

Politechnika Poznańska
Wydział Informatyki i Telekomunikacji

AUTOMATYCZNE GENEROWANIE STRUKTURY BAZY DANYCH NA PODSTAWIE TAKSONOMII I ŁADOWANIE DANYCH Z RAPORTÓW

Autorzy:
ADAM KORBIK
KRYSTIAN BRYŃSKI
MARTA JÓŹWIAK

05.2025r.

Spis treści

1	Cel projektu	2
2	Użyte technologie	3
3	Generowanie struktury na podstawie taksonomii	4
3.1	Ekstrakcja ścieżek kluczowych plików	4
3.2	Ekstrakcja nazwy oraz wersji taksonomii	5
3.3	Ekstrakcja etykiet wraz z przypisanym tekstem	7
3.4	Ekstrakcja etykiet wraz z przyporządkowanymi punktami danych	8
3.5	Ekstrakcja etykiet wraz z przypisaniem do osi	9
3.6	Ekstrakcja etykiet wraz z nazwą metryki	10
3.7	Ekstrakcja typów danych	11
3.8	Ekstrakcja unikalnych nazw formularzy	12
3.9	Integracja wyekstrahowanych danych	13
3.10	Proces tworzenia finalnej struktury dla jednego arkusza	14
3.11	Proces automatyzacji	15
4	Struktura tabel i relacji w bazie danych	16
4.1	Schemat bazy danych	16
4.2	Opis struktury bazy danych	17
5	Ekstrakcja danych z raportu	18
5.1	Struktura wejściowa	18
5.2	Opis procesu ekstrakcji danych z raportu	19
5.2.1	Wyodrębnienie nazwy arkusza	19
5.2.2	Usunięcie kolumn pomocniczych	19
5.2.3	Ekstrakcja kodów pozycji	20
5.2.4	Ekstrakcja danych finansowych	20
5.2.5	Zapis danych z raportu do formatu JSON	21
6	Automatyczny proces ładowania danych do bazy MSSQL	22
6.1	Połączenie z MSSQL za pomocą SQLAlchemy	22
6.2	Utworzenie tabel w MSSQL	22
6.3	Załadunek struktury taksonomii	22
6.3.1	Logika załadunku struktury	23
6.4	Załadunek danych z raportów	27
6.4.1	Logika załadunku danych z raportów	27
7	Obsługa programu	29
7.1	Instalacja bibliotek	29
7.2	Tworzenie katalogów	29
7.3	Użycie programu	29
7.3.1	Tworzenie struktury bazodanowej na podstawie taksonomii	29
7.3.2	Ekstrakcja danych z raportu	30
7.3.3	Ładowanie struktury bazodanowej do bazy danych	31
7.3.4	Ładowanie danych z raportu do bazy danych	31
7.3.5	Wyjście z programu	31

1 Cel projektu

Celem projektu jest zaprojektowanie i implementacja kompletnego systemu do automatycznego przetwarzania danych finansowych opartych na zewnętrznej strukturze taksonomii. System umożliwia:

- ekstrakcję informacji strukturalnych z plików XML opisujących taksonomię,
- wygenerowanie relacyjnej struktury bazy danych odzwierciedlającej istotne zależności,
- ekstrakcję danych z raportów Excel zgodnie z definicją taksonomii,
- automatyczne ładowanie struktury oraz danych raportowych do bazy danych Microsoft SQL Server,
- integrację z systemem API, pozwalającym na dostęp do danych oraz ich dalsze przetwarzanie.

Projekt zakłada pełną automatyzację przetwarzania — od momentu dostarczenia plików wejściowych (model taksonomii wraz z raportem) program utworzy strukturę bazodanową, wyekstrahuje dane z raportu oraz zasili bazę tymi danymi.

2 Użyte technologie

- Biblioteki Python

- **xml.etree.ElementTree** – Używana do przetwarzania plików XML. Umożliwia skuteczne i wydajne parsowanie oraz nawigację po strukturze dokumentów XML.
- **json** – W projekcie używamy tego formatu jako standardowego formatu transportowego do przechowywania danych wyodrębnionych z raportów oraz struktury opartej na taksonomii. Biblioteka JSON jest niezbędna do zapisu i odczytu tych danych.
- **os** – Moduł wykorzystywany do operacji na systemie plików i zarządzania ścieżkami plików. W projekcie służy do uzyskiwania docelowych ścieżek, łączenia ich w sposób niezależny od systemu operacyjnego, a także do obsługi operacji takich jak tworzenie katalogów, sprawdzanie istnienia plików czy pobieranie nazw plików.
- **dataclasses** – Używamy tego modułu, aby kod był bardziej czytelny i przejrzysty. To ułatwia zarządzanie strukturą danych. Ponadto, dataclasses wspierają typowanie, co pomaga pisać bardziej bezbłędny kod.
- **pandas** – W naszym projekcie służy przede wszystkim do łatwego parsowania i przetwarzania plików raportów w formacie Excel (.xls, .xlsx). Dzięki **pandas** możliwe jest szybkie odczytanie danych z arkuszy kalkulacyjnych, ich filtrowanie oraz przygotowanie do dalszego przetwarzania i eksportu do plików transportowych.
- **pyodbc** - Służy jako sterownik ODBC umożliwiający nawiązanie połączenia z instancją serwera MSSQL. Biblioteka ta zapewnia niskopoziomowy interfejs do wykonywania zapytań SQL oraz komunikacji z bazą danych poprzez protokół ODBC.
- **SQLAlchemy** – Pełni funkcję warstwy abstrakcji nad silnikiem bazy danych. Umożliwia wykonywanie zapytań SQL, mapowanie danych do struktur Pythona z użyciem mechanizmu **bind parameters**, zarządzanie sesją oraz transakcjami, a także obsługę błędów aplikacyjnych.
- **Microsoft SQL Server** - Pełni w projekcie funkcję centralnej relacyjnej bazy danych, w której gromadzone są zarówno dane strukturalne wynikające z definicji taksonomii, jak i dane raportowe pochodzące z plików wejściowych. Zapewnia integralności danych dzięki zastosowaniu kluczy głównych i obcych, które łączą tabele między sobą.
- **FastAPI** - Pełni rolę interfejsu dostępowego (API), umożliwiającego komunikację między aplikacją a bazą danych Microsoft SQL Server.
- **Git** - Systemu kontroli wersji.

3 Generowanie struktury na podstawie taksonomii

3.1 Ekstrakcja ścieżek kluczowych plików

Opracowaliśmy metodę automatycznego identyfikowania i zapisywania ścieżek do kluczowych plików taksonomicznych, aby uogólnić i usprawnić nasze rozwiązanie. Naszym celem jest wyodrębnienie plików o określonych prefiksach: `rend`, `lab-pl` oraz `lab-codes` dla każdego arkusza, ponieważ zawierają one istotne informacje dotyczące struktury danych arkuszy. W ramach taksonomii każdy arkusz posiada unikalny folder, np. `n.ro.bk.00`, w którym znajdują się odpowiednie pliki źródłowe. Aby zwiększyć przejrzystość i efektywność pracy, zdecydowaliśmy się na automatyczne wyodrębnienie nazwy arkusza (czyli nazwy folderu) oraz trzech kluczowych ścieżek do plików zawierających: `rend`, `lab-pl` i `lab-codes` dla każdego arkusza. Dzięki temu podejściu możemy w sposób systematyczny i zautomatyzowany analizować strukturę taksonomii, co znacząco ułatwia dalsze przetwarzanie danych. W kolejnych podpunktach szczegółowo opiszemy zawartość plików `rend`, `lab-pl` oraz `lab-codes`, a także dokładnie określimy, jakie informacje są z nich wyciągane i w jaki sposób są wykorzystywane.

Funkcja o nazwie `collect_target_paths_and_sheet_name` została zaprojektowana tak, aby automatycznie przeszukiwać katalog z taksonomią, identyfikować odpowiednie pliki oraz zwracać listę zawierającą krotki z kluczowymi informacjami. Każda zwrócona krotka składa się z dwóch elementów: Listy ścieżek do trzech plików (`rend`, `lab-pl`, `lab-codes`), które zawierają istotne informacje o strukturze danych i nazwy arkusza, odpowiadającej nazwie folderu, w którym pliki się znajdują.

3.2 Ekstrakcja nazwy oraz wersji taksonomii

Aby zachować pełną strukturę danych w bazie, niezbędne jest uwzględnienie informacji o nazwie oraz wersji taksonomii. Jest to kluczowa informacja, ponieważ przed dodaniem nowej struktury konieczne jest sprawdzenie, czy dana taksonomia już istnieje w bazie. Dzięki temu unikamy duplikacji danych i zapewniamy spójność przechowywanych informacji. Weryfikacja wersji taksonomii pozwala na kontrolę aktualizacji oraz integrację nowych danych bez ryzyka nadpisania istniejących wpisów.

Te informacje możemy znaleźć w katalogu META-IFNO w pliku `taxonomyPackage.xml` z którego można pobrać nazwę oraz wersję taksonomii.

```
<?xml version="1.0" encoding="UTF-8"?>
<taxonomyPackage xml:lang="en" xmlns="http://xbrl.org/2016/taxonomy-package"

  <identifier>http://www.knf.gov.pl/pl/xbrl</identifier>
  <name>BION</name>
  <description>Sprawozdanie BION</description>
  <version>1.0</version>
  <publisher>Komisja Nadzoru Finansowego</publisher>
  <publisherURL>http://www.knf.gov.pl/</publisherURL>
  <publicationDate>2024-10-21</publicationDate>

</taxonomyPackage>
```

Rysunek 1: Przykładowa zawartość pliku `taxonomyPackage.xml`

W celu zapewnienia poprawnej identyfikacji struktury danych, został opracowany skrypt, który automatycznie wyodrębnia nazwę oraz wersję taksonomii. Funkcja realizująca to zadanie nosi nazwę `collect_name_and_version_taxonomy` i została zaprojektowana tak, aby skutecznie przeszukiwać plik w celu odnalezienia tych kluczowych informacji.

Po zakończeniu analizy funkcja zapisuje pobrane dane jako plik transportowy JSON w formacie obiektu:

```
{
  "name": "name_tax",
  "version": "version_tax"
}
```

Dzięki temu podejściu możliwe jest przechowywanie informacji o nazwie i wersji taksonomii w czytelny, ustandaryzowany sposób. Jeśli w tabeli naszej bazy danych

brakuje wpisu dotyczącego konkretnej nazwy i wersji taksonomii, dane z pliku transportowego JSON zostaną załadowane, obejmując zarówno nazwę i wersję taksonomii, jak i całą jej strukturę – pod warunkiem, że nie istnieje już wpis dla tej taksonomii. Szczegóły dotyczące procesu ładowania oraz integracji struktury danych zostaną opisane w kolejnych punktach.

3.3 Ekstrakcja etykiet wraz z przypisanym tekstem

W plikach XML zawierających tzw. „lab-pl” znajdują się etykiety wraz z przypisanymi do nich tekstami. Na podstawie tych plików jesteśmy w stanie określić, jaki tekst odpowiada poszczególnym etykiетom. Operacja ta jest niezbędna do prawidłowego odwzorowania struktury danych.

```
<link:loc xlink:type="locator" xlink:href="n.zb.bk.02.01-rend.xml#uknf_c19" xlink:label="loc_uknf_c19" />
<label:label xlink:type="resource" xlink:label="label_uknf_c19" xml:lang="pl" xlink:role="http://www.xbrl.org/2008/role/label">inne</label:label>
<gen:arc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/element-label" xlink:from="loc_uknf_c19" xlink:to="label_uknf_c19" />
```

Rysunek 2: Fragment pliku zawierającego „lab-pl”

Na Rysunku 2 przedstawiono fragment pliku XML. Jak widać, z tego pliku należy wyodrębnić etykietę wraz z przypisanym jej tekstem. Docelowo interesuje nas wydobycie pary, np. `uknf_c19: Inne`.

W tym celu została zaimplementowana funkcja `extract_lab_pl_labels_and_values`, która zwraca listę wszystkich znalezionych par etykieta–tekst, zgodnie z opisanym powyżej formatem.

3.4 Ekstrakcja etykiet wraz z przyporządkowanymi punktami danych

W plikach XML zawierających tzw. „lab-codes” znajdują się etykiety wraz z przypisanymi do nich punktami danych. Korzystając z tych plików, dla każdego arkusza jesteśmy w stanie wyodrębnić etykiety wraz z odpowiadającym jej punktami danych. Proces ten jest niezbędny, aby sklasyfikować etykietę wraz z przypisanym punktem danych i utworzyć poprawną strukturę danych.

```
<link:loc xlink:type="locator" xlink:href="n.pif.bk.00-rend.xml#uknf_c5" xlink:label="loc_uknf_c5" />
<label:label xlink:type="resource" xlink:label="label_uknf_c5" xml:lang="en" xlink:role="http://www.eurofiling.info/xbrl/role/rc-code">0040</label:label>
<gen:arc xlink:type="arc" xlink:arcrole="http://xbrl.org/arcrole/2008/element-label" xlink:from="loc_uknf_c5" xlink:to="label_uknf_c5" />
```

Rysunek 3: Fragment pliku zawierającego „lab-codes”

Na Rysunku 3 przedstawiono fragment pliku XML, który można sparsować w celu ekstrakcji etykiet wraz z przypisanymi do nich wartościami, czyli punktami danych, na przykład: `uknf_c5` odpowiada wartości `0040`. Taka ekstrakcja jest kluczowa do dalszego przetwarzania danych. W naszym skrypcie została zaimplementowana funkcja `extract_lab_codes_labels_and_values`, która przyjmuje na wejściu sparsowany plik XML. Funkcja ta służy do wyodrębnienia powiązań między etykietami a odpowiadającymi im punktami danych. Dzięki temu możliwe jest dokładne określenie, jaki konkretny punkt danych jest przypisany do której etykiety. W praktyce funkcja przeszukuje elementy XML o nazwie `label`, pobiera ich atrybuty identyfikujące oraz tekst zawierający wartość, a następnie zwraca listę par (etykieta, punkt danych). Do dalszego przetwarzania potrzebujemy identyfikatorów etykiet w formie np. `uknf_c50`, a nie z prefiksem `label_uknf_c50`. Dlatego w funkcji zastosowano mechanizm filtrowania, który usuwa prefiks `label_` z nazwy etykiety. Takie podejście zapewnia spójność identyfikatorów etykiet w całym procesie przetwarzania oraz umożliwia poprawne i jednolite mapowanie etykiet na odpowiadające im punkty danych. Dzięki temu unikamy problemów wynikających z różnic w nazwach między różnymi plikami XML i upraszczamy dalszą analizę danych.

3.5 Ekstrakcja etykiet wraz z przypisaniem do osi

Ważnym etapem przetwarzania danych jest przypisanie poszczególnych etykiet do odpowiednich osi x lub y . W tym celu wykorzystywany jest plik XML zawierający w nazwie człon „rend”, z którego możliwe jest odczytanie kolejności występowania etykiet oraz znaczników osi.

Dla arkusza dwuwymiarowego przykładowa sekwencja może wyglądać następująco:

`[uknf_c1, uknf_c2, uknf_c3, x, uknf_c4, y]`

Na podstawie takiej listy etykiety pojawiające się przed pierwszym wystąpieniem osi x są przypisywane do osi x , natomiast wszystkie etykiety po tym znaczniku klasyfikowane są jako należące do osi y .

W przypadku arkusza jednowymiarowego sekwencja może przyjąć formę:

`[uknf_c1, uknf_c2, uknf_c3, x]`

Wówczas wszystkie etykiety poprzedzające wystąpienie znacznika osi x przypisuje się do tej właśnie osi.

Taka klasyfikacja umożliwia prawidłowe odwzorowanie struktury arkuszy w dalszych etapach przetwarzania danych.

Funkcja `extract_rend_ordered_labels_and_axes` odpowiada za odczyt kolejności występowania etykiet oraz osi na podstawie struktury pliku XML zawierającego człon „rend”. Jej zadaniem jest wygenerowanie listy identyfikatorów, w której zachowana jest kolejność pojawiania się zarówno etykiet, jak i znaczników osi.

Funkcja działa w następujący sposób:

- przeszukuje elementy XML w poszukiwaniu tagów `ruleNode`, z których odczytuje atrybut `id` — identyfikator etykiety,
- identyfikuje elementy `tableBreakdownArc`, z których pobierany jest atrybut `axis` — reprezentujący oś (np. x , y),
- na końcu stosowane jest filtrowanie, które usuwa elementy pomocnicze, tj. identyfikatory rozpoczynające się od prefiksu `uknf_a`, ponieważ nie są one istotne dla przypisania etykiet do osi.

Tak przygotowana lista pozwala na jednoznaczne przypisanie każdej etykiety do odpowiedniej osi, zgodnie z jej kolejnością występowania w strukturze dokumentu XML. To podejście jest kluczowe dla dalszego przetwarzania danych i poprawnego odwzorowania struktury arkusza.

3.6 Ekstrakcja etykiet wraz z nazwą metryki

W pliku XML zawierającym człon „rend” można znaleźć informacje o powiązaniach pomiędzy etykietami a odpowiadającymi im metrykami. Informacja ta jest kluczowa, ponieważ umożliwia zachowanie spójności typów danych w strukturze danych — każda etykieta jest powiązana z metryką, która definiuje jej typ danych.

```
<table:ruleNode xlink:type="resource" xlink:label="uknf_c4" id="uknf_c4">
  <formula:concept>
    <formula:qname>uknf_met:si2165</formula:qname>
  </formula:concept>
</table:ruleNode>
```

Rysunek 4: Fragment pliku zawierającego „rend”

Jak przedstawiono na Rysunku 4, możliwe jest odczytanie z pliku nazwy metryki (atrybut `qname`), która została przypisana do konkretnej etykiety. W zaprezentowanym przykładzie etykieta `uknf_c4` jest powiązana z metryką `uknf_met:si2165`. Tego rodzaju odwzorowanie stanowi istotny krok w procesie budowania kompletnej struktury danych, ponieważ pozwala na późniejsze przypisanie odpowiednich typów danych na podstawie informacji z pliku `met.xsd`.

W naszym skrypcie funkcja `extract_rend_labels_and_qnames` odpowiada za realizację tej operacji. Przetwarza odpowiedni plik XML i zwraca listę krotek w formacie: (etykieta, nazwa metryki). Dzięki temu możliwe jest jednoznaczne powiązanie każdej etykiety z odpowiadającą jej metryką, co stanowi podstawę do przypisania właściwego typu danych w dalszych etapach przetwarzania.

3.7 Ekstrakcja typów danych

W katalogu `dict/met` znajduje się plik `met.xsd`, który odpowiada za klasyfikację nazw metryk według typów danych. Plik ten jest niezbędny do zapewnienia poprawnego typowania danych w dalszym etapie przetwarzania. Na podstawie tego pliku ekstraktowane są wszystkie dostępne metryki wraz z przypisanymi do nich typami danych. Każda etykieta w strukturze danych jest powiązana z konkretną metryką, dlatego wyodrębnienie tych informacji stanowi kluczowy element budowy spójnej struktury danych. Dzięki pozyskanym z pliku `met.xsd` informacjom możliwe jest przypisanie odpowiedniego typu danych do każdej etykiety, poprzez odniesienie się do metryki, z którą dana etykieta jest powiązana. Pozwala to zachować integralność typów w całym zbiorze danych oraz umożliwia poprawną interpretację wartości występujących w poszczególnych polach.

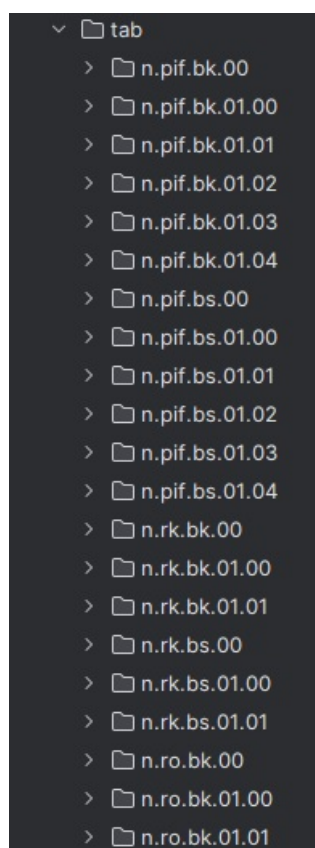
```
<xs:element name="di2120" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2120" xbrli:periodType="instant" nillable="true"
<xs:element name="di2121" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2121" xbrli:periodType="instant" nillable="true"
<xs:element name="di2122" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2122" xbrli:periodType="instant" nillable="true"
<xs:element name="di2160" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2160" xbrli:periodType="instant" nillable="true"
<xs:element name="di2161" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2161" xbrli:periodType="instant" nillable="true"
<xs:element name="di2273" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2273" xbrli:periodType="instant" nillable="true"
<xs:element name="di2274" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di2274" xbrli:periodType="instant" nillable="true"
<xs:element name="di3002" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di3002" xbrli:periodType="instant" nillable="true"
<xs:element name="di3023" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di3023" xbrli:periodType="instant" nillable="true"
<xs:element name="di3024" type="xbrli:dateItemType" substitutionGroup="xbrli:item" id="uknf_di3024" xbrli:periodType="instant" nillable="true"
<xs:element name="ei1" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei1" xbrli:periodType="instant" nillable="true" model
<xs:element name="ei2118" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2118" xbrli:periodType="instant" nillable="true"
<xs:element name="ei2162" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2162" xbrli:periodType="instant" nillable="true"
<xs:element name="ei2163" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2163" xbrli:periodType="instant" nillable="true"
<xs:element name="ei2195" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2195" xbrli:periodType="instant" nillable="true"
<xs:element name="ei2196" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2196" xbrli:periodType="instant" nillable="true"
<xs:element name="ei2272" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2272" xbrli:periodType="instant" nillable="true"
<xs:element name="ei2282" type="xbrli:QNameItemType" substitutionGroup="xbrli:item" id="uknf_ei2282" xbrli:periodType="instant" nillable="true"
```

Rysunek 5: Fragment pliku `met.xsd`

W celu pozyskania metryk wraz z odpowiadającymi im typami danych została zaimplementowana funkcja `extract_types_and_names`. Funkcja ta przyjmuje jako argument sparsowany plik `met.xsd`, a następnie przeszukuje wszystkie elementy typu `element`. Dla każdego z nich wyodrębnia atrybuty `name` oraz `type`, które reprezentują odpowiednio nazwę metryki oraz przypisany jej typ danych. Wynikiem działania funkcji jest lista par (`nazwa`, `typ`), która stanowi podstawę do dalszego przypisywania typów danych poszczególnym etykiетom w strukturze danych. Takie podejście pozwala zachować spójność i poprawność typowania w całym procesie przetwarzania raportów.

3.8 Ekstrakcja unikalnych nazw formularzy

W naszej strukturze danych niezbędne jest pozyskanie unikalnych nazw formularzy występujących w taksonomii. W tym celu wykorzystujemy zawartość katalogu o nazwie `tab`, który zawiera podkatalogi odpowiadające poszczególnym arkuszom formularzy, np. `n.pif.bk.00`, `n.pif.bs.01.00` itp. Każdy z tych folderów zawiera pliki opisujące strukturę pojedynczego arkusza danego formularza. Aby uzyskać zestaw unikalnych nazw formularzy, analizujemy wszystkie nazwy folderów w katalogu `tab` i redukujemy je do postaci składającej się wyłącznie z pierwszych trzech segmentów oddzielonych kropkami. Przykładowo, z nazwy `n.pif.bk.00` uzyskujemy `n.pif.bk`, a z `n.pif.bs.01.00` — `n.pif.bs`. W ten sposób otrzymujemy zestaw jednoznacznych identyfikatorów formularzy wykorzystywanych w taksonomii, które mogą być dalej wykorzystywane w procesach analizy i odwzorowywania danych.



Rysunek 6: Przykład zawartości katalogu `tab`

W tym celu została zaimplementowana funkcja `collect_unique_form_names`, która przeszukuje katalog `tab`, analizuje nazwy podkatalogów i zwraca listę unikalnych nazw formularzy. Funkcja ta usuwa z nazw folderów wszelkie segmenty występujące po trzecim znaku kropki, co pozwala na uzyskanie jednoznacznych identyfikatorów formularzy. Dzięki temu mechanizmowi możliwe jest efektywne zarządzanie oraz dalsze przetwarzanie danych związanych z poszczególnymi formularzami, zapewniając spójność i unikalność ich identyfikatorów w całej strukturze taksonomii.

3.9 Integracja wyekstrahowanych danych

W celu wstępnego połączenia danych funkcja `combine_data_from_files`, której zadaniem jest określenie, czy dany formularz jest jednoosiowy czy dwuosiowy. Na podstawie tej klasyfikacji proces przetwarzania danych odbywa się poprzez wywołanie odpowiedniej funkcji: `combine_one_dimensional_data` dla formularzy jednoosiowych oraz `combine_two_dimensional_data` dla formularzy dwuosiowych.

Funkcja `combine_one_dimensional_data` generuje strukturę w formie słownika, w którym kluczami są etykiety odnoszące się do wierszy lub kolumn (w zależności od arkusza), a wartościami tekst powiązany z etykietą, przypisane punkty danych oraz informacja o osi, w której znajduje się dany klucz. Informacja o osi będzie wykorzystywana w dalszych etapach analizy do określenia orientacji arkusza (pionowej lub poziomej).

Funkcja `combine_two_dimensional_data` tworzy strukturę słownikową, w której kluczami są etykiety wierszy, a wartościami tekst przypisany do wiersza, zintegrowane punkty danych oraz uporządkowane teksty kolumnowe odpowiadające kolejności punktów danych. Proces integracji punktów danych obejmuje krzyżowanie wartości, polegające na połączeniu wszystkich punktów przypisanych do danej etykiety, np. 0010X0010, 0010X0020. Dzięki temu możliwa jest pełna synchronizacja kolumn i wierszy, co usprawnia dalszą analizę struktury arkusza.

Do tej struktury danych dodawany jest również typ danych oraz nazwa metryki. Funkcja `match_labels_with_data_types` tworzy mapę etykiet, przypisując im odpowiednie typy danych oraz nazwy metryk. Dzięki temu możliwe jest późniejsze wykorzystanie funkcji `match_datatypes_and_qnames`, która przyjmuje wcześniej zdefiniowaną strukturę słownika. W jej działaniu następuje dopasowanie typu danych oraz nazwy metryki do każdego klucza w słowniku na podstawie przypisanych etykiet.

Możliwa jest sytuacja, w której funkcja nie przypisze żadnej nazwy metryki ani typu danych. Może to wynikać z braku odpowiedniego wpisu dla danej etykiety w pliku `met.xsd`, który zawiera wszystkie dostępne typy danych wraz z nazwami metryk.

W przypadku braku dopasowania obowiązuje domyślna konfiguracja, w której: `typ danych` przyjmuje wartość `"xbrli:stringItemType"`, `nazwa metryki` zostaje ustawione na `"None"`.

Do finalnej struktury będzie jeszcze dodana nazwa arkusza i nazwa formularza

3.10 Proces tworzenia finalnej struktury dla jednego arkusza

Jako plik transportowy wybrano format JSON, który zapewnia spójność i czytelność danych oraz umożliwia ich bezproblemowe załadowanie do bazy danych. W takim formacie finalna struktura będzie przechowywana, co pozwoli na jej łatwą integrację i dalsze przetwarzanie.

Funkcja `transform_data` przyjmuje na wejściu słownik zawierający większość informacji opisanych w punkcie 3.9, a następnie generuje finalną strukturę JSON, dostosowaną do wymagań systemu.

Finalny obiekt JSON przyjmuje następującą postać:

```
{
  "form_name": "example_form",
  "data": {
    "label_1": {
      "value_row": "Example row value",
      "data_points": ["0010X0010", "0010X0020"],
      "value_columns": ["Example column 1", "Example column 2"],
      "datatype": "xbrli:stringItemType",
      "qname": "None",
      "sheet_name": "Sheet1"
    },
    "label_2": {
      "value_row": "Another row value",
      "data_points": ["0020X0030"],
      "value_columns": "Single column value",
      "datatype": "xbrli:monetaryItemType",
      "qname": "SomeQName",
      "sheet_name": "Sheet1"
    }
  }
}
```

W ten sposób dla pojedynczego arkusza został utworzony plik transportowy, w którym głównym kluczem jest nazwa formularza. Wszystkie związane z nim dane są przechowywane wewnątrz tej struktury, co pozwala na jednoznaczną identyfikację formularza oraz jego zawartości. Taki sposób organizacji pliku zapewnia spójność i czytelność.

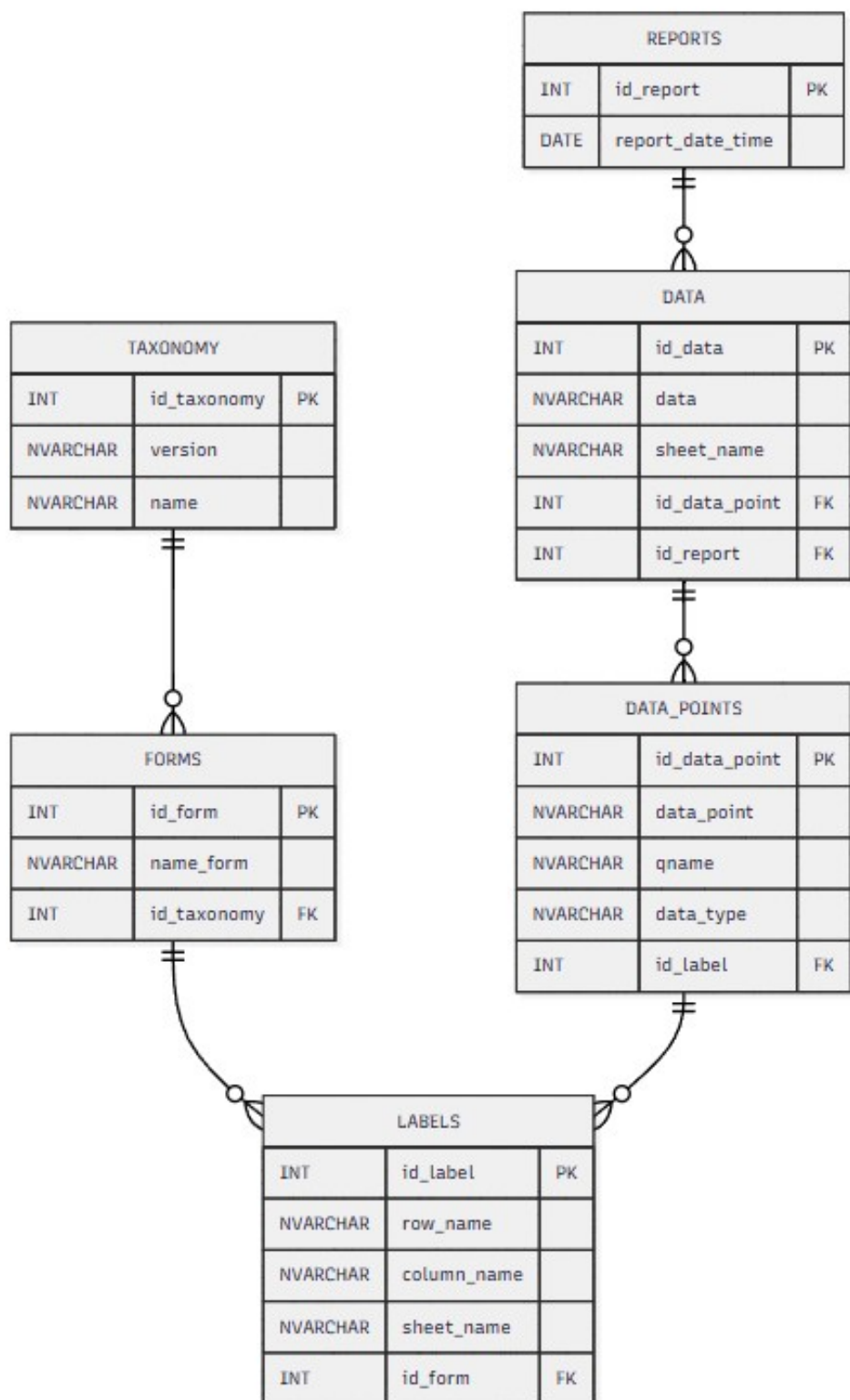
3.11 Proces automatyzacji

W celu zautomatyzowania procesu została opracowana funkcja `create_structure`, której głównym zadaniem jest iteracja przez wszystkie unikalne nazwy formularzy, identyfikacja powiązanych arkuszy wraz z plikami docelowymi (proces ich ekstrakcji i opis znajduje się w punkcie 3.1). Następnie każda nazwa formularza oraz arkusza, wraz z odpowiadającymi im plikami docelowymi, jest przekazywana do funkcji `build_json_for_sheet`. Funkcja ta odpowiada za integrację wszystkich modułów, utworzenie kompletnej struktury w formacie JSON dla danego arkusza oraz jej zapis do dedykowanego folderu.

Po zakończeniu głównej iteracji w funkcji `create_structure` generowana jest pełna struktura obejmująca wszystkie arkusze, zapisana w formatach JSON. Liczba utworzonych plików transportowych jest równa liczbie arkuszy występujących w obrębie formularzy, co zapewnia przejrzystość danych oraz ich jednoznaczną identyfikację. Struktura ta pozwala na łatwe przechowywanie i dalsze przetwarzanie informacji w kontekście analizy taksonomicznej.

4 Struktura tabel i relacji w bazie danych

4.1 Schemat bazy danych



Rysunek 7: Struktura bazy danych

4.2 Opis struktury bazy danych

Diagram obrazuje docelową strukturę relacyjnej bazy danych, złożoną z sześciu kluczowych tabel: **Reports**, **Data**, **Data_points**, **Labels**, **Taxonomy** oraz **Forms**. Każda tabela posiada własny klucz główny (PK), zaś relacje pomiędzy nimi realizowane są poprzez odpowiadające im klucze obce (FK). W praktyce:

Tabela **Taxonomy** definiuje kategorie i ich wersje, a tabela **Forms** odwołuje się do niej za pomocą klucza obcego `id_taxonomy`, wskazując, do której taksonomii należy dany formularz.

Formularze w tabeli **Forms** składają się z wielu etykiet przechowywanych w tabeli **Labels**, gdzie kolumna `id_form` stanowi klucz obcy łączący etykiety z odpowiednim formularzem.

Każda etykieta w tabeli **Labels** zawiera zestaw punktów danych zdefiniowanych w tabeli **Data_points**, które są powiązane poprzez kolumnę `id_label` (FK).

Tabela **Data** gromadzi dane z raportów, przy czym każdy rekord wskazuje, do którego punktu danych (`id_data_point`) oraz do którego raportu (`id_report`) należy (obie kolumny są kluczami obcymi).

Tabela **Reports** pełni rolę nadrzędnej kolekcji zestawów danych z raportów, agregując wszystkie rekordy z tabeli **Data**.

5 Ekstrakcja danych z raportu

Celem procesu ekstrakcji jest automatyczne wydobywanie danych z tabel zawartych w raportach finansowych. W trakcie działania skryptu pozyskiwane są nazwy arkuszy, kody pozycji (tzw. datapoint'y), a następnie z tych pozycji pobierane są odpowiadające im dane. Proces obsługuje trzy różne układy tabel i zapisuje wyodrębnione dane w formacie JSON, aby umożliwić dalszy ładunek do bazy danych.

5.1 Struktura wejściowa

Skrypt przyjmuje jako wejście plik Excel (*.xlsx, *.xls) zawierający raport. Plik ma ustandaryzowaną strukturę wynikającą z wytycznych Taksonomii BION:

- Każdy raport zawiera wiele arkuszy
- Każda wartość finansowa reprezentowana jest przez datapoint/y
- Dane tabelaryczne mogą być zapisane w układach: horyzontalny, wertykalny, mieszany.

Poniżej przedstawione są przykładowe obrazy trzech układów danych tabelarycznych.

BION UKNF		
N.PiF.BK.01.00 Nagłówek sekcji - Informacje pomocnicze do oceny poziomu ryzyka płynności i finansowania		
Sporządził(a):		
Imię	0010	
Nazwisko	0020	
telefon	0030	
e-mail	0040	
Zatwierdził(a):		
Imię	0050	
Nazwisko	0060	
telefon	0070	
e-mail	0080	
Komórka ds. zgodności:		
Imię	0090	
Nazwisko	0100	
telefon	0110	
e-mail	0120	
Audyt wewnętrzny:		
Imię	0130	
Nazwisko	0140	
telefon	0150	
e-mail	0160	

Rysunek 8: Układ wertykalny

BION UKNF N.PiF.BK.01.04 Zestawienie posiadanych przez bank długoterminowych źródeł finansowania									
Zestawienie posiadanych przez bank długoterminowych źródeł finansowania (z pierwotnym terminem wymagalności dłuższym niż 1 rok) wg stanu na 31.12.RRRR (w zestawieniu proszę nie uwzględniać depozytów)									
Lp.	Podmiot udzielający finansowania (proszę podać pełną nazwę podmiotu)	Rodzaj finansowania (np. kredyt/pożyczka/pożyczka podporządkowana/obligacje)	Kwota finansowania w walucie oryginalnej w dniu udzielenia finansowania (w PLN)	Waluta udzielonego finansowania	Kwota pozostająca do spłaty wg stanu na koniec okresu sprawozdawczego	Data udzielenia finansowania	Pierwotny termin spłaty	Aktualny termin spłaty	Informacje dodatkowe / komentarze
0010	0020	0030	0040	0050	0060	0070	0080	0090	0100
1.									
2.									
3.									
4.									
5.									
6.									
7.									
8.									
9.									
10.									
11.									
12.									
13.									
14.									
15.									
16.									
17.									
18.									
19.									
20.									
21.									
22.									

Rysunek 9: Układ horyzontalny

BION UKNF
N.PiF.BK.01.01 | Informacje o wskaźnikach LCR

LCR		Należy wprowadzić wartości wyliczone na podstawie historycznych dziennych wskaźników LCR			
		I kwartał RRRR	II kwartał RRRR	III kwartał RRRR	IV kwartał RRRR
		0010	0020	0030	0040
LCR - najniższa dzienna wartość wskaźnika w kwartale	0010				
LCR - najwyższa dzienna wartość wskaźnika w kwartale	0020				
LCR - średnia wartość wskaźnika w kwartale	0030				
LCR - mediana wskaźnika w kwartale	0040				

Rysunek 10: Układ mieszany

5.2 Opis procesu ekstrakcji danych z raportu

Z założeń wynikających w podpunkcie 4.1. procesem ekstrakcji zostały objęte poniższe dane:

- nazwa arkusza
- kody pozycji
- dane finansowe

Dane finansowe zostały wyekstrahowane w odniesieniu do współrzędnych kodów pozycji. Poniżej znajdują się szczegółowe opisy etapów algorytmu ekstrakcji.

5.2.1 Wyodrębnienie nazwy arkusza

Pierwszym krokiem jest pozyskanie wszystkich nazw arkuszy z danego pliku Excel.

5.2.2 Usunięcie kolumn pomocniczych

W następnym kroku następuje usunięcie z arkusza kolumn, które występują w arkuszach nieregularnie i utrudniają proces ekstrakcji.

5.2.3 Ekstrakcja kodów pozycji

Trzy różne układy danych tabelarycznych oznaczają trzy różne ułożenia kodów pomocniczych (tzw. datapoint'ów). Początek procesu ich ekstrakcji rozpoczyna się więc od zidentyfikowania, który z trzech układ jest ekstrahowany.

Za identyfikację odpowiada funkcja `find_sequence_positions`, a jej działanie polega najpierw na odnalezieniu datapoint'ów "0010" oraz sprawdzenia ich liczby. Jedna komórka "0010" oznacza układ horyzontalny lub wertykalny. Dwie komórki "0010" oznaczają układ mieszany, czyli zarówno horyzontalny jak i wertykalny. Następnie algorytm odnajduje resztę sekwencji datapoint'ów, w poniższy sposób:

- W przypadku jednej komórki "0010" algorytm porusza się w dół i w prawo szukając kontynuacji datapoint'ów oraz dodatkowo pomija wartości puste. Jeżeli kontynuacja odnaleziona zostanie w prawym kierunku oznacza to, że układ jest horyzontalny, a jeżeli w dolnym kierunku układ jest wertykalny. Współrzędne odnalezionych datapoint'ów zostają więc zwracane przez funkcję w odpowiednich listach: `positions_horizontal` lub `positions_vertical`
- W przypadku dwóch komórek "0010" algorytm porównuje numer wierszy we współrzędnych komórkach. Z rozmieszczenia datapoint'ów we wszystkich arkuszach można założyć, że komórka z niższym numerem wiersza rozpoczyna sekwencję horyzontalną, a komórka z wyższym numerem sekwencję wertykalną. Dalej algorytm wyszukuje kontynuację datapoint'ów z pominięciem wartości pustych (odpowiednio w dół lub w prawo) i zwraca ich współrzędne do obu list `positions_horizontal` i `positions_vertical`.

5.2.4 Ekstrakcja danych finansowych

W zależności jakiej funkcja `find_sequence_positions` zwraca współrzędne, algorytm wybiera jedną z trzech strategii ekstrakcji:

- Układ mieszany: oba zbiory datapoint'ów są obecne → `extract_intersections`
Funkcja przeszukuje dane znajdujące się na skrzyżowaniu datapoint'ów zidentyfikowanych zarówno w kolumnach, jak i w wierszach. Dla każdej niepustej komórki w tej siatce tworzony jest wpis w formacie:

```
{datapoint_poziomy}x{datapoint_pionowy}: wartość
```

- Układ horyzontalny: tylko datapoint'y w poziomie → `extract_from_horizontal`
Funkcja stosowana w przypadku, gdy datapointy występują w górnym wierszu (w poziomie). Każda kolumna traktowana jest jako osobny datapoint, a dane znajdują się bezpośrednio pod nim. Dla każdej niepustej komórki, tworzony jest wpis w formacie:

```
{datapoint_poziomy}: wartość
```

- Układ wertykalny: tylko Datapointy w pionie → `extract_from_vertical`
Funkcja stosowana w przypadku, gdy Datapointy występują w kolumnie po lewej stronie (w pionie). Każdy wiersz traktowany jest jako osobny Datapoint, a dane

znajdują się po jego prawej stronie. Analogicznie dla każdej niepustej komórki, tworzony jest wpis w formacie:

`{datapoint_pionowy}: wartość`

Każdy wyodrębniony element jest reprezentowany jako para klucz-wartość.

5.2.5 Zapis danych z raportu do formatu JSON

Dla każdego arkusza generowany jest jeden plik transportowy w formacie JSON, zawierający dane wyodrębnione z tabel. Struktura pliku przyjmuje postać:

```
{
  "sheet_name": [
    {
      "datapointA x datapointB": wartość
    },
    {
      "datapointC x datapointD": wartość
    },
    ...
  ]
}
```

- `sheet_name` – nazwa arkusza kalkulacyjnego, z którego pochodzi zestaw danych.
- `datapointA x datapointB` - klucz reprezentuje kontekst danej wartości.
- `data` – wartość odpowiadająca danemu kluczowi.

f

6 Automatyczny proces ładowania danych do bazy MSSQL

Proces ładowania danych do relacyjnej bazy danych MSSQL został podzielony na dwa główne etapy:

- załadunek struktury taksonomii,
- załadunek danych z raportu

Całość realizowana jest przez aplikację FastAPI udostępniającą interfejsy dostępne (API) do inicjalizacji każdego z tych etapów.

6.1 Połączenie z MSSQL za pomocą SQLAlchemy

Do komunikacji z bazą danych wykorzystywana jest biblioteka SQLAlchemy. Połączenie realizowane jest za pomocą sterownika ODBC oraz mechanizmu uwierzytelniania Windows. Każda operacja wykonywana jest wewnątrz transakcji, przy użyciu konstrukcji `engine.begin()`.

6.2 Utworzenie tabel w MSSQL

Po połączeniu z MSSQL funkcja `create_tables` tworzy puste tabele (4.1) jeśli nie istnieją.

6.3 Załadunek struktury taksonomii

Struktura taksonomii jest ładowana z dwóch rodzajów plików transportowych. Pierwszy rodzaj to wcześniej utworzony plik transportowy w formacie JSON, zawierający nazwę oraz wersję taksonomii (opis ekstrakcji znajduje się w podpunkcie 3.2). Drugi rodzaj to utworzona struktura taksonomii — liczba tych plików transportowych odpowiada liczbie arkuszy w obrębie formularzy. Pliki te również zapisane są w formacie JSON (szczegółowy opis znajduje się w podpunkcie 3.10).

6.3.1 Logika załadunku struktury

Proces załadunku struktury taksonomii do wcześniej utworzonych tabel **Taxonomy**, **Datapoints**, **Labels**, **Forms** realizuje następującą logikę:

1. Odczytywany jest plik transportowy z nazwą i wersją taksonomii. W bazie danych w tabeli **Taxonomy** następuje sprawdzenie czy ładowana wersja taksonomii znajduje się już w bazie danych. Jeżeli tak - program przerywa działanie procesu. Jeżeli nie - do tabeli wstawiany jest nowy wpis odpowiadający tej wersji wraz z nazwą i kontynuowany jest proces ładowania.

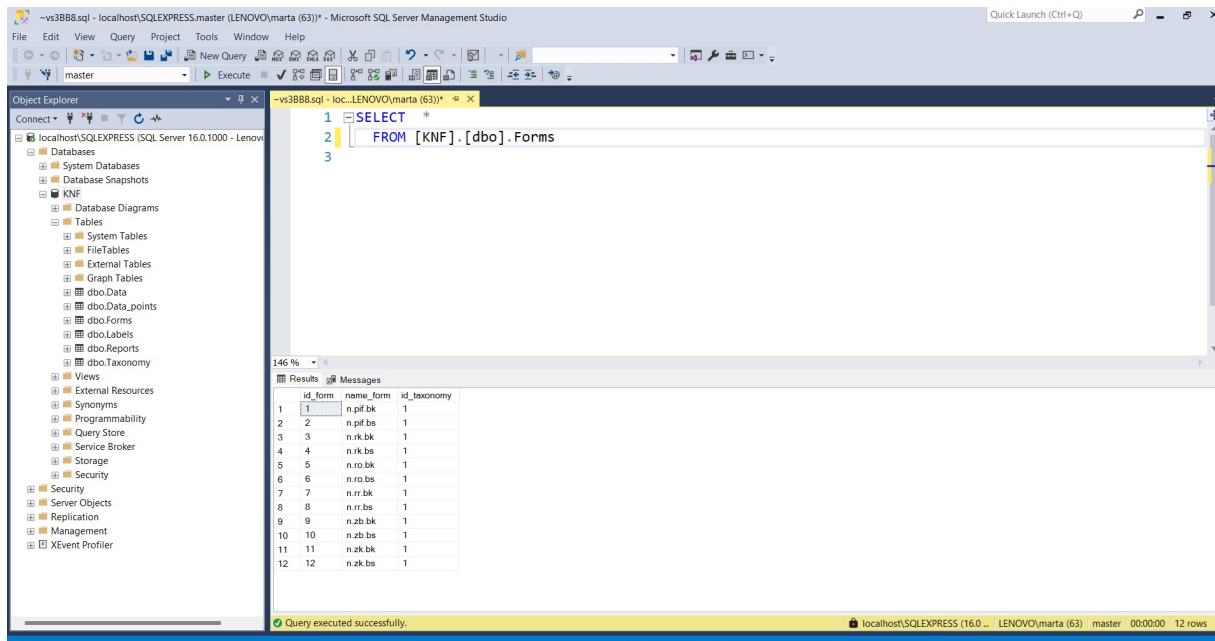
The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The Object Explorer on the left displays the database structure for 'localhost\SQLEXPRESS (SQL Server 16.0.1000 - LENOVO)'. The SQL Query window shows a query: `SELECT * FROM [KNF].[dbo].[Taxonomy]`. The Results window displays the query output:

id_taxonomy	version	name
1	1.0	BION

The status bar at the bottom indicates 'Query executed successfully.'

Rysunek 11: Tabela Taxonomy w bazie danych

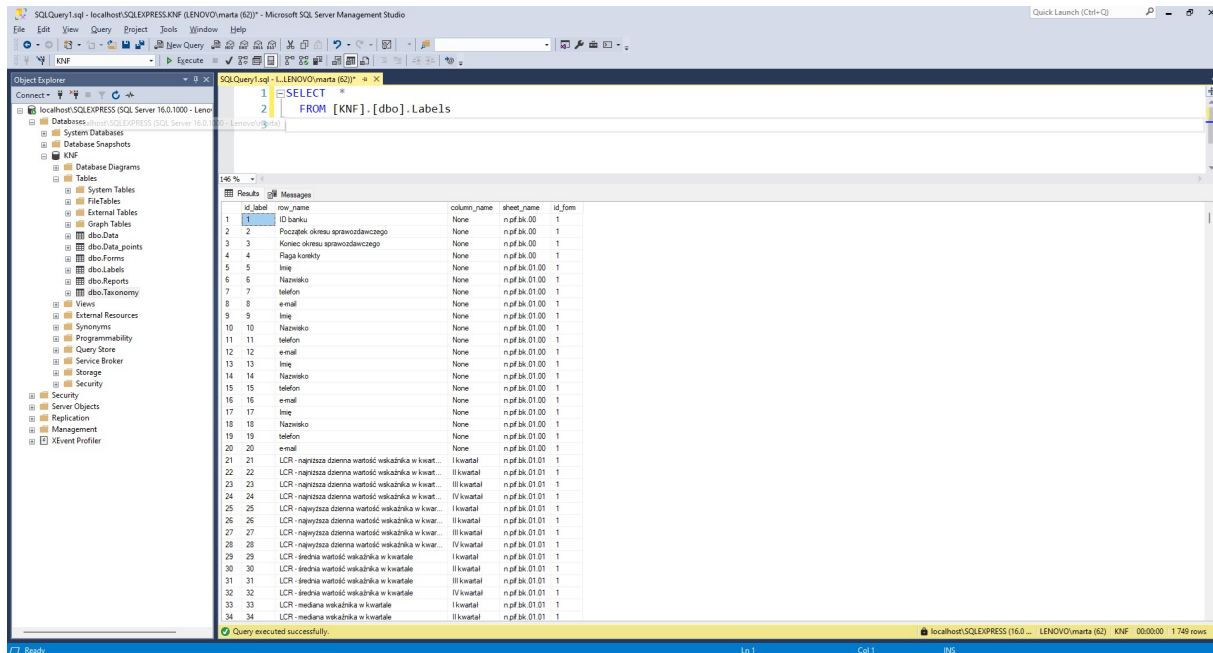
2. Dla każdej wartości `form_name` zdefiniowanej w strukturze:
- (a) Tworzony jest rekord w tabeli `Forms`, zawierający nazwę formularza (`name_form`) oraz powiązany identyfikator taksonomii (`id_taxonomy`),



Rysunek 12: Tabela Forms w bazie danych

3. Następnie, dla każdego elementu opisanego w polu data:

- Do tabeli **Labels** dodawany jest rekord zawierający:
 - * unikalny id rekordu (**id_label**),
 - * nazwa wiersza (**row_name**),
 - * nazwa kolumny (**column_name**),
 - * nazwa arkusza (**sheet_name**),
 - * powiązanie z identyfikatorem formularza (**id_form**).



The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The 'Object Explorer' on the left displays the database structure, including the 'KMF' database and its tables. The 'Query Editor' in the center shows a SQL query: `SELECT * FROM [KMF].[dbo].[Labels]`. The 'Results' pane on the right displays the data from the 'Labels' table, which includes columns for `id_label`, `row_name`, `column_name`, `sheet_name`, and `id_form`. The data is organized into 34 rows, with the first 10 rows representing individual data points and the remaining 24 rows representing aggregated data for various quarters.

id_label	row_name	column_name	sheet_name	id_form
1	1	(ID banku	None	n.pf.bk.00
2	2	Początek okresu sprawozdawczego	None	n.pf.bk.00
3	3	Koniec okresu sprawozdawczego	None	n.pf.bk.00
4	4	Rega korekty	None	n.pf.bk.00
5	5	Imię	None	n.pf.bk.01.00
6	6	Nazwisko	None	n.pf.bk.01.00
7	7	telefon	None	n.pf.bk.01.00
8	8	email	None	n.pf.bk.01.00
9	9	Imię	None	n.pf.bk.01.00
10	10	Nazwisko	None	n.pf.bk.01.00
11	11	telefon	None	n.pf.bk.01.00
12	12	email	None	n.pf.bk.01.00
13	13	Imię	None	n.pf.bk.01.00
14	14	Nazwisko	None	n.pf.bk.01.00
15	15	telefon	None	n.pf.bk.01.00
16	16	email	None	n.pf.bk.01.00
17	17	Imię	None	n.pf.bk.01.00
18	18	Nazwisko	None	n.pf.bk.01.00
19	19	telefon	None	n.pf.bk.01.00
20	20	email	None	n.pf.bk.01.00
21	21	LCR - najniższa dzienne wartość wskaźnika w kwart...	I kwartał	n.pf.bk.01.01
22	22	LCR - najniższa dzienne wartość wskaźnika w kwart...	II kwartał	n.pf.bk.01.01
23	23	LCR - najniższa dzienne wartość wskaźnika w kwart...	III kwartał	n.pf.bk.01.01
24	24	LCR - najniższa dzienne wartość wskaźnika w kwart...	IV kwartał	n.pf.bk.01.01
25	25	LCR - najwyższa dzienne wartość wskaźnika w kwart...	I kwartał	n.pf.bk.01.01
26	26	LCR - najwyższa dzienne wartość wskaźnika w kwart...	II kwartał	n.pf.bk.01.01
27	27	LCR - najwyższa dzienne wartość wskaźnika w kwart...	III kwartał	n.pf.bk.01.01
28	28	LCR - najwyższa dzienne wartość wskaźnika w kwart...	IV kwartał	n.pf.bk.01.01
29	29	LCR - średnia wartość wskaźnika w kwartale	I kwartał	n.pf.bk.01.01
30	30	LCR - średnia wartość wskaźnika w kwartale	II kwartał	n.pf.bk.01.01
31	31	LCR - średnia wartość wskaźnika w kwartale	III kwartał	n.pf.bk.01.01
32	32	LCR - średnia wartość wskaźnika w kwartale	IV kwartał	n.pf.bk.01.01
33	33	LCR - mediana wskaźnika w kwartale	I kwartał	n.pf.bk.01.01
34	34	LCR - mediana wskaźnika w kwartale	II kwartał	n.pf.bk.01.01

Rysunek 13: Tabela Labels w bazie danych

- Następnie dla każdego elementu listy `data_points` tworzony jest rekord w tabeli `Data_points`, zawierający:
 - * unikalny id rekordu (`id_data_pint`),
 - * datapoint (`data_point`),
 - * nazwa metryki (`qname`),
 - * typ danych (`data_type`),
 - * klucz obcy wskazujący na odpowiadający rekord z tabeli `Labels` (`id_label`)

SQL Query Log - L:\ENOVUmarta (82) - Microsoft SQL Server Management Studio

Object Explorer: Localhost\SQLEXPRESS (SQL Server 16.0.1000 - Lenovo)

Query: `SELECT FROM [KNF].[dbo].Data_points`

	id_data_pint	id_label	data_point	qname	data_type
1	1	1	0010	a2101	xakt.datetimeType
2	2	2	0020	a2273	xakt.datetimeType
3	3	3	0030	a2274	xakt.datetimeType
4	4	4	0040	a1	xakt.datetimeType
5	5	5	0010	a2102	xakt.datetimeType
6	6	6	0020	a2103	xakt.datetimeType
7	7	7	0030	a2104	xakt.datetimeType
8	8	8	0040	a2105	xakt.datetimeType
9	9	9	0050	a2102	xakt.datetimeType
10	10	10	0060	a2103	xakt.datetimeType
11	11	11	0070	a2104	xakt.datetimeType
12	12	12	0080	a2105	xakt.datetimeType
13	13	13	0090	a2102	xakt.datetimeType
14	14	14	0100	a2103	xakt.datetimeType
15	15	15	0110	a2104	xakt.datetimeType
16	16	16	0120	a2105	xakt.datetimeType
17	17	17	0130	a2102	xakt.datetimeType
18	18	18	0140	a2103	xakt.datetimeType
19	19	19	0150	a2104	xakt.datetimeType
20	20	20	0160	a2105	xakt.datetimeType
21	21	21	0010x0010	a2107	num.percentbenType
22	22	22	0010x0020	a2107	num.percentbenType
23	23	23	0010x0030	a2107	num.percentbenType
24	24	24	0010x0040	a2107	num.percentbenType
25	25	25	0020x0010	a2108	num.percentbenType
26	26	26	0020x0020	a2108	num.percentbenType
27	27	27	0020x0030	a2108	num.percentbenType
28	28	28	0020x0040	a2108	num.percentbenType
29	29	29	0030x0010	a2109	num.percentbenType
30	30	30	0030x0020	a2109	num.percentbenType
31	31	31	0030x0030	a2109	num.percentbenType
32	32	32	0030x0040	a2109	num.percentbenType
33	33	33	0040x0010	a2111	num.percentbenType
34	34	34	0040x0020	a2111	num.percentbenType

Query executed successfully. 1749 rows

Rysunek 14: Tabela `Data_points` w bazie danych

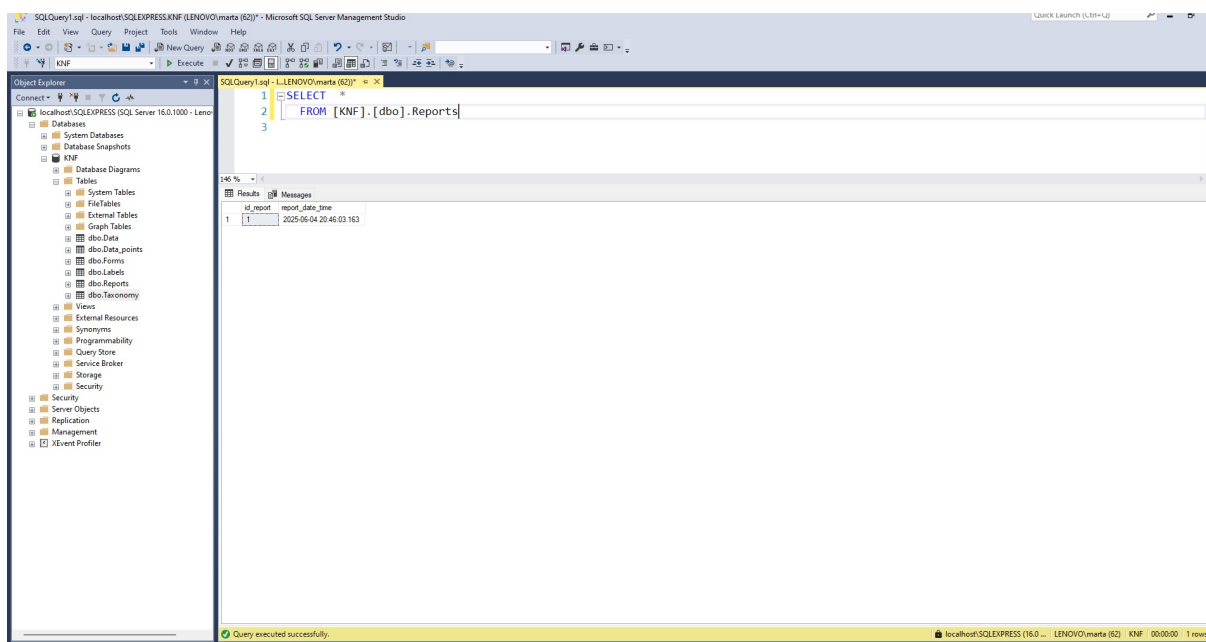
6.4 Załadunek danych z raportów

Proces ładowania danych z raportu następuje z wcześniej wyekstrahowanych plików transportowych (5.2.5). Jeden plik JSON zawiera dane w kontekście konkretnego arkusza z formularza EXCEL.

6.4.1 Logika załadunku danych z raportów

Proces załadunku do wcześniej utworzonych tabel **Data** i **Reports** obejmuje:

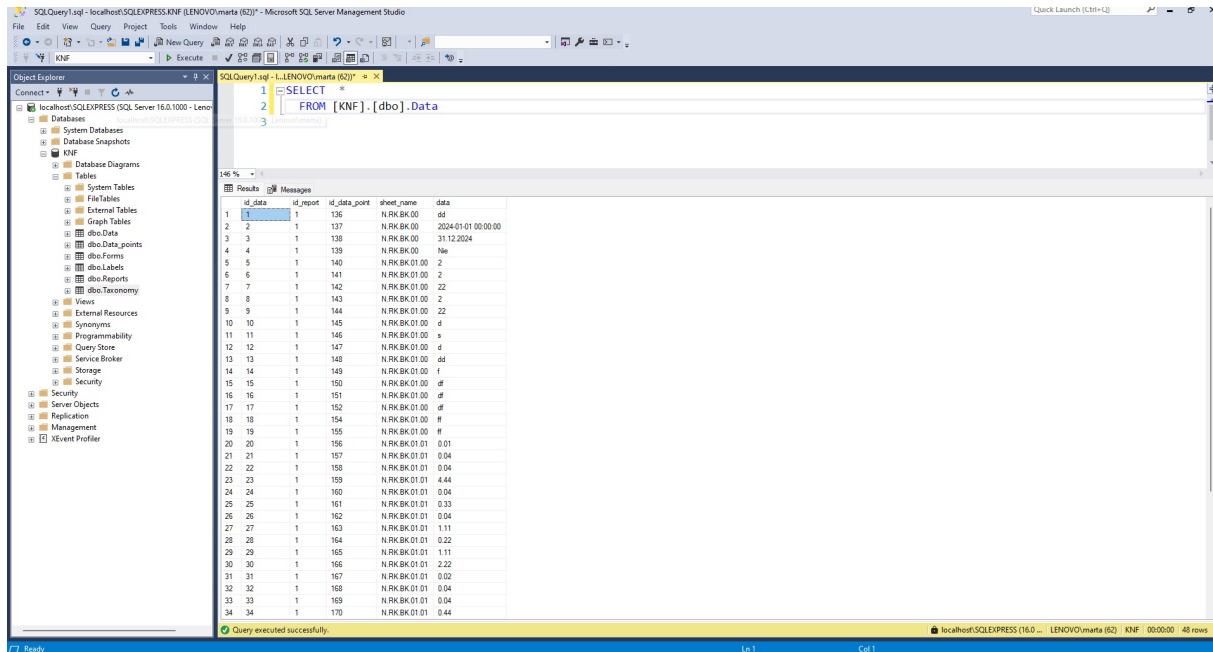
1. Dla jednego wyekstrahowanego raportu zostanie dodany nowy rekord do tabeli **Reports** rekord będzie zawierał:
 - unikalny identyfikator rekordu (**id_report**),
 - datę rozładowanego raportu (**report_date_time**)



Rysunek 15: Tabela Reports w bazie danych

2. Dla każdego arkusza (**sheet_name**) występującego w plikach transportowych:
 - (a) Na podstawie nazwy arkusza (**sheet_name**) pobierane są wszystkie punkty danych (**data_point**) z tabeli **Data_points**. Zapytanie realizuje to poprzez połączenie tabeli **Data_points** z tabelą **Labels** za pomocą klucza obcego **id_label**. Następnie wybierane są tylko te rekordy, dla których wartość pola **sheet_name** w tabeli **Labels** odpowiada nazwie arkusza.
 - (b) Dla każdej pary **data_point**: wartość znajdującej się w pliku JSON:
 - Wykonywana jest walidacja – sprawdzana jest spójność danych. Każdy **data_point** jest weryfikowany pod kątem obecności w strukturze taksonomii, w odniesieniu do **data_point** wcześniej wyekstrahowanego z raportu. Jeżeli dany punkt nie występuje w taksonomii, proces zostaje przerwany, a użytkownik otrzymuje komunikat o błędzie spójności.

- Jeżeli struktura jest spójna, tworzony jest rekord w tabeli Data, zawierający:
 - identyfikator raportu (`id_report`),
 - identyfikator punktu danych (`id_data_point`),
 - nazwę arkusza (`sheet_name`),
 - wartość (`data`).
 - klucz obcy (`id_label`)



Rysunek 16: Tabela Data w bazie danych

7 Obsługa programu

7.1 Instalacja bibliotek

Zainstaluj wymagane zależności i biblioteki za pomocą poniższego polecenia:

```
1 pip install -r .\requirements.txt
```

7.2 Tworzenie katalogów

Aby stworzyć potrzebne katalogi należy uruchomić skrypt `src/main.py`. Skrypt ten automatycznie utworzy wszystkie niezbędne katalogi. W terminalu skorzystaj z polecenia:

```
1 python src/main.py
```

7.3 Użycie programu

Po uruchomieniu programu `src/main.py` w terminalu wyświetli się menu dla użytkownika, a zaraz po nim prośba o wybranie numeru akcji.

```
1 === MENU PROGRAMU ===
2 Wybierz jedna z opcji:
3
4 1 - Tworzenie struktury bazodanowej na podstawie taksonomii
5 2 - Ekstrakcja danych z raportu
6 3 - Ladowanie struktury bazodanowej do bazy danych
7 4 - Ladowanie danych z raportu do bazy danych
8 0 - Wyjscie z programu
9
10 =====
11
12 Akcja:
```

7.3.1 Tworzenie struktury bazodanowej na podstawie taksonomii

Do powstałego katalogu `data/taxonomy` należy wrzucić model taksonomii, przed wybraniem akcji.

Input

```
KNF/
  data/
    taxonomy/
      Taxonomy
```

Po wrzuceniu modelu taksonomii można w terminalu wpisać wartość 1, aby uruchomić skrypt realizujący to zadanie.

Output

```
KNF/  
structure/  
  full_structure/  
    form_name_1.json  
    form_name_2.json  
    form_name_3.json  
  taxonomy_info/  
    taxonomy_info.json
```

Skrypt, na podstawie dostarczonej taksonomii, wygeneruje pliki transportowe. Liczba tych plików — każdy zawierający strukturę dla konkretnego arkusza — odpowiada liczbie arkuszy we wszystkich formularzach. Każdy plik zostanie zapisany w formacie JSON i nazwany zgodnie z nazwą odpowiadającego mu arkusza. Ponadto, skrypt zapisze plik `taxonomy_info.json` w katalogu `taxonomy_info`, zawierający nazwę oraz wersję taksonomii.

7.3.2 Ekstrakcja danych z raportu

Do katalogu `data/reports` należy wrzucić jeden raport w formacie `.xls` lub `.xlsx` z uzupełnionymi danymi, przed wykonaniem akcji.

Input

```
KNF/  
data/  
  reports/  
    report.xlsx
```

Po wrzuceniu raportu można w terminalu wpisać wartość 2, aby uruchomić skrypt.

Output

```
KNF/  
report_data/  
  sheet_name1.json  
  sheet_name2.json  
  sheet_name3.json
```

Skrypt na podstawie wrzuconego raportu wyekstrahuje dane z raportu oraz utworzy tyle plików transportowych, ile jest arkuszy w danym formularzu.

7.3.3 Ładowanie struktury bazodanowej do bazy danych

Przed załadunkiem struktury bazodanowej trzeba najpierw wykonać akcję nr 1, aby utworzyć pliki transportowe (pełną strukturę dla taksonomii zapisaną w formatach JSON). Należy uruchomić API w terminalu, jeśli tego jeszcze nie zrobiłeś.

```
1 uvicorn src.api_interface.api:app --reload
```

Po poprawnym uruchomieniu dostaniemy informację: "INFO: Application startup complete". Teraz możemy uruchomić akcję i wpisać w terminal wartość 3. Skrypt załaduje Twoją strukturę do bazy danych i wyświetli w terminalu komunikat: "Struktura została załadowana do bazy danych", jeśli operacja zakończy się sukcesem.

7.3.4 Ładowanie danych z raportu do bazy danych

Przed załadunkiem danych do bazy danych upewnij się, że wykonałeś akcję 2. Należy uruchomić API w terminalu, jeśli tego jeszcze nie zrobiłeś.

```
1 uvicorn src.api_interface.api:app --reload
```

Po poprawnym uruchomieniu dostaniemy informację: "INFO: Application startup complete". Teraz możemy uruchomić akcję i wpisać w terminal wartość 4. Skrypt załaduje twoje wyekstrahowane dane do bazy danych i wyświetli w terminalu komentarz: "Raport został pomyślnie załadowany do bazy danych", jeśli operacja zakończy się sukcesem.

7.3.5 Wyjście z programu

Wybierz akcję 0, jeśli chcesz zakończyć program.