                        rename_sql = f"ALTER TABLE {ext_table} RENAME COLUMN {old_name} TO {cfieldname}"
                        self.db.execute_query(rename_sql, fetch=False)
                # Обновляем запись в meta.fields
                sql = """UPDATE meta.fields SET
                        cfieldname=%s, cfieldtype=%s, calias=%s, mcomment=%s,
                        nlength=%s, ndecimals=%s, lsindexed=%s, ref_entitytypeid=%s, status_id=%s
                    WHERE entitytypeid=%s AND cfieldname=%s"""
                self.db.execute_query(sql, (cfieldname, cfieldtype, calias, mcomment,
                                            nlength, ndecimals, lsindexed, ref_entitytypeid, status_id,
                                            entitytypeid, old_name), fetch=False)
            else:
                # Вставка нового поля
                sql = """INSERT INTO meta.fields
                        (entitytypeid, cfieldname, cfieldtype, calias, mcomment,
                         nlength, ndecimals, lsindexed, ref_entitytypeid, status_id)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s) RETURNING id"""
                res = self.db.execute_query(sql, (entitytypeid, cfieldname, cfieldtype, calias, mcomment,
                                                  nlength, ndecimals, lsindexed, ref_entitytypeid, status_id))
                if res:
                    field_id = res[0][0]
                    # Синхронизация с ext-таблицей
                    self.db.execute_procedure("meta.p_field_sync", (field_id,))
            self.load_fields()
        except Exception as e:
            QMessageBox.critical(self, self.translator.tr('error'), self.translator.tr('error_save_field') + str(e))
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/import_engine.py (относит

```
# import_tab.py
import os
from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QHBoxLayout, QFormLayout,
                             QLabel, QComboBox, QLineEdit, QPushButton,
                             QTableWidget, QTableWidgetItem, QHeaderView,
                             QMessageBox, QFileDialog, QGroupBox, QCheckBox,
                             QDialogButtonBox)
from PyQt5.QtCore import Qt
from database.import_DatabaseService
from crud_buttons import CrudButtons
from import_models import SourceSettings, Mapping
from import_engine import ImportEngine
from import_utils import load_source_settings
```

```
try:
    from dbfread import DBF
    DBF_AVAILABLE = True
except ImportError:
    DBF_AVAILABLE = False

class SourceCheckDialog(QDialog):
    """Диалог для проверки источника и выбора полей для маппинга."""
    def __init__(self, parent=None, db=None, entity_id=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.entity_id = entity_id
        self.source_fields = []  # список полей из источника
        self.mappings = []  # список объектов Mapping
        self.setWindowTitle("Проверка источника и сопоставление полей")
        self.resize(900, 600)
        self.init_ui()
    def init_ui(self):
        layout = QVBoxLayout(self)
        # Выбор файла/URL и формата
        form = QFormLayout()
        self.source_type_combo = QComboBox()
        self.source_type_combo.addItem("Файл", 'F')
        self.source_type_combo.addItem("URL", 'U')
        self.source_type_combo.currentIndexChanged.connect(self.on_source_type_changed)
        form.addRow("Тип источника:", self.source_type_combo)
        self.source_path_edit = QLineEdit()
        self.btn_browse = QPushButton("Обзор...")
        self.btn_browse.clicked.connect(self.browse_file)
        path_layout = QHBoxLayout()
        path_layout.addWidget(self.source_path_edit)
        path_layout.addWidget(self.btn_browse)
        form.addRow("Путь/URL:", path_layout)
        self.format_combo = QComboBox()
        self.format_combo.addItem("JSON", 'J')
        self.format_combo.addItem("XML", 'X')
        self.format_combo.addItem("DBF", 'D')
        form.addRow("Формат:", self.format_combo)
        self.id_field_combo = QComboBox()
        self.id_field_combo.setEnabled(False)
        form.addRow("Поле-идентификатор (sourceid):", self.id_field_combo)
        self.btn_check = QPushButton("Проверить источник")
        self.btn_check.clicked.connect(self.check_source)
        form.addRow(self.btn_check)
        layout.addLayout(form)
        # Таблица сопоставления полей
        self.mapping_table = QTableWidget(0, 4)
self.mapping_table.setHorizontalHeaderLabels(["Поле сущности", "Поле источника", "Ссылка", "Целевая
сущность"])
        self.mapping_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        layout.addWidget(QLabel("Сопоставление полей:"))
        layout.addWidget(self.mapping_table)
        # Кнопки OK/Cancel
        btn_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        btn_box.accepted.connect(self.accept)
        btn_box.rejected.connect(self.reject)
        layout.addWidget(btn_box)
    def on_source_type_changed(self, index):
        is_file = self.source_type_combo.currentData() == 'F'
        self.btn_browse.setEnabled(is_file)
        if not is_file:
            self.source_path_edit.clear()
    def browse_file(self):
        path = QFileDialog.getOpenFileName(self, "Выберите файл источника")
        if path:
            self.source_path_edit.setText(path)
    def check_source(self):
        source_type = self.source_type_combo.currentData()
        path = self.source_path_edit.text().strip()
        fmt = self.format_combo.currentData()
        if not path:
            QMessageBox.warning(self, "Ошибка", "Укажите путь к источнику")
            return
        try:
            if fmt == 'J':
                self.source_fields = self.read_json_structure(path)
            elif fmt == 'X':
                self.source_fields = self.read_xml_structure(path)
            elif fmt == 'D':
                self.source_fields = self.read_dbf_structure(path)
            else:
                QMessageBox.critical(self, "Ошибка", "Неподдерживаемый формат")
                return
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось прочитать источник: {str(e)}")
            return
        if not self.source_fields:
            QMessageBox.warning(self, "Предупреждение", "Не удалось получить поля источника")
            return
        # Обновляем комбобокс для идентификатора
        self.id_field_combo.clear()
        self.id_field_combo.addItem("-- не выбрано --", None)
        for f in self.source_fields:
            self.id_field_combo.addItem(f, f)
        self.id_field_combo.setEnabled(True)
        # Получаем список полей сущности (системные + из meta.fields)
        entity_fields = self.get_entity_fields()
        self.mapping_table.setRowCount(len(entity_fields))
        for row, (field_name, display_name) in enumerate(entity_fields):
            # Колонка 0: название поля сущности (только для чтения)
            self.mapping_table.setItem(row, 0, QTableWidgetItem(display_name))
            # Колонка 1: выпадающий список полей источника
            combo = QComboBox()
            combo.addItem("-- не сопоставлено --", None)
            for s in self.source_fields:
                combo.addItem(s, s)
            self.mapping_table.setCellWidget(row, 1, combo)
            # Колонка 2: чекбокс "Ссылка"
            chk = QCheckBox()
            chk.stateChanged.connect(lambda state, r=row: self.on_reference_toggled(r, state))
            self.mapping_table.setCellWidget(row, 2, chk)
            # Колонка 3: выпадающий список целевых сущностей (изначально disabled)
            ref_combo = QComboBox()
            ref_combo.setEnabled(False)
            ref_combo.addItem("-- не выбрано --", None)
            entity_types = self.get_entity_types_list()
            for etid, etname in entity_types:
                ref_combo.addItem(f"{etname} (Letid})", etid)
            self.mapping_table.setCellWidget(row, 3, ref_combo)
            # Сохраняем имя поля для последующего использования
            self.mapping_table.item(row, 0).setData(Qt.UserRole, field_name)
        QMessageBox.information(self, "Информация", f"Найдено {len(self.source_fields)} полей в источнике")
    def on_reference_toggled(self, row, state):
        """При включении чекбокса активируем выбор целевой сущности."""
        ref_combo = self.mapping_table.cellWidget(row, 3)
        ref_combo.setEnabled(state == Qt.Checked)
    def read_json_structure(self, path):
        with open(path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        if isinstance(data, list) and len(data) > 0:
            return list(data[0].keys())
        elif isinstance(data, dict):
            return list(data.keys())
        else:
            return []
    def read_xml_structure(self, path):
        tree = ET.parse(path)
        root = tree.getroot()
        if len(root) > 0:
```

```
        return [child.tag for child in root[0]]
    return []
def read_dbf_structure(self, path):
    if not DBF_AVAILABLE:
        raise ImportError("Библиотека dbfread не установлена")
    table = DBF(path, ignore_missing_memofile=True)
    return [field.name for field in table.fields]
def get_entity_fields(self):
    """Возвращает список полей сущности в формате [(field_name, display_name), ...]."""
    fields = []
    # Системные поля
    fields.append(('cname', 'Наименование (cname)'))
    fields.append(('parentinstanceid', 'Родитель (parentinstanceid)'))
    # Поля из meta.fields
    sql = """SELECT cfieldname, calias FROM meta.fields
             WHERE entitytypeid = %s ORDER BY sortorder"""
    rows = self.db.execute_query(sql, (self.entity_id,))
    for cfieldname, calias in rows:
        fields.append((cfieldname, f"{calias} ({cfieldname})"))
    return fields
def get_entity_types_list(self):
    sql = "SELECT id, cname FROM meta.entitytypes ORDER BY cname"
    return self.db.execute_query(sql)
def get_mappings(self):
    """Возвращает список объектов Mapping."""
    mappings = []
    # source_id
    id_field = self.id_field_combo.currentData()
    if id_field:
        mappings.append(Mapping(target_field='sourceid', source_field=id_field))
    for row in range(self.mapping_table.rowCount()):
        target = self.mapping_table.item(row, 0).data(Qt.UserRole)
        combo = self.mapping_table.cellWidget(row, 1)
        source = combo.currentData()
        if source is None:
            continue
        chk = self.mapping_table.cellWidget(row, 2)
        is_ref = (chk.checkState() == Qt.Checked)
        ref_entitytype_id = None
        ref_field_name = None
        if is_ref:
            ref_combo = self.mapping_table.cellWidget(row, 3)
            ref_entitytype_id = ref_combo.currentData()
            # Генерируем имя ссылочного поля автоматически
            ref_field_name = f"ref_{source}"
        mappings.append(Mapping(
            target_field=target,
            source_field=source,
            is_reference=is_ref,
            ref_entitytype_id=ref_entitytype_id,
            ref_field_name=ref_field_name
        ))
    return mappings

class ImportTab(QWidget):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.current_entity_id = None
        self.source_id = None  # id записи в import_sources для текущей сущности
        layout = QVBoxLayout(self)
        layout.setContentsMargins(5, 5, 5, 5)
        # Группа параметров источника
        self.source_group = QGroupBox("Настройки источника")
        source_layout = QFormLayout(self.source_group)
        self.source_name_edit = QLineEdit()
        source_layout.addRow("Название источника:", self.source_name_edit)
        self.source_type_combo = QComboBox()
        self.source_type_combo.addItem("Файл", 'F')
        self.source_type_combo.addItem("URL", 'U')
        self.source_type_combo.currentIndexChanged.connect(self.on_source_type_changed)
        source_layout.addRow("Тип:", self.source_type_combo)
        self.source_path_edit = QLineEdit()
        self.btn_browse = QPushButton("Обзор...")
        self.btn_browse.clicked.connect(self.browse_file)
        path_layout = QHBoxLayout()
        path_layout.addWidget(self.source_path_edit)
        path_layout.addWidget(self.btn_browse)
        source_layout.addRow("Путь/URL:", path_layout)
        self.format_combo = QComboBox()
        self.format_combo.addItem("JSON", 'J')
        self.format_combo.addItem("XML", 'X')
        self.format_combo.addItem("DBF", 'D')
        source_layout.addRow("Формат:", self.format_combo)
        self.btn_check = QPushButton("Проверить и настроить сопоставление")
        self.btn_check.clicked.connect(self.check_source)
        source_layout.addRow(self.btn_check)
        layout.addWidget(self.source_group)
        # Кнопки сохранения и импорта
        btn_layout = QHBoxLayout()
        self.btn_save = QPushButton("Сохранить настройки")
        self.btn_save.clicked.connect(self.save_settings)
        self.btn_import = QPushButton("Запустить импорт")
        self.btn_import.clicked.connect(self.run_import)
        btn_layout.addWidget(self.btn_save)
        btn_layout.addWidget(self.btn_import)
        btn_layout.addStretch()
        layout.addLayout(btn_layout)
        # Статус последнего импорта
        self.status_label = QLabel("Последний импорт: никогда")
        layout.addWidget(self.status_label)
        layout.addStretch()
    def on_source_type_changed(self, index):
        is_file = self.source_type_combo.currentData() == 'F'
        self.btn_browse.setEnabled(is_file)
        if not is_file:
            self.source_path_edit.clear()
    def browse_file(self):
        path, _ = QFileDialog.getOpenFileName(self, "Выберите файл источника")
        if path:
            self.source_path_edit.setText(path)
    def set_entity_id(self, entity_id):
        self.current_entity_id = entity_id
        self.load_settings()
    def load_settings(self):
        """Загружает настройки импорта для текущей сущности."""
        if not self.current_entity_id:
            return
        sql = """SELECT id, source_name, source_type, source_path, format, id_field, last_import
                 FROM meta.import_sources WHERE entitytypeid = %s"""
        rows = self.db.execute_query(sql, (self.current_entity_id,))
        if rows:
            self.source_id = rows[0][0]
            self.source_name_edit.setText(rows[0][1] or "")
            self.source_type_combo.setCurrentIndex(self.source_type_combo.findData(rows[0][2]))
            self.source_path_edit.setText(rows[0][3])
            self.format_combo.setCurrentIndex(self.format_combo.findData(rows[0][4]))
            if rows[0][6]:
                self.status_label.setText(f"Последний импорт: {rows[0][6]}")
            else:
                self.status_label.setText("Последний импорт: никогда")
        else:
            self.source_id = None
            self.source_name_edit.clear()
            self.source_path_edit.clear()
            self.format_combo.setCurrentIndex(0)
            self.status_label.setText("Последний импорт: никогда")
    def check_source(self):
        if not self.current_entity_id:
            QMessageBox.warning(self, "Внимание", "Сначала выберите сущность")
```

```
            return
        dlg = SourceCheckDialog(self.db, self.current_entity_id)
        # Передаем текущие данные в диалог
        dlg.source_type_combo.setCurrentIndex(self.source_type_combo.currentIndex())
        dlg.source_path_edit.setText(self.source_path_edit.text())
        dlg.format_combo.setCurrentIndex(self.format_combo.currentIndex())
        if dlg.exec_() == QDialog.Accepted:
            self.last_mappings = dlg.get_mappings()
QMessageBox.information(self, "Информация", "Сопоставление полей выполнено. Нажмите 'Сохранить настройки' для сохранения.")
    def save_settings(self):
        if not self.current_entity_id:
            QMessageBox.warning(self, "Внимание", "Сначала выберите сущность")
            return
        if not hasattr(self, 'last_mappings') or not self.last_mappings:
QMessageBox.warning(self, "Внимание", "Сначала выполните проверку источника и сопоставление полей")
            return
        source_type = self.source_type_combo.currentData()
        source_path = self.source_path_edit.text().strip()
        fmt = self.format_combo.currentData()
        source_name = self.source_name_edit.text().strip()
        if not source_path:
            QMessageBox.warning(self, "Ошибка", "Укажите путь к источнику")
            return
        try:
            # Находим id_field (первый маппинг с target='sourceid')
            id_field = None
            for m in self.last_mappings:
                if m.target_field == 'sourceid':
                    id_field = m.source_field
                    break
            # Сохраняем или обновляем запись в import_sources
            if self.source_id:
                sql = """UPDATE meta.import_sources SET
source_name=%s, source_type=%s, source_path=%s, format=%s, id_field=%s, updated_at=now()
                    WHERE id=%s"""
self.db.execute_query(sql, (source_name, source_type, source_path, fmt, id_field, self.source_id), fetch=False)
            else:
                sql = """INSERT INTO meta.import_sources
                    (entitytypeid, source_name, source_type, source_path, format, id_field)
                    VALUES (%s, %s, %s, %s, %s, %s) RETURNING id"""
res = self.db.execute_query(sql, (self.current_entity_id, source_name, source_type, source_path, fmt, id_field))
                if res:
                    self.source_id = res[0][0]
            # Удаляем старые маппинги
self.db.execute_query("DELETE FROM meta.import_mappings WHERE import_source_id = %s", (self.source_id,), fetch=False)
# Для каждого маппинга создаём или обновляем поле в meta.fields (если нужно) и вставляем запись в import_mappings
            for mapping in self.last_mappings:
                if mapping.target_field == 'sourceid':
                    # sourceid уже создается автоматически, просто сохраняем маппинг
                    pass
                elif mapping.is_reference:
# Для ссылочного поля убеждаемся, что поле с именем ref_field_name существует и имеет тип 'R'
                    # Проверяем, есть ли уже такое поле
                    sql_check = "SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname=%s"
                    exists = self.db.execute_query(sql_check, (self.current_entity_id, mapping.ref_field_name))
                    if not exists:
                        # Создаём новое поле типа 'R'
                        sql_insert = """INSERT INTO meta.fields
(entitytypeid, cfieldname, cfieldtype, calias, nlength, ndecimals, lisindexed, ref_entitytypeid, status_id)
                            VALUES (%s, %s, 'R', %s, 0, 0, false, %s, 1) RETURNING id"""
target_entity = self.db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (mapping.ref_entitytype_id,))
self.db.execute_query(sql_insert, (self.current_entity_id, mapping.ref_field_name, alias, mapping.ref_entitytype_id), fetch=False)
                        # Сохраняем маппинг: target = ref_field_name
                        target = mapping.ref_field_name
                else:
                    target = mapping.target_field
                # Вставляем запись в import_mappings (для sourceid тоже вставляем, но is_reference=false)
                sql_ins_map = """INSERT INTO meta.import_mappings
                    (import_source_id, target_field, source_field, is_reference, ref_entitytypeid, ref_field_name)
                    VALUES (%s, %s, %s, %s, %s, %s)"""
self.db.execute_query(sql_ins_map, (self.source_id, target, mapping.source_field, mapping.is_reference, mapping.ref_entitytype_id, mapping.ref_field_name), fetch=False)
            QMessageBox.information(self, "Информация", "Настройки импорта сохранены")
            self.load_settings() # обновим статус
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить настройки: {str(e)}")
    def run_import(self):
        if not self.current_entity_id or not self.source_id:
            QMessageBox.warning(self, "Внимание", "Сначала сохраните настройки импорта")
            return
        # Загружаем настройки из БД
        settings = load_source_settings(self.db, self.source_id)
        if not settings:
            QMessageBox.critical(self, "Ошибка", "Не удалось загрузить настройки импорта")
            return
        # Добавляем id в settings для использования в engine (для обновления last_import)
        settings.id = self.source_id
        engine = ImportEngine(self.db, settings, self.current_entity_id)
        try:
            # В будущем можно добавить прогресс-бар и выполнение в отдельном потоке
            engine.run()
            QMessageBox.information(self, "Информация", "Импорт успешно завершён")
            self.load_settings() # обновим статус
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Импорт не удался: {str(e)}")
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/import_models.py (относи

```
# import_models.py
from dataclasses import dataclass
from typing import Optional, List

@dataclass
class Mapping:
    """Сопоставление одного поля"""
    target_field: str        # имя поля в сущности (или специальное 'sourceid')
    source_field: str        # поле в источнике
    is_reference: bool = False
    ref_entitytype_id: Optional[int] = None   # для ссылок
    ref_field_name: Optional[str] = None      # имя создаваемого ссылочного поля

@dataclass
class SourceSettings:
    """Настройки источника импорта"""
    source_type: str         # 'F' или 'D'
    source_path: str
    format: str              # 'J', 'X', 'D'
    source_name: str = ""
    id_field: str = ""       # поле в источнике, которое будет sourceid
    mappings: List[Mapping] = None
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/import_readers.py (относи

```
# import_readers.py
import csv
import xml.etree.ElementTree as ET
from abc import ABC, abstractmethod
from typing import List, Dict, Any, Iterator
try:
```

```
    from dbfread import DBF
    DBF_AVAILABLE = True
except ImportError:
    DBF_AVAILABLE = False

class BaseReader(ABC):
    """Абстрактный читатель источника данных."""
    @abstractmethod
    def get_fields(self) -> List[str]:
        """Возвращает список имён полей в источнике."""
        pass
    @abstractmethod
    def __iter__(self) -> Iterator[Dict[str, Any]]:
        """Итератор по записям, каждая запись – словарь поле->значение."""
        pass

class JsonReader(BaseReader):
    def __init__(self, path: str):
        self.path = path
        with open(path, 'r', encoding='utf-8') as f:
            data = json.load(f)
        if isinstance(data, list):
            self.records = data
        else:
            self.records = [data]
    def get_fields(self) -> List[str]:
        if self.records:
            return list(self.records[0].keys())
        return []
    def __iter__(self) -> Iterator[Dict[str, Any]]:
        for rec in self.records:
            yield rec

class XmlReader(BaseReader):
    def __init__(self, path: str):
        self.path = path
        self.tree = ET.parse(path)
        self.root = self.tree.getroot()
        # Предполагаем, что записи – это дочерние элементы корня
        self.records = list(self.root)
    def get_fields(self) -> List[str]:
        if self.records:
            # Собираем все возможные теги из первого элемента
            return [child.tag for child in self.records[0]]
        return []
    def __iter__(self) -> Iterator[Dict[str, Any]]:
        for elem in self.records:
            rec = {child.tag: child.text for child in elem}
            yield rec

class DbfReader(BaseReader):
    def __init__(self, path: str):
        if not DBF_AVAILABLE:
            raise ImportError("Библиотека dbfread не установлена. Установите: pip install dbfread")
        self.table = DBF(path, ignore_missing_memofile=True)
        self.records = list(self.table)
    def get_fields(self) -> List[str]:
        return [field.name for field in self.table.fields]
    def __iter__(self) -> Iterator[Dict[str, Any]]:
        for rec in self.records:
            yield dict(rec)
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/import_tab.py (относител

```
# tabs/entity_subtabs/import_tab.py
# -*- coding: utf-8 -*-
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton
from database.import_db_service import DatabaseService
from lang.local_translator import LocalTranslator
from mozart_import.dialogs.profile_manager_dialog import ProfileManagerDialog

class ImportTab(QWidget):
    """Вкладка импорта для конкретной сущности."""
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.translator = LocalTranslator()
        self.current_entity_id = None
        layout = QVBoxLayout(self)
        layout.setContentsMargins(5, 5, 5, 5)
        self.btn_manage = QPushButton(self.translator.tr('btn_start_import_designer'))
        self.btn_manage.clicked.connect(self.open_profile_manager)
        layout.addWidget(self.btn_manage)
        layout.addStretch()
        self.retranslate_ui()
    def retranslate_ui(self):
        self.btn_manage.setText(self.translator.tr('btn_start_import_designer'))
    def set_entity_id(self, entity_id):
        self.current_entity_id = entity_id
    def open_profile_manager(self):
        """Открывает диалог управления профилями импорта."""
        dlg = ProfileManagerDialog(self.db, self)
        dlg.exec_()
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/import_utils.py (относите

```
# import_utils.py
from database.import_DatabaseService
from typing import Dict, Optional, List, Tuple

def get_entity_cname(db: DatabaseService, entitytype_id: int) -> Optional[str]:
    """Возвращает системное имя сущности (cname) по её ID."""
    res = db.execute_query("SELECT cname FROM meta.entitytypes WHERE id=%s", (entitytype_id,))
    return res[0][0] if res else None

def ensure_sourceid_field(db: DatabaseService, entitytype_id: int, field_type: str = 'C', length: int = 100) ->
None:
    """
    Проверяет наличие поля sourceid в ext-таблице и в meta.fields, создаёт при необходимости.
    """
    # Проверяем в meta.fields
    res = db.execute_query("SELECT id FROM meta.fields WHERE entitytypeid=%s AND cfieldname='sourceid'",
(entitytype_id,))
    if not res:
        # Создаём поле
        sql = """INSERT INTO meta.fields
                 (entitytypeid, cfieldname, cfieldtype, calias, nlength, lisindexed, status_id)
                 VALUES (%s, 'sourceid', %s, 'Source ID', %s, true, 1) RETURNING id"""
        new_id = db.execute_query(sql, (entitytype_id, field_type, length))
        if new_id:
            # Вызываем синхронизацию
            db.execute_procedure("meta.p_field_sync", (new_id[0][0],))
    # Поле уже существует – ничего не делаем

def load_reference_mappings(db: DatabaseService, ref_entitytype_id: int) -> Dict[str, int]:
    """
    Загружает маппинг sourceid -> instanceid для указанной сущности.
    Возвращает словарь {sourceid_value: instanceid}.
    """
    cname = get_entity_cname(db, ref_entitytype_id)
    if not cname:
        return {}
    ext_table = f"ext.ext_{cname}"
    sql = f"""
```

```
        SELECT e.sourceid, v.instanceid
        FROM {text_table} e
        JOIN meta.entity_versions v ON v.id = e.versionid
    """WHERE e.sourceid IS NOT NULL
    rows = db.execute_query(sql)
    return {str(row[0]): row[1] for row in rows}

def load_source_settings(db: DatabaseService, import_source_id: int) -> Optional['SourceSettings']:
    """
    Загружает настройки из БД в объект SourceSettings.
    from import_models import SourceSettings, Mapping
    # Загружаем основную запись
    sql_src = """SELECT source_type, source_path, format, source_name, id_field
                 FROM meta.import_sources WHERE id = %s"""
    src_row = db.execute_query(sql_src, (import_source_id,))
    if not src_row:
        return None
    source_type, source_path, fmt, source_name, id_field = src_row[0]
    # Загружаем маппинги
    sql_map = """SELECT target_field, source_field, is_reference, ref_entitytypeid, ref_field_name
                 FROM meta.import_mappings
                 WHERE import_source_id = %s
                 ORDER BY id"""
    map_rows = db.execute_query(sql_map, (import_source_id,))
    mappings = []
    for row in map_rows:
        target, source, is_ref, ref_etype, ref_fname = row
        mappings.append(Mapping(
            target_field=target,
            source_field=source,
            is_reference=is_ref,
            ref_entitytype_id=ref_etype,
            ref_field_name=ref_fname
        ))
    return SourceSettings(
        source_type=source_type,
        source_path=source_path,
        format=fmt,
        source_name=source_name,
        id_field=id_field,
        mappings=mappings
    )
```

Файл: /home/sergey/Documents/configurate/tabs/entity_subtabs/routes_tab.py (относитель

```
# tabs/entity_subtabs/routes_tab.py
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QLabel
from database.import DatabaseService
from lang.local_translator import LocalTranslator

class RoutesTab(QWidget):
    def __init__(self, parent=None, db=None):
        super().__init__(parent)
        self.db = db or DatabaseService()
        self.translator = LocalTranslator()
        self.current_entity_id = None
        layout = QVBoxLayout(self)
        self.label_status = QLabel(self.translator.tr('routes_in_development'))
        layout.addWidget(self.label_status)
        self.retranslate_ui()
    def retranslate_ui(self):
        self.label_status.setText(self.translator.tr('routes_in_development'))
    def set_entity_id(self, entity_id):
        self.current_entity_id = entity_id
```

Файл: /home/sergey/Documents/configurate/tabs/relations_tab.py (относительно: tabs/rela

```
# tabs/relations_tab.py
from PyQt5.QtWidgets import (QWidget, QTabWidget, QTreeWidget, QTreeWidgetItem, QTableWidget,
                             QTableWidgetItem, QPushButton, QHBoxLayout, QVBoxLayout,
                             QMessageBox, QDialog, QFormLayout, QLineEdit,
                             QComboBox, QLabel, QHeaderView)
from PyQt5.QtCore import Qt
from base_tab import BaseTab
from crud_buttons import CrudButtons

class RefTypeEditDialog(QDialog):
    def __init__(self, parent=None, data=None):
        super().__init__(parent)
        self.setWindowTitle("Тип связи")
        self.data = data or {}
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(self.data.get("cname", ""))
        self.edit_alias = QLineEdit(self.data.get("calias", ""))
        self.edit_comment = QLineEdit(self.data.get("mcomment", ""))
        form.addRow("Наименование:", self.edit_name)
        form.addRow("Алиас:", self.edit_alias)
        form.addRow("Комментарий:", self.edit_comment)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton("OK")
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton("Отмена")
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "cname": self.edit_name.text(),
            "calias": self.edit_alias.text(),
            "mcomment": self.edit_comment.text()
        }

class RefRuleEditDialog(QDialog):
    def __init__(self, parent=None, data=None, ref_types=None, entity_types=None):
        super().__init__(parent)
        self.setWindowTitle("Правило связи")
        self.data = data or {}
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.combo_ref_type = QComboBox()
        for tid, tname in (ref_types or []):
            self.combo_ref_type.addItem(f"{tname} ({tid})", tid)
        self.combo_entity = QComboBox()
        for eid, ename in (entity_types or []):
            self.combo_entity.addItem(f"{ename} ({eid})", eid)
        self.combo_role_source = QComboBox()
        for eid, ename in (entity_types or []):
            self.combo_role_source.addItem(f"{ename} ({eid})", eid)
        if data:
            idx = self.combo_ref_type.findData(data.get("reftypeid"))
            if idx >= 0: self.combo_ref_type.setCurrentIndex(idx)
            idx = self.combo_entity.findData(data.get("entitytypeid"))
            if idx >= 0: self.combo_entity.setCurrentIndex(idx)
            idx = self.combo_role_source.findData(data.get("role_source_entityid"))
            if idx >= 0: self.combo_role_source.setCurrentIndex(idx)
        form.addRow("Тип связи:", self.combo_ref_type)
        form.addRow("Тип сущности участника:", self.combo_entity)
        form.addRow("Источник ролей:", self.combo_role_source)
```

```
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton("OK")
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton("Отмена")
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "reltypeid": self.combo_ref_type.currentData(),
            "entityTypeid": self.combo_entity.currentData(),
            "role_source_entityid": self.combo_role_source.currentData()
        }

class RelationsTab(BaseTab):
    def __init__(self, parent=None, db=None):
        super().__init__(parent, db)
        self.tabs = QTabWidget()
        self.layout.addWidget(self.tabs)
        # Вкладка "Типы связей"
        self.types_tab = self._create_types_tab()
        self.tabs.addTab(self.types_tab, "Типы связей")
        # Вкладка "Правила"
        self.rules_tab = self._create_rules_tab()
        self.tabs.addTab(self.rules_tab, "Правила")
        # Вкладка "Экземпляры связей" (заглушка)
        self.instances_tab = self._create_instances_tab()
        self.tabs.addTab(self.instances_tab, "Экземпляры связей")
    def _create_types_tab(self):
        tab = QWidget()
        layout = QVBoxLayout(tab)
        self.type_crud = CrudButtons(self)
        layout.addWidget(self.type_crud)
        self.type_table = QTableWidget()
        self.type_table.setColumnCount(3)
        self.type_table.setHorizontalHeaderLabels(["ID", "Алиас", "Наименование"])
        self.type_table.horizontalHeader().setStretchLastSection(True)
        self.type_table.setSelectionBehavior(QTableWidget.SelectRows)
        self.type_table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.type_table.itemDoubleClicked.connect(self.edit_type)
        layout.addWidget(self.type_table)
        self.type_crud.add_clicked.connect(self.add_type)
        self.type_crud.edit_clicked.connect(self.edit_type)
        self.type_crud.delete_clicked.connect(self.delete_type)
        self._load_types()
        return tab
    def _create_rules_tab(self):
        tab = QWidget()
        layout = QVBoxLayout(tab)
        self.rule_crud = CrudButtons(self)
        layout.addWidget(self.rule_crud)
        self.rule_table = QTableWidget()
        self.rule_table.setColumnCount(4)
        self.rule_table.setHorizontalHeaderLabels(["ID", "Тип связи", "Тип участника", "Источник ролей"])
        self.rule_table.horizontalHeader().setStretchLastSection(True)
        self.rule_table.setSelectionBehavior(QTableWidget.SelectRows)
        self.rule_table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.rule_table.itemDoubleClicked.connect(self.edit_rule)
        layout.addWidget(self.rule_table)
        self.rule_crud.add_clicked.connect(self.add_rule)
        self.rule_crud.edit_clicked.connect(self.edit_rule)
        self.rule_crud.delete_clicked.connect(self.delete_rule)
        self._load_rules()
        return tab
    def _create_instances_tab(self):
        tab = QWidget()
        layout = QVBoxLayout(tab)
        layout.addWidget(QLabel("Раздел в разработке: просмотр и редактирование экземпляров связей."))
        return tab
    # -- Типы связей --
    def _load_types(self):
        sql = "SELECT id, calias, cname FROM meta.entity_ref_types ORDER BY id"
        rows = self.db.execute_query(sql)
        self.type_table.setRowCount(0)
        for row in rows:
            pos = self.type_table.rowCount()
            self.type_table.insertRow(pos)
            for col, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                if col == 0:
                    item.setData(Qt.UserRole, val)
                self.type_table.setItem(pos, col, item)
    def add_type(self):
        dlg = RefTypeEditDialog(self)
        if dlg.exec():
            data = dlg.get_data()
            sql = "INSERT INTO meta.entity_ref_types (calias, cname, mcomment) VALUES (%s, %s, %s)"
            self.db.execute_query(sql, (data["calias"], data["cname"], data["mcomment"]), fetch=False)
            self._load_types()
    def edit_type(self):
        row = self.type_table.currentRow()
        if row < 0:
            self.show_warning("Выберите тип связи")
            return
        item = self.type_table.item(row, 0)
        tid = item.data(Qt.UserRole)
        sql = "SELECT calias, cname, mcomment FROM meta.entity_ref_types WHERE id=%s"
        row_data = self.db.execute_query(sql, (tid,))
        if not row_data:
            return
        data = {
            "calias": row_data[0][0],
            "cname": row_data[0][1],
            "mcomment": row_data[0][2]
        }
        dlg = RefTypeEditDialog(self, data)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = "UPDATE meta.entity_ref_types SET calias=%s, cname=%s, mcomment=%s WHERE id=%s"
self.db.execute_query(sql, (new_data["calias"], new_data["cname"], new_data["mcomment"], tid), fetch=False)
            self._load_types()
    def delete_type(self):
        row = self.type_table.currentRow()
        if row < 0:
            self.show_warning("Выберите тип связи")
            return
        item = self.type_table.item(row, 0)
        tid = item.data(Qt.UserRole)
        if QMessageBox.question(self, "Подтверждение", "Удалить тип связи?") == QMessageBox.Yes:
            sql = "DELETE FROM meta.entity_ref_types WHERE id=%s"
            self.db.execute_query(sql, (tid,), fetch=False)
            self._load_types()
    # -- Правила --
    def _load_rules(self):
        sql = "SELECT r.id, rt.cname, et.cname, rs.cname
                FROM meta.entity_ref_rules r
                JOIN meta.entity_ref_types rt ON r.reftypeid = rt.id
                JOIN meta.entitytypes et ON r.entityTypeid = et.id
                JOIN meta.entitytypes rs ON r.role_source_entityid = rs.id
                ORDER BY r.id"
        rows = self.db.execute_query(sql)
        self.rule_table.setRowCount(0)
        for row in rows:
            pos = self.rule_table.rowCount()
            self.rule_table.insertRow(pos)
            for col, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                if col == 0:
                    item.setData(Qt.UserRole, row[0])
                self.rule_table.setItem(pos, col, item)
    def get_ref_types_list(self):
        sql = "SELECT id, cname FROM meta.entity_ref_types ORDER BY cname"
        return self.db.execute_query(sql)
```

```
    def get_entity_types_list(self):
        sql = "SELECT id, cname FROM meta.entitytypes ORDER BY cname"
        return self.db.execute_query(sql)
    def add_rule(self):
        ref_types = self.get_ref_types_list()
        entity_types = self.get_entity_types_list()
        dlg = RefRuleEditDialog(self, ref_types=ref_types, entity_types=entity_types)
        if dlg.exec():
            data = dlg.get_data()
            sql = "INSERT INTO meta.entity_ref_rules (reftypeid, entitytypeid, role_source_entityid)
VALUES (%s, %s, %s)"
self.db.execute_query(sql, (data["reftypeid"], data["entitytypeid"], data["role_source_entityid"]), fetch=False)
            self._load_rules()
    def edit_rule(self):
        row = self.rule_table.currentRow()
        if row < 0:
            self.show_warning("Выберите правило")
            return
        item = self.rule_table.item(row, 0)
        rid = item.data(Qt.UserRole)
        sql = "SELECT reftypeid, entitytypeid, role_source_entityid FROM meta.entity_ref_rules WHERE id=%s"
        row_data = self.db.execute_query(sql, (rid,))
        if not row_data:
            return
        data = {
            "reftypeid": row_data[0][0],
            "entitytypeid": row_data[0][1],
            "role_source_entityid": row_data[0][2]
        }
        ref_types = self.get_ref_types_list()
        entity_types = self.get_entity_types_list()
        dlg = RefRuleEditDialog(self, data, ref_types, entity_types)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = "UPDATE meta.entity_ref_rules SET
                  reftypeid=%s, entitytypeid=%s, role_source_entityid=%s
                  WHERE id=%s"
            self.db.execute_query(sql, (new_data["reftypeid"], new_data["entitytypeid"],
                                        new_data["role_source_entityid"], rid), fetch=False)
            self._load_rules()
    def delete_rule(self):
        row = self.rule_table.currentRow()
        if row < 0:
            self.show_warning("Выберите правило")
            return
        item = self.rule_table.item(row, 0)
        rid = item.data(Qt.UserRole)
        if QMessageBox.question(self, "Подтверждение", "Удалить правило?") == QMessageBox.Yes:
            sql = "DELETE FROM meta.entity_ref_rules WHERE id=%s"
            self.db.execute_query(sql, (rid,), fetch=False)
            self._load_rules()
```

Файл: /home/sergey/Documents/configurate/tabs/user_roles/__init__.py (относительно: tab

Файл: /home/sergey/Documents/configurate/tabs/user_roles/postgres_sync.py (относитель

Файл: /home/sergey/Documents/configurate/tabs/user_roles/user_edit_dialog.py (относите

```
# tabs/users_roles/user_edit_dialog.py
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QFormLayout, QLineEdit, QCheckBox, QPushButton,
QHBoxLayout, QMessageBox
from lang.local_translator import LocalTranslator
from database import DatabaseService
class UserEditDialog(QDialog):
    def __init__(self, parent=None, data=None, db=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_user_edit'))
        self.data = data or {}
        self.db = db or DatabaseService()
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_login = QLineEdit(self.data.get("clogin", ""))
        self.edit_pass = QLineEdit(self.data.get("cpassword_hash", ""))
        self.edit_pass.setEchoMode(QLineEdit.Password)
        self.chk_active = QCheckBox(self.translator.tr('label_active'))
        self.chk_active.setChecked(self.data.get("lactive", True))
        self.edit_person = QLineEdit(str(self.data.get("person_instanceid", "")))
        form.addRow(self.translator.tr('field_login'), self.edit_login)
        form.addRow(self.translator.tr('field_password'), self.edit_pass)
        form.addRow(self.translator.tr('field_person_instance'), self.edit_person)
        form.addRow(self.chk_active)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "clogin": self.edit_login.text().strip(),
            "cpassword_hash": self.edit_pass.text(),   # открытый пароль, потом захешируем
            "lactive": self.chk_active.isChecked(),
            "person_instanceid": self.edit_person.text().strip() or None
        }
```

Файл: /home/sergey/Documents/configurate/tabs/user_roles/users_roles_tab.py (относител

Файл: /home/sergey/Documents/configurate/tabs/user_roles/users_tab.py (относительно: t

Файл: /home/sergey/Documents/configurate/tabs/users_roles_tab.py (относительно: tabs/u

```
# tabs/users_roles_tab.py
# tabs/users_roles_tab.py
from PyQt5.QtWidgets import (QWidget, QTabWidget, QTreeWidget, QTreeWidgetItem, QTableWidget,
                             QTableWidgetItem, QPushButton, QHBoxLayout, QVBoxLayout,
                             QMessageBox, QDialog, QFormLayout, QLineEdit,
                             QComboBox, QCheckBox, QLabel, QHeaderView)
from PyQt5.QtCore import Qt
from base_tab import BaseTab
from crud_buttons import CrudButtons
from widgets import ReferenceEditor
from lang.local_translator import LocalTranslator

class UserEditDialog(QDialog):
    def __init__(self, parent=None, data=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_user_edit'))
```

```
        self.data = data or {}
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_login = QLineEdit(self.data.get("clogin", ""))
        self.edit_pass = QLineEdit(self.data.get("cpassword_hash", ""))
        self.edit_pass.setEchoMode(QLineEdit.Password)
        self.chk_active = QCheckBox(self.translator.tr('label_active'))
        self.chk_active.setChecked(self.data.get("lactive", True))
        self.edit_person = QLineEdit(str(self.data.get("person_instanceid", "")))
        form.addRow(self.translator.tr('field_login'), self.edit_login)
        form.addRow(self.translator.tr('field_password_hash'), self.edit_pass)
        form.addRow(self.translator.tr('field_person_instance'), self.edit_person)
        form.addRow(self.chk_active)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "clogin": self.edit_login.text(),
            "cpassword_hash": self.edit_pass.text(),
            "lactive": self.chk_active.isChecked(),
            "person_instanceid": self.edit_person.text() or None
        }

class RoleEditDialog(QDialog):
    def __init__(self, parent=None, data=None, db=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_role_edit'))
        self.data = data or {}
        self.db = db
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.edit_name = QLineEdit(self.data.get("cname", ""))
        self.edit_alias = QLineEdit(self.data.get("calias", ""))
        # Ссылка на родительскую роль
        self.ref_parent = ReferenceEditor(
            table_name="auth.roles",
            display_field="cname",
            db=self.db
        )
        if self.data.get("parentid"):
            sql = "SELECT cname FROM auth.roles WHERE id = %s"
            res = self.db.execute_query(sql, (self.data["parentid"],))
            if res:
                self.ref_parent.set_value(self.data["parentid"], res[0][0])
        self.chk_interface = QCheckBox(self.translator.tr('label_interface_role'))
        self.chk_interface.setChecked(self.data.get("lsinterface", False))
        self.edit_path = QLineEdit(self.data.get("cinterface_path", ""))
        self.edit_comment = QLineEdit(self.data.get("mcomment", ""))
        form.addRow(self.translator.tr('field_name'), self.edit_name)
        form.addRow(self.translator.tr('field_alias'), self.edit_alias)
        form.addRow(self.translator.tr('field_parent_role'), self.ref_parent)
        form.addRow(self.chk_interface)
        form.addRow(self.translator.tr('field_interface_path'), self.edit_path)
        form.addRow(self.translator.tr('field_comment'), self.edit_comment)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "cname": self.edit_name.text(),
            "calias": self.edit_alias.text(),
            "parentid": self.ref_parent.get_value(),
            "lsinterface": self.chk_interface.isChecked(),
            "cinterface_path": self.edit_path.text(),
            "mcomment": self.edit_comment.text()
        }

class ContextEditDialog(QDialog):
    def __init__(self, parent=None, users=None, roles=None, bus=None):
        super().__init__(parent)
        self.translator = LocalTranslator()
        self.setWindowTitle(self.translator.tr('title_context_assign'))
        layout = QVBoxLayout(self)
        form = QFormLayout()
        self.combo_user = QComboBox()
        for uid, ulogin in users or []:
            self.combo_user.addItem(f"{ulogin} ({uid})", uid)
        self.combo_role = QComboBox()
        for rid, rname in roles or []:
            self.combo_role.addItem(f"{rname} ({rid})", rid)
        self.combo_bu = QComboBox()
        for bid, bname in bus or []:
            self.combo_bu.addItem(f"{bname} ({bid})", bid)
        form.addRow(self.translator.tr('field_user'), self.combo_user)
        form.addRow(self.translator.tr('field_role'), self.combo_role)
        form.addRow(self.translator.tr('field_balance_unit'), self.combo_bu)
        layout.addLayout(form)
        btns = QHBoxLayout()
        btn_ok = QPushButton(self.translator.tr('btn_ok'))
        btn_ok.clicked.connect(self.accept)
        btn_cancel = QPushButton(self.translator.tr('btn_cancel'))
        btn_cancel.clicked.connect(self.reject)
        btns.addStretch()
        btns.addWidget(btn_ok)
        btns.addWidget(btn_cancel)
        layout.addLayout(btns)
    def get_data(self):
        return {
            "userid": self.combo_user.currentData(),
            "roleid": self.combo_role.currentData(),
            "balanceunitid": self.combo_bu.currentData()
        }

class UsersRolesTab(BaseTab):
    def __init__(self, parent=None, db=None):
        super().__init__(parent, db)
        self.tabs = QTabWidget()
        self.layout.addWidget(self.tabs)
        # Вкладка "Пользователи"
        self.users_tab = self._create_users_tab()
        self.tabs.addTab(self.users_tab, self.translator.tr('tab_users'))
        # Вкладка "Роли"
        self.roles_tab = self._create_roles_tab()
        self.tabs.addTab(self.roles_tab, self.translator.tr('tab_roles'))
        # Вкладка "Контекст (назначения)"
        self.context_tab = self._create_context_tab()
        self.tabs.addTab(self.context_tab, self.translator.tr('tab_context'))
        self.retranslate_ui()
    def retranslate_ui(self):
        # Обновляем заголовки внутренних вкладок
        self.tabs.setTabText(0, self.translator.tr('tab_users'))
        self.tabs.setTabText(1, self.translator.tr('tab_roles'))
        self.tabs.setTabText(2, self.translator.tr('tab_context'))
```

```
    # Обновляем заголовки таблиц и деревьев
    self._update_users_table_headers()
    self._update_roles_tree_headers()
    self._update_context_table_headers()
    # Обновляем кнопки CRUD
    if hasattr(self, 'user_crud'):
        self.user_crud.retranslate_ui()
    if hasattr(self, 'role_crud'):
        self.role_crud.retranslate_ui()
    if hasattr(self, 'context_crud'):
        self.context_crud.retranslate_ui()
def _update_users_table_headers(self):
    if hasattr(self, 'user_table'):
        self.user_table.setHorizontalHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_login'),
            self.translator.tr('field_active'),
            self.translator.tr('field_person_instance')
        ])
def _update_roles_tree_headers(self):
    if hasattr(self, 'role_tree'):
        self.role_tree.setHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_name'),
            self.translator.tr('field_alias'),
            self.translator.tr('field_interface'),
            self.translator.tr('field_interface_path')
        ])
def _update_context_table_headers(self):
    if hasattr(self, 'context_table'):
        self.context_table.setHorizontalHeaderLabels([
            self.translator.tr('field_id'),
            self.translator.tr('field_user'),
            self.translator.tr('field_role'),
            self.translator.tr('field_balance_unit')
        ])
def _create_users_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)
    self.user_crud = CrudButtons(self)
    layout.addWidget(self.user_crud)
    self.user_table = QTableWidget()
    self.user_table.setColumnCount(4)
    self.user_table.setHorizontalHeaderLabels([
        self.translator.tr('field_id'),
        self.translator.tr('field_login'),
        self.translator.tr('field_active'),
        self.translator.tr('field_person_instance')
    ])
    self.user_table.horizontalHeader().setStretchLastSection(True)
    self.user_table.setSelectionBehavior(QTableWidget.SelectRows)
    self.user_table.setEditTriggers(QTableWidget.NoEditTriggers)
    self.user_table.itemDoubleClicked.connect(self.edit_user)
    layout.addWidget(self.user_table)
    self.user_crud.add_clicked.connect(self.add_user)
    self.user_crud.edit_clicked.connect(self.edit_user)
    self.user_crud.delete_clicked.connect(self.delete_user)
    self._load_users()
    return tab
def _create_roles_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)
    self.role_crud = CrudButtons(self)
    layout.addWidget(self.role_crud)
    self.role_tree = QTreeWidget()
    self.role_tree.setHeaderLabels([
        self.translator.tr('field_id'),
        self.translator.tr('field_name'),
        self.translator.tr('field_alias'),
        self.translator.tr('field_interface'),
        self.translator.tr('field_interface_path')
    ])
    self.role_tree.itemDoubleClicked.connect(self.edit_role)
    layout.addWidget(self.role_tree)
    self.role_crud.add_clicked.connect(self.add_role)
    self.role_crud.edit_clicked.connect(self.edit_role)
    self.role_crud.delete_clicked.connect(self.delete_role)
    self._load_roles()
    return tab
def _create_context_tab(self):
    tab = QWidget()
    layout = QVBoxLayout(tab)
    self.context_crud = CrudButtons(self, with_add=True, with_edit=False, with_delete=True)
    layout.addWidget(self.context_crud)
    self.context_table = QTableWidget()
    self.context_table.setColumnCount(4)
    self.context_table.setHorizontalHeaderLabels([
        self.translator.tr('field_id'),
        self.translator.tr('field_user'),
        self.translator.tr('field_role'),
        self.translator.tr('field_balance_unit')
    ])
    self.context_table.horizontalHeader().setStretchLastSection(True)
    self.context_table.setSelectionBehavior(QTableWidget.SelectRows)
    self.context_table.setEditTriggers(QTableWidget.NoEditTriggers)
    layout.addWidget(self.context_table)
    self.context_crud.add_clicked.connect(self.add_context)
    self.context_crud.delete_clicked.connect(self.delete_context)
    self._load_context()
    return tab
# ---------------------------------------------------------------------
# Загрузка данных
# ---------------------------------------------------------------------
def _load_users(self):
    sql = "SELECT id, clogin, lactive, person_instanceid FROM auth.users ORDER BY id"
    rows = self.db.execute_query(sql)
    self.user_table.setRowCount(0)
    for row in rows:
        pos = self.user_table.rowCount()
        self.user_table.insertRow(pos)
        for col, val in enumerate(row):
            item = QTableWidgetItem(str(val) if val is not None else "")
            if col == 0:
                item.setData(Qt.UserRole, val)
            self.user_table.setItem(pos, col, item)
def _load_roles(self):
    sql = "SELECT id, parentid, level, data FROM get_hierarchy_iterative(%s, %s, %s)"
    rows = self.db.execute_query(sql, ('auth.roles', 'id', 'parentid'))
    self.role_tree.clear()
    items = {}
    for row in rows:
        node_id, parent_id, level, data = row
        name = data.get('cname', '')
        alias = data.get('calias', '')
        is_interface = data.get('lisinterface', False)
        interface_path = data.get('cinterface_path', '')
        item = QTreeWidgetItem()
        item.setText(0, str(node_id))
        item.setText(1, name)
        item.setText(2, alias)
        item.setText(3, "Да" if is_interface else "Нет")
        item.setText(4, interface_path or '')
        item.setData(0, Qt.UserRole, node_id)
        items[node_id] = item
        if parent_id is None:
            self.role_tree.addTopLevelItem(item)
        else:
            if parent_id in items:
                items[parent_id].addChild(item)
            else:
                self.role_tree.addTopLevelItem(item)
    self.role_tree.expandAll()
def _load_context(self):
    sql = "SELECT urc.id, u.clogin, r.cname, bu.cname
            FROM auth.usersrolescontext urc
            JOIN auth.users u ON urc.userid = u.id
            JOIN auth.roles r ON urc.roleid = r.id
            JOIN auth.balanceunits bu ON urc.balanceunitid = bu.id
            ORDER BY urc.id"
    rows = self.db.execute_query(sql)
```

```
        self.context_table.setRowCount(0)
        for row in rows:
            pos = self.context_table.rowCount()
            self.context_table.insertRow(pos)
            for col, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                if col == 0:
                    item.setData(Qt.UserRole, val)
                self.context_table.setItem(pos, col, item)
    # ---------------------------------------------------------------------
    # CRUD для пользователей
    # ---------------------------------------------------------------------
    def add_user(self):
        dlg = UserEditDialog(self)
        if dlg.exec():
            data = dlg.get_data()
            sql = """INSERT INTO auth.users (clogin, cpassword_hash, lactive, person_instanceid)
                     VALUES (%s, %s, %s, %s)"""
            self.db.execute_query(sql, (data["clogin"], data["cpassword_hash"], data["lactive"], data["person_instanceid"]),
fetch=False)
            self._load_users()
    def edit_user(self):
        row = self.user_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_user'))
            return
        item = self.user_table.item(row, 0)
        user_id = item.data(Qt.UserRole)
        sql = "SELECT clogin, cpassword_hash, lactive, person_instanceid FROM auth.users WHERE id=%s"
        row_data = self.db.execute_query(sql, (user_id,))
        if not row_data:
            return
        data = {
            "clogin": row_data[0][0],
            "cpassword_hash": row_data[0][1],
            "lactive": row_data[0][2],
            "person_instanceid": row_data[0][3]
        }
        dlg = UserEditDialog(self, data)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = """UPDATE auth.users SET clogin=%s, cpassword_hash=%s, lactive=%s, person_instanceid=%s WHERE
id=%s"""
            self.db.execute_query(sql, (new_data["clogin"], new_data["cpassword_hash"], new_data["lactive"],
new_data["person_instanceid"], user_id), fetch=False)
            self._load_users()
    def delete_user(self):
        row = self.user_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_user'))
            return
        item = self.user_table.item(row, 0)
        user_id = item.data(Qt.UserRole)
        if QMessageBox.question(self, self.translator.tr('confirm_delete'), self.translator.tr('confirm_delete_user')) ==
QMessageBox.Yes:
            sql = "DELETE FROM auth.users WHERE id=%s"
            self.db.execute_query(sql, (user_id,), fetch=False)
            self._load_users()
    # ---------------------------------------------------------------------
    # CRUD для ролей
    # ---------------------------------------------------------------------
    def add_role(self):
        dlg = RoleEditDialog(self, db=self.db)
        if dlg.exec():
            data = dlg.get_data()
            sql = """INSERT INTO auth.roles (cname, calias, parentid, lisinterface, cinterface_path, mcomment)
                     VALUES (%s, %s, %s, %s, %s, %s)"""
            self.db.execute_query(sql, (data["cname"], data["calias"], data["parentid"], data["lisinterface"],
data["cinterface_path"], data["mcomment"]), fetch=False)
            self._load_roles()
    def edit_role(self):
        item = self.role_tree.currentItem()
        if not item:
            self.show_warning(self.translator.tr('warning_select_role'))
            return
        role_id = item.data(0, Qt.UserRole)
        sql = "SELECT cname, calias, parentid, lisinterface, cinterface_path, mcomment FROM auth.roles WHERE
id=%s"
        row_data = self.db.execute_query(sql, (role_id,))
        if not row_data:
            return
        data = {
            "cname": row_data[0][0],
            "calias": row_data[0][1],
            "parentid": row_data[0][2],
            "lisinterface": row_data[0][3],
            "cinterface_path": row_data[0][4],
            "mcomment": row_data[0][5]
        }
        dlg = RoleEditDialog(self, data, db=self.db)
        if dlg.exec():
            new_data = dlg.get_data()
            sql = """UPDATE auth.roles SET cname=%s, calias=%s, parentid=%s, lisinterface=%s, cinterface_path=%s,
mcomment=%s WHERE id=%s"""
            self.db.execute_query(sql, (new_data["cname"], new_data["calias"], new_data["parentid"],
new_data["lisinterface"], new_data["cinterface_path"], new_data["mcomment"], role_id), fetch=False)
            self._load_roles()
    def delete_role(self):
        item = self.role_tree.currentItem()
        if not item:
            self.show_warning(self.translator.tr('warning_select_role'))
            return
        role_id = item.data(0, Qt.UserRole)
        if QMessageBox.question(self, self.translator.tr('confirm_delete'), self.translator.tr('confirm_delete_role')) ==
QMessageBox.Yes:
            sql = "DELETE FROM auth.roles WHERE id=%s"
            self.db.execute_query(sql, (role_id,), fetch=False)
            self._load_roles()
    # ---------------------------------------------------------------------
    # CRUD для контекста
    # ---------------------------------------------------------------------
    def get_users_list(self):
        sql = "SELECT id, clogin FROM auth.users ORDER BY clogin"
        return self.db.execute_query(sql)
    def get_roles_list(self):
        sql = "SELECT id, cname FROM auth.roles ORDER BY cname"
        return self.db.execute_query(sql)
    def get_bus_list(self):
        sql = "SELECT id, cname FROM auth.balanceunits ORDER BY cname"
        return self.db.execute_query(sql)
    def add_context(self):
        users = self.get_users_list()
        roles = self.get_roles_list()
        bus = self.get_bus_list()
        if not users or not roles or not bus:
            self.show_warning(self.translator.tr('warning_context_required'))
            return
        dlg = ContextEditDialog(self, users, roles, bus)
        if dlg.exec():
            data = dlg.get_data()
            sql = """INSERT INTO auth.userrolecontext (userid, roleid, balanceunitid)
                     VALUES (%s, %s, %s)"""
            self.db.execute_query(sql, (data["userid"], data["roleid"], data["balanceunitid"]), fetch=False)
            self._load_context()
    def delete_context(self):
        row = self.context_table.currentRow()
        if row < 0:
            self.show_warning(self.translator.tr('warning_select_context'))
            return
        item = self.context_table.item(row, 0)
        ctx_id = item.data(Qt.UserRole)
        if QMessageBox.question(self, self.translator.tr('confirm_delete'), self.translator.tr('confirm_delete_context')) ==
QMessageBox.Yes:
            sql = "DELETE FROM auth.userrolecontext WHERE id=%s"
            self.db.execute_query(sql, (ctx_id,), fetch=False)
            self._load_context()
```

Файл: /home/sergey/Documents/configurate/tensor_crypto.py (относительно: tensor_crypt

```
# tensor_crypto.py
import random
import numpy as np
from hashlib import sha256
# ----------------------------------------------------------
# Арифметика в конечном поле GF(p)
# ----------------------------------------------------------
class GFp:
    """Элемент поля Fp (простое p)."""
    def __init__(self, value, p):
        self.p = p
        self.value = value % p
    def __add__(self, other):
        return GFp((self.value + other.value) % self.p, self.p)
    def __sub__(self, other):
        return GFp((self.value - other.value) % self.p, self.p)
    def __mul__(self, other):
        return GFp((self.value * other.value) % self.p, self.p)
    def __truediv__(self, other):
        inv = pow(other.value, self.p-2, self.p)
        return GFp((self.value * inv) % self.p, self.p)
    def __neg__(self):
        return GFp(-self.value) % self.p, self.p)
    def __eq__(self, other):
        return self.value == other.value and self.p == other.p
    def __int__(self):
        return self.value
    def __repr__(self):
        return str(self.value)
def random_element(p):
    return GFp(random.randrange(p), p)
# ----------------------------------------------------------
# Матрицы n x n над GF(p)
# ----------------------------------------------------------
class Matrix:
    def __init__(self, n, p, data=None):
        self.n = n
        self.p = p
        if data is not None:
            self.data = data
        else:
            self.data = [[GFp(0, p) for _ in range(n)] for _ in range(n)]
    def __getitem__(self, idx):
        return self.data[idx]
    def __setitem__(self, idx, val):
        self.data[idx] = val
    @staticmethod
    def identity(n, p):
        m = Matrix(n, p)
        for i in range(n):
            m[i][i] = GFp(1, p)
        return m
    @staticmethod
    def random_invertible(n, p):
        """Генерация случайной обратимой матрицы через LU-подобный метод."""
        # Создаем верхнетреугольную с единицами на диагонали и случайными наддиагональными
        # и нижнетреугольную со случайными поддиагональными и диагональю 1.
        # Перемножаем, получаем обратимую.
        L = Matrix.identity(n, p)
        U = Matrix.identity(n, p)
        for i in range(n):
            for j in range(i+1, n):
                L[j][i] = random_element(p)
                U[i][j] = random_element(p)
        # Перемножаем L * U
        M = Matrix(n, p)
        for i in range(n):
            for j in range(n):
                s = GFp(0, p)
                for k in range(n):
                    s = s + L[i][k] * U[k][j]
                M[i][j] = s
        return M
    def inverse(self):
        """Обратная матрица методом Гаусса-Жордана."""
        n = self.n
        # Расширенная матрица [A | I]
        a = [[self[i][j] for j in range(n)] for i in range(n)]
        inv = [[GFp(1, p) if i==j else GFp(0, p) for j in range(n)] for i in range(n)]
        for col in range(n):
            # Поиск ненулевого элемента в столбце col
            pivot_row = col
            while pivot_row < n and a[pivot_row][col] == GFp(0, self.p):
                pivot_row += 1
            if pivot_row == n:
                raise ValueError("Matrix is singular")
            if pivot_row != col:
                a[col], a[pivot_row] = a[pivot_row], a[col]
                inv[col], inv[pivot_row] = inv[pivot_row], inv[col]
            pivot = a[col][col]
            inv_pivot = GFp(1, self.p) / pivot
            for j in range(n):
                a[col][j] = a[col][j] * inv_pivot
                inv[col][j] = inv[col][j] * inv_pivot
            for r in range(n):
                if r != col:
                    factor = a[r][col]
                    if factor != GFp(0, self.p):
                        for j in range(n):
                            a[r][j] = a[r][j] - factor * a[col][j]
                            inv[r][j] = inv[r][j] - factor * inv[col][j]
        result = Matrix(n, self.p)
        for i in range(n):
            for j in range(n):
                result[i][j] = inv[i][j]
        return result
    def __mul__(self, other):
        """Умножение матриц."""
        n = self.n
        result = Matrix(n, self.p)
        for i in range(n):
            for k in range(n):
                aik = self[i][k]
                if aik != GFp(0, self.p):
                    for j in range(n):
                        result[i][j] = result[i][j] + aik * other[k][j]
        return result
    def to_bytes(self):
        """Сериализация матрицы в байты (для хэширования)."""
        data = b''
        for i in range(self.n):
            for j in range(self.n):
                data += int(self[i][j]).to_bytes(2, 'big')  # т.к. p < 2^16
        return data
# ----------------------------------------------------------
# 3-тензоры n x n x n над GF(p)
# ----------------------------------------------------------
class Tensor3:
    def __init__(self, n, p, data=None):
        self.n = n
        self.p = p
        if data is not None:
            self.data = data
        else:
            self.data = np.empty((n, n, n), dtype=object)
            for i in range(n):
                for j in range(n):
                    for k in range(n):
                        self.data[i,j,k] = GFp(0, p)
    @staticmethod
```

```
    def fixed_tensor(n, p):
        """
        Эталонный тензор T0 с ненулевыми элементами только при i=j и k=1.
        T0[i,j,1] = 1 для всех i.
        Индексы: 0..n-1.
        T = Tensor3(n, p)
        for i in range(n):
            T.data[i,i,1] = GFp(1, p)
        return T
    def apply_isometry(self, A, B, C):
        """Применяет (A, B, C) к тензору: возвращает новый тензор."""
        n = self.n
        p = self.p
        # Преобразование по первому измерению
        T1 = np.empty((n, n, n), dtype=object)
        for i in range(n):
            for j1 in range(n):
                coeff = A[i,i1]
                if coeff != GFp(0, p):
                    for j in range(n):
                        for k in range(n):
                            if i == 0 and j == 0 and k == 0:
                                T1[i,j,k] = coeff * self.data[i1,j,k]
                            else:
                                T1[i,j,k] = T1[i,j,k] + coeff * self.data[i1,j,k]
        # по второму измерению
        T2 = np.empty((n, n, n), dtype=object)
        for i in range(n):
            for j1 in range(n):
                coeff = B[j,j1]
                if coeff != GFp(0, p):
                    for j in range(n):
                        for k in range(n):
                            if i == 0 and j == 0 and k == 0:
                                T2[i,j,k] = coeff * T1[i,j1,k]
                            else:
                                T2[i,j,k] = T2[i,j,k] + coeff * T1[i,j1,k]
        # по третьему измерению
        T3 = np.empty((n, n, n), dtype=object)
        for k in range(n):
            for k1 in range(n):
                coeff = C[k,k1]
                if coeff != GFp(0, p):
                    for j in range(n):
                        for i in range(n):
                            if i == 0 and j == 0 and k == 0:
                                T3[i,j,k] = coeff * T2[i,j,k1]
                            else:
                                T3[i,j,k] = T3[i,j,k] + coeff * T2[i,j,k1]
        result = Tensor3(n, p)
        result.data = T3
        return result
    def __repr__(self):
        return f"Tensor3({self.n}, {self.p})"
    def to_bytes(self):
        """Сериализация тензора в байты."""
        data = b''
        for i in range(self.n):
            for j in range(self.n):
                for k in range(self.n):
                    data += int(self.data[i,j,k]).to_bytes(2, 'big')
        return data
# ---------------------------------------------------------------
# Параметры системы (должны быть одинаковы у всех участников)
# ---------------------------------------------------------------
p = 65521        # простое число (2^16 - 15)
N = 16           # размерность тензора (можно увеличить до 32 для безопасности)
T0 = Tensor3.fixed_tensor(N, p)
# ---------------------------------------------------------------
# Основные функции криптосистемы
# ---------------------------------------------------------------
def generate_keypair():
    """Генерирует пару (публичный тензор, секретные матрицы)."""
    B = Matrix.random_invertible(N, p)
    C = Matrix.random_invertible(N, p)
    I = Matrix.identity(N, p)
    T_pub = T0.apply_isometry(I, B, C)
    return T_pub, (B, C)
def message_to_matrix(message: bytes, n: int, p: int) -> Matrix:
    """
    Преобразует байтовое сообщение в матрицу n x n над GF(p).
    Сообщение дополняется хэшем для обеспечения целостности и произвольной длины.
    """
    # Вычисляем хэш сообщения (32 байта) и добавляем в начало
    h = sha256(message).digest()
    data = h + message
    # Дополняем нулями до n*n*2 байт (т.к. каждый элемент матрицы кодируется 2 байтами)
    total_cells = n * n
    needed = total_cells * 2
    if len(data) < needed:
        data += b'\x00' * (needed - len(data))
    else:
        data = data[:needed]
    # Разбиваем на пары байт и превращаем в числа
    M = Matrix(n, p)
    idx = 0
    for i in range(n):
        for j in range(n):
            val = data[idx] * 256 + data[idx+1]
            M[i,j] = GFp(val, p)
            idx += 2
    return M
def matrix_to_message(M: Matrix) -> bytes:
    """
    Извлекает сообщение из матрицы.
    Ожидается, что первые 32 байта – хэш, затем само сообщение, затем дополнение.
    """
    n = M.n
    # Собираем байты
    data = b''
    for i in range(n):
        for j in range(n):
            v = int(M[i,j])
            data += v.to_bytes(2, 'big')
    # Извлекаем хэш (первые 32 байта) и остальное
    h_received = data[:32]
    rest = data[32:]
    # Удаляем нулевое дополнение в конце
    rest = rest.rstrip(b'\x00')
    # Проверяем хэш
    h_calc = sha256(rest).digest()
    if h_received != h_calc:
        raise ValueError("Контрольная сумма не совпадает – сообщение повреждено")
    return rest
def encrypt(public_tensor: Tensor3, message: bytes) -> Tensor3:
    """Шифрует сообщение с использованием публичного тензора."""
    X = message_to_matrix(message, public_tensor.n, public_tensor.p)
    I = Matrix.identity(public_tensor.n, public_tensor.p)
    # Вычисляем Y = (X, I, I) · T_pub
    Y = public_tensor.apply_isometry(X, I, I)
    return Y
def decrypt(secret_matrices, cipher_tensor: Tensor3) -> bytes:
    """Дешифрует тензор, используя секретные матрицы (B, C)."""
    B, C = secret_matrices
    n = cipher_tensor.n
    p = cipher_tensor.p
    I = Matrix.identity(n, p)
    Binv = B.inverse()
    Cinv = C.inverse()
    # Z = (I, Binv, Cinv) · Y
    Z = cipher_tensor.apply_isometry(I, Binv, Cinv)
    # Извлекаем матрицу X из слоя k=1 (индекс 1)
    X = Matrix(n, p)
    for i in range(n):
        for j in range(n):
            X[i,j] = Z.data[i,j,1]
    return matrix_to_message(X)
```

Файл: /home/sergey/Documents/configurate/view_sqlite.py (относительно: view_sqlite.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import sys
import sqlite3
import re
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                             QHBoxLayout, QPushButton, QTableView, QFileDialog,
                             QMessageBox, QLabel, QLineEdit, QHeaderView)
from PyQt5.QtCore import Qt, QAbstractTableModel, QModelIndex
from PyQt5.QtGui import QFont

class SQLiteTableModel(QAbstractTableModel):
    """Модель для отображения данных из SQLite с поддержкой фильтрации."""
    def __init__(self, data, headers):
        super().__init__()
        self._data = data              # список записей (кортежи)
        self._headers = headers        # список имен колонок
        self._filtered_rows = list(range(len(data)))
        self._filters = {}             # {col: pattern}
    def rowCount(self, parent=QModelIndex()):
        return len(self._filtered_rows)
    def columnCount(self, parent=QModelIndex()):
        return len(self._headers)
    def data(self, index, role=Qt.DisplayRole):
        if not index.isValid():
            return None
        if role == Qt.DisplayRole:
            row = self._filtered_rows[index.row()]
            col = index.column()
            value = self._data[row][col]
            return str(value) if value is not None else ""
        return None
    def headerData(self, section, orientation, role=Qt.DisplayRole):
        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
            return self._headers[section]
        return None
    def setFilter(self, col, pattern):
        if pattern:
            self._filters[col] = pattern
        else:
            self._filters.pop(col, None)
        self.updateFilter()
    def clearFilters(self):
        self._filters.clear()
        self.updateFilter()
    def updateFilter(self):
        self.beginResetModel()
        self._filtered_rows = []
        for row_idx, row in enumerate(self._data):
            match = True
            for col, pattern in self._filters.items():
                value = str(row[col]) if row[col] is not None else ""
                regex = re.escape(pattern).replace(r'\?', '.').replace(r'\*', '.*') + '$'
                if not re.match(regex, value):
                    match = False
                    break
            if match:
                self._filtered_rows.append(row_idx)
        self.endResetModel()

class SQLiteViewer(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Просмотрщик SQLite")
        self.resize(900, 700)
        central = QWidget()
        self.setCentralWidget(central)
        main_layout = QVBoxLayout(central)
        # Верхняя панель
        top_layout = QHBoxLayout()
        self.btn_open = QPushButton("Открыть SQLite файл")
        self.btn_open.clicked.connect(self.open_file)
        top_layout.addWidget(self.btn_open)
        self.btn_apply = QPushButton("Применить фильтр")
        self.btn_apply.clicked.connect(self.apply_filters)
        top_layout.addWidget(self.btn_apply)
        self.btn_clear = QPushButton("Сбросить фильтры")
        self.btn_clear.clicked.connect(self.clear_filters)
        top_layout.addWidget(self.btn_clear)
        self.info_label = QLabel("Файл не открыт")
        top_layout.addStretch()
        top_layout.addWidget(self.info_label)
        main_layout.addLayout(top_layout)
        # Панель фильтров (горизонтальные QLineEdit)
        self.filter_layout = QHBoxLayout()
        main_layout.addLayout(self.filter_layout)
        # Таблица
        self.table_view = QTableView()
        font = self.table_view.font()
        font.setPointSize(12)  # Увеличенный шрифт
        self.table_view.setFont(font)
        self.table_view.horizontalHeader().setStretchLastSection(True)
        self.table_view.setSortingEnabled(False)
        main_layout.addWidget(self.table_view)
        self.model = None
        self.filter_edits = []
        self.conn = None
    def open_file(self):
        file_path, _ = QFileDialog.getOpenFileName(
            self, "Выберите SQLite файл", "", "SQLite files (*.sqlite *.db);;All files (*)"
        )
        if not file_path:
            return
        try:
            # Подключаемся к базе
            self.conn = sqlite3.connect(file_path)
            cursor = self.conn.cursor()
            # Получаем список таблиц
            cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
            tables = [row[0] for row in cursor.fetchall()]
            if not tables:
                QMessageBox.warning(self, "Ошибка", "В базе нет таблиц")
                return
            # Для простоты используем первую таблицу (kladr или любую)
            # Можно дать выбор, но пока берем первую
            table_name = tables[0]
            # Загружаем данные
            cursor.execute(f"SELECT rowid, * FROM \"{table_name}\"")
            data = cursor.fetchall()
            headers = [description[0] for description in cursor.description]
            # Создаём модель
            self.model = SQLiteTableModel(data, headers)
            self.table_view.setModel(self.model)
            # Создаём фильтры
            self._create_filter_inputs(headers)
            self.info_label.setText(f"Файл: {file_path.split('/')[-1]} | Таблица: {table_name} | Записей: {len(data)}")
        except Exception as e:
            QMessageBox.critical(self, "Ошибка", f"Не удалось прочитать файл:\n{str(e)}")
            if self.conn:
                self.conn.close()
                self.conn = None
```

```
    def _create_filter_inputs(self, headers):
        self.clear_filter_layout()
        self.filter_edits = []
        for col, header in enumerate(headers):
            line_edit = QLineEdit()
            line_edit.setPlaceholderText(header)
            line_edit.returnPressed.connect(self.apply_filters)
            self.filter_layout.addWidget(line_edit)
            self.filter_edits.append((col, line_edit))
    def _clear_filter_layout(self):
        for i in reversed(range(self.filter_layout.count())):
            widget = self.filter_layout.itemAt(i).widget()
            if widget:
                widget.deleteLater()
    def apply_filters(self):
        if self.model is None:
            return
        for col, line_edit in self.filter_edits:
            text = line_edit.text().strip()
            self.model.setFilter(col, text if text else None)
    def clear_filters(self):
        if self.model is None:
            return
        self.model.clearFilters()
        for _, line_edit in self.filter_edits:
            line_edit.clear()
    def closeEvent(self, event):
        if self.conn:
            self.conn.close()
        event.accept()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    viewer = SplitViewer()
    viewer.show()
    sys.exit(app.exec_())
```

Файл: /home/sergey/Documents/configurate/widgets/__init__.py (относительно: widgets/__

```
# widgets/__init__.py
from .reference_editor import ReferenceEditor
from .reference_selector import ReferenceSelector
__all__ = ['ReferenceEditor', 'ReferenceSelector']
```

Файл: /home/sergey/Documents/configurate/widgets/reference_editor.py (относительно: w

```
#reference_editor.py
from PyQt5.QtWidgets import QWidget, QHBoxLayout, QLineEdit, QToolButton, QMessageBox
from PyQt5.QtCore import pyqtSignal
from database import DatabaseService

class ReferenceEditor(QWidget):
    valueChanged = pyqtSignal(object)
    def __init__(self, parent=None, table_name=None, display_field='cname',
                 id_field='id', dialog=None, db=None):
        super().__init__(parent)
        self.table_name = table_name
        self.display_field = display_field
        self.id_field = id_field
        self.dialog = dialog
        self.db = db or DatabaseService()
        self.current_id = None
        layout = QHBoxLayout(self)
        layout.setContentsMargins(0, 0, 0, 0)
        self.line_edit = QLineEdit()
        self.line_edit.setReadOnly(True)
        layout.addWidget(self.line_edit)
        self.btn_select = QToolButton()
        self.btn_select.setText("...")
        self.btn_select.clicked.connect(self.select_value)
        layout.addWidget(self.btn_select)
        self.btn_clear = QToolButton()
        self.btn_clear.setText("✖")
        self.btn_clear.clicked.connect(self.clear_value)
        layout.addWidget(self.btn_clear)
    def set_value(self, item_id, display_text=None):
        self.current_id = item_id
        if display_text is not None:
            self.line_edit.setText(display_text)
        else:
            self.line_edit.setText(str(item_id) if item_id else "")
        self.valueChanged.emit(item_id)
    def get_value(self):
        return self.current_id
    def setReadOnly(self, ro):
        self.btn_select.setEnabled(not ro)
        self.btn_clear.setEnabled(not ro)
    def select_value(self):
        if not self.table_name:
            QMessageBox.warning(self, "Ошибка", "Не указана таблица-источник")
            return
        if self.dialog:
            dlg = self.dialog(self.table_name, self.display_field, self.id_field, db=self.db)
        else:
            from widgets.reference_selector import ReferenceSelector
            dlg = ReferenceSelector(self.table_name, self.display_field, self.id_field, db=self.db, parent=self)
        if dlg.exec():
            selected_id, selected_display = dlg.get_selected()
            if selected_id:
                self.set_value(selected_id, selected_display)
    def clear_value(self):
        self.set_value(None, "")
```

Файл: /home/sergey/Documents/configurate/widgets/reference_selector.py (относительно:

```
# widgets/reference_selector.py
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QTableWidget, QTableWidgetItem, QPushButton,
QHBoxLayout, QLineEdit, QHeaderView
from PyQt5.QtCore import Qt
from database import DatabaseService

class ReferenceSelector(QDialog):
    """
    Универсальный диалог выбора записи из указанной таблицы.
    """
    def __init__(self, table_name, display_field='cname', id_field='id',
                 parent=None, db=None):
        super().__init__(parent)
        self.setWindowTitle("Выбор значения")
        self.resize(600, 400)
        self.table_name = table_name
        self.display_field = display_field
        self.id_field = id_field
        self.db = db or DatabaseService()
        self.selected_id = None
        self.selected_display = None
        layout = QVBoxLayout(self)
```

```
        self.search_edit = QLineEdit()
        self.search_edit.setPlaceholderText("Поиск...")
        self.search_edit.textChanged.connect(self.load_data)
        layout.addWidget(self.search_edit)
        self.table = QTableWidget()
        self.table.setColumnCount(2)
        self.table.setHorizontalHeaderLabels(["ID", "Наименование"])
        self.table.horizontalHeader().setStretchLastSection(True)
        self.table.setSelectionBehavior(QTableWidget.SelectRows)
        self.table.setSelectionMode(QTableWidget.SingleSelection)
        self.table.doubleClicked.connect(self.accept_selection)
        layout.addWidget(self.table)
        btn_layout = QHBoxLayout()
        btn_ok = QPushButton("Выбрать")
        btn_ok.clicked.connect(self.accept_selection)
        btn_cancel = QPushButton("Отмена")
        btn_cancel.clicked.connect(self.reject)
        btn_layout.addStretch()
        btn_layout.addWidget(btn_ok)
        btn_layout.addWidget(btn_cancel)
        layout.addLayout(btn_layout)
        self.load_data()
    def load_data(self):
        filter_text = self.search_edit.text().strip()
        if filter_text:
            sql = f"SELECT {self.id_field}, {self.display_field} FROM {self.table_name} WHERE {self.display_field} ILIKE %s
ORDER BY {self.display_field} LIMIT 200"
            params = (f"%{filter_text}%",)
        else:
            sql = f"SELECT {self.id_field}, {self.display_field} FROM {self.table_name} ORDER BY {self.display_field} LIMIT
200"
            params = None
        rows = self.db.execute_query(sql, params)
        self.table.setRowCount(0)
        for row in rows:
            pos = self.table.rowCount()
            self.table.insertRow(pos)
            for col, val in enumerate(row):
                item = QTableWidgetItem(str(val))
                if col == 0:
                    item.setData(Qt.UserRole, val)
                self.table.setItem(pos, col, item)
    def accept_selection(self):
        current_row = self.table.currentRow()
        if current_row >= 0:
            item = self.table.item(current_row, 0)
            self.selected_id = item.data(Qt.UserRole)
            self.selected_display = self.table.item(current_row, 1).text()
            self.accept()
    def get_selected(self):
        return self.selected_id, self.selected_display
```