Budowa narzędzia wspierającego zarządzanie systemem CAR

Politechnika Poznańska 2017

Marcin Kamiński, Hubert Półtorak

2017

Spis treści

[1. Analiza wymagań 1](#_Toc486008751)

[1.1 Charakterystyka środowiska CAR 1](#_Toc486008752)

[1.2 Informacje dotyczące natury problemu zaobserwowane w funkcjonowaniu CAR. 4](#_Toc486008753)

[1.2.1 Rozszerzanie środowiska CAR o nowe charakterystyki. 4](#_Toc486008754)

[2. Zamierzony efekt prac 5](#_Toc486008755)

[3. Ogólna koncepcja rozwiązania 8](#_Toc486008756)

[3.1 Warstwa pierwsza 9](#_Toc486008757)

[3.2 Warstwa druga 14](#_Toc486008758)

[3.3 Warstwa trzecia 14](#_Toc486008759)

[4. Wykorzystanie narzędzi/komponentów 16](#_Toc486008760)

[4.1 Budowa środowiska 17](#_Toc486008761)

[4.1.1 Instalacja Anaconda Navigator w systemie Windows 18](#_Toc486008762)

[4.3 Konfiguracja narzędzi 24](#_Toc486008763)

[4.3.1 Konfiguracja komponentu "Spyder" 24](#_Toc486008764)

[4.3.2 Konfiguracja oprogramowania Gephi 28](#_Toc486008765)

[4.3.3 Konfiguracja oprogramowania CMT\_Widget 30](#_Toc486008766)

[5. Podsumowanie 33](#_Toc486008767)

# 

# 

## Analiza wymagań

## 1.1 Charakterystyka środowiska CAR

Główną ideą istnienia i funkcjonowania środowiska CAR jest zdefiniowanie integralnej i spójnej warstwy zagregowanych charakterystyk na poziomie logicznym. Pod pojęciem integralności kryję się dostępność predefiniowanych danych w jednym miejscu. Natomiast spójność oznacza spójność definicji i implementacji otrzymanych wyników dla wszystkich procesów biznesowych. Warstwa danych powinna być na tyle kompletna, aby przy jej użyciu możliwe było uzyskanie przekrojowych informacji na temat klienta oraz jego produktów oraz wykorzystanie ich w ściśle zdefiniowanych procesach biznesowych. Produkty posiadane przez klienta zostały określone w trzech głównych grupach, które dotyczą:

* Produktów bankowych,
* Produktów nabytych w innych instytucjach,
* Usług (np. elektroniczne, karty debetowe, zlecenia stałe, polecenia zapłaty, alerty, ubezpieczenia do kredytów).

Środowisko CAR w swojej ogólności stanowi pełen zbiór charakterystyk wykorzystywanych w procesach biznesowych, które wykazują zapotrzebowanie korzystania ze środowiska. To właśnie wspomniane charakterystyki są produktem finalnym będącym pochodną integralnej i spójnej warstwy danych.

W celu zapewnienia jak najbardziej przekrojowego spojrzenia na całe środowisko oraz łatwości w nawigacji poszczególnych grup informacyjnych dane w CAR zostały podzielone na obszary oraz podobszary. Ich specyfikacja jest uzależniona od grup tematycznych, których dotykają dane opisujące dany obszar. Co więcej wspomagają one utrzymanie unikalności zmiennych oraz w definicji mają zagwarantować łatwiejsze i wydajniejsze zarządzanie charakterystykami pod kątem biznesowym. W całym środowisku wydzielono następujące obszary:

* 4 obszary klientowskie
* 5 obszarów produktowych
* 1 obszar zabezpieczeń
* 1 obszar relacji
* 3 obszary słownikowe

Każdy obszar składa się ze zbioru tabel, w których następuje składowanie oraz aktualizacja danych osadzona w wystandaryzowanym procesie. Większość przetwarzań jest wykonywana równolegle, jednak z zachowaniem kolejności wynikających z zależności pomiędzy charakterystykami (tzw. przetwarzania wielokrokowe czyli takie, w których w celu wyliczenia wartości pewnej charakterystyki korzystamy z innych wartości charakterystyk wyliczonych w kroku poprzednim). Proces obejmuje retencje danych w częstotliwości dziennej D-1 (dane na wczoraj). Ponadto w trakcie opracowywania jest część systemu zawierająca dane on-line, jednak ta część nie jest przedmiotem niniejszego projektu.

Źródłem danych do CAR są miejsca, w których składowanie danych może odbyć się jedynie dzięki zastosowaniu zaawansowanych technik z zakresu ETL, co sprawia, iż wszelkie informacje trafiające finalnie do każdego z obszarów są bardzo wysokiej jakości. Źródłami, z których pobierane są dane do środowiska CAR jest:

* Korporacyjna Hurtownia Danych (KHD) - podstawowym źródłem danych dziennych i miesięcznych jest system KHD. Jeżeli dane nie są dostępne w systemie danych KHD, powinien zostać powołany projekt dodający ten zakres informacji Jeżeli nie jest to możliwe, do CAR pobierane są dane z innych źródeł.
* ODS - jeżeli nie jest możliwe pobranie odpowiedniego pakietu danych przy użyciu systemu KHD, wówczas dane pobierane są bezpośrednio z systemów źródłowych (produkcyjnych), które są następnie przeładowywane do dedykowanej bazy technicznej CAR (ODS) na tej samej maszynie produkcyjnej co KHD i CAR.

W ramach każdego obszaru zostały wydzielone podobszary stanowiące kolejny poziom szczegółowości. CAR zawiera charakterystyki dotykające różnych grup aktywności klienta. Charakterystyka może bazować na zmiennych z różnych obszarów oraz podobszarów. Kolejność wyliczania charakterystyk zdefiniowana jest w dokumentach CMT. Jeden dokument CMT definiuje jeden podobszar CAR. W ramach dokumentu zdefiniowane jest jedna lub wiele zakładek odpowiadających pojedynczym zapytaniom SQL. Pliki CMT muszą być tworzone zgodnie ze ściśle zdefiniowanym formatem, zachowując jednocześnie standardy pozwalające na ich wykorzystanie w procesie zasilania. To one stanowią jedyne źródło informacyjne dla kodów zastosowanych do wyliczania charakterystyk, co poza samymi charakterystykami obejmuje najniższy poziom szczegółowości w całym środowisku CAR. Wspomniane zakładki pliku CMT są miejscem składowania tychże kodów, a każda pojedyncza zakładka dotyczy jednego SQL Part, czyli kodu SQL napisanego w dialekcie Teradata, a następnie przekształconego zgodnie z pewnymi standardami na język rozpoznawany przez pakiet narzędzi odpowiadających za proces zasilania oraz za ostateczne wyliczanie charakterystyk. Na chwilę obecną całe środowisko CAR składa się z ok. 850 SQL Part odpowiadających za wyliczanie ok. 1,7 tys. Charakterystyk.

Załącznik prezentuje pusty szablon pliku CMT.



## 1.2 Informacje dotyczące natury problemu zaobserwowane w funkcjonowaniu CAR.

Istotą istnienia i funkcjonowania CAR jest kreowanie maksymalnie przekrojowego obrazu na temat klienta, jego produktów, trendów, zachowań, preferencji, potencjałów czy aktywności. Środowisko może rozrastać się w sposób niezwykle dynamiczny osiągając przy tym rozmiary, które w definicji mogą być limitowane poprzez:

* moce obliczeniowe,
* przestrzeń składowania danych,
* czas przetwarzania lub udostępnienia danych dla biznesu,
* zaangażowanie czynnika ludzkiego w rozwój i utrzymanie środowiska,
* wreszcie przez potrzeby biznesowe.

Rozszerzanie zakresu danych w CAR podyktowane jest potrzebami biznesowymi w obszarach CRM oraz ryzyka kredytowego, dzięki czemu odbiorcy biznesowi są w stanie dotrzeć do klienta z coraz lepszymi ofertami produktów bankowych. W tym jednak miejscu można zaobserwować pewne niedogodności natury optymalizacyjnej oraz wydajnościowej wynikające ze sposobu dodawania nowych charakterystyk do CAR.

### 1.2.1 Rozszerzanie środowiska CAR o nowe charakterystyki.

Proces rozszerzania środowiska CAR o dodatkowe charakterystyki polega na stworzeniu kodu będącego nowym SQL Part. Aby wyliczane charakterystyki mogły finalnie uczestniczyć w procesie generowania niezbędne jest określenie obszaru, a następnie podobszaru, do którego zostaną przyporządkowane. Kolejnym krokiem jest wydzielenie odrębnej zakładki w dedykowanym pliku CMT, który będzie stanowił miejsce składowania kwerendy. Na tym etapie pojawia się największa niedogodność związana z rozwojem środowiska, ponieważ walidacja dodawanego SQL Part opiera się jedynie na sprawdzeniu kodu pod kątem optymalności jego napisania. Ostatnią fazą, która determinuje możliwość wdrożenia nowych charakterystyk jest zrewidowanie zwracanych wyników oraz dostosowanie kodu pod kątem umożliwiającym jego implementacje w narzędziu odpowiadającym za proces zasilania i generowania.

Na tym etapie można dostrzec, iż warstwa danych obsługująca CAR pozostaje niezmienna w pewnym czasie, co przekłada się na wykorzystywanie tych samych tabel i ich atrybutów na poziomie wielu SQL Part. Wszelkie warunki selekcji, projekcji lub transformacje poszczególnych atrybutów mogą znajdować swoje zastosowanie w całym szeregu SQL Part. Implikuje to wiele niedogodności związanych z wydajnością oraz optymalizacją działania środowiska CAR. Patrząc przekrojowo na dane wykorzystane w CAR można zauważyć powszechne wykorzystanie kilku lub kilkunastu tabel wraz z ich atrybutami oraz użytymi warunkami mimo tego, że każda charakterystyka jest unikalna w przekroju całego środowiska, a jej zastosowanie rozszerza funkcjonalność procesów biznesowych. Należy mieć jednak na uwadze, że wiele nowych charakterystyk może być wyliczane przy pomocy istniejących SQL Part, a zabiegi związane z ich implementacją ograniczały by się jedynie w dużej mierze do nieznacznych modyfikacji kwerend zawierających potrzebne grupy danych. Pomimo, że wspomniany podział na obszary i podobszary stanowi dobry sposób pozwalający na kategoryzacje danych grup charakterystyk, to na etapie identyfikacji powtarzających się źródeł stanowi niemałe wyzwanie. Rozproszona forma składowania SQL Part przekłada się na brak takich możliwości przy zastosowaniu technik ograniczających się do wykorzystania czynnika ludzkiego. Bardzo ciężkie, a w zasadzie niemożliwe jest wychwycenie pewnych prawidłowości związanych z wyliczaniem charakterystyk. Taka forma magazynowania SQL Part stanowi wsparcie techniki CMT, która w definicji powinna ułatwić pracę analitykom zaangażowanym w proces oraz optymalizować proces pobierania danych ze źródeł. Rozwój samego środowiska uwarunkowany jest jednak poprzez wiele innych czynników, których analiza potwierdza przesłanki na temat tego, że wykorzystywanie tych samych bloków kwerend w różnych SQL Part znacznie wpływa na optymalne wykorzystanie zasobów CAR.

# 2. Zamierzony efekt prac

Wszelkie działania powadzone w ramach przedstawionych w rozdziale pierwszym narzędzi mają za zadanie usprawnienie oraz optymalizację całokształtu prac prowadzonych obszarze CAR. Tego typu akcje znajdą również zastosowanie w wielu kwestiach biznesowych, ponieważ powinny umożliwić zdecydowanie szybszą analizę poszczególnych komponentów CAR. W rezultacie możliwe będzie uzyskanie natychmiastowej odpowiedzi na wiele pytań biznesowych. Dzięki takim zabiegom rozwój środowiska CAR będzie powinien przebiegać w zdecydowanie wydajniejszy sposób. Poza tym zaproponowane rozwiązanie pozwoli na większą kontrolę nowych obiektów dodawanych do środowiska. Główna koncepcja rozwiązania polega na koncentracji wszystkich SQL Part w jednym miejscu, która mimo dużych rozmiarów CAR zapewni kompaktowość działania. Zalety skupienia wszystkich kwerend SQL w jednym miejscu będą widoczne również poprzez możliwość natychmiastowej analizy pod kątem występowania poszczególnych bloków kwerend. Spodziewany efekt prac został określony w trzech głównych grupach, które przedstawiają szczegółową koncepcje.

Pierwsza grupa dotyczy budowy parsera analizującego poszczególne zapytania SQL wchodzące w skład CAR. Podstawą do przeprowadzenia takiej analizy jest zbiór wszystkich plików CMT. Analiza polegać będzie na rozbiciu każdego SQL Part na czynniki pierwsze i wydzieleniu głównych składowych będącymi warunkami selekcji, projekcji i grupowania, użytych tabel, a także wszelkich transformacji dokonanych na poszczególnych atrybutach. Wyniki podziału SQL Part będą odkładane do dedykowanej w tym celu sieci (grafu), która ostatecznie będzie zawierała struktury wszystkich kodów wchodzących w skład środowiska CAR. Wyznacznikiem finalizacji działań przewidzianych dla tej grupy jest maksymalna koncentracja wszystkich SQL Part.

Ograniczenie się jedynie do tego typu działań nie zapewnia jednak możliwości zaobserwowania współzależności. Kolejnym krokiem, po tak przeprowadzonych działaniach jest eksport ostatecznie utworzonej sieci do pliku płaskiego, który zostanie wykorzystany jako plik wsadowy w narzędziach umożliwiających złożone analizy graficzne bądź pozwalających na selektywne analizy logiczne polegające na dopasowywaniu nowych struktur do tych już istniejących. Graficzne przedstawienie zintegrowanej struktury środowiska CAR pozwoli na szczegółową analizę częstotliwości występowania danych komponentów na poziomie wszystkich SQL Part. Zamierzonym efektem prac jest utworzenie metabazy grafowej pokazującej przepływy wszystkich ładunków informacyjnych. Wizualizacja przyjmie postać ogromnego grafu spajającego wszystko to, co zostało utworzone przy pomocy parsera, a co za tym idzie możliwe będzie uzyskanie wszelkich informacji na temat SQL Part na poziomie całego środowiska CAR.

Poprzez finalizacje działań prowadzonych w grupie pierwszej możliwe jest przejście do kolejnego i niemniej ważnego etapu zastosowania narzędzia. Opiera się ono na rozszerzeniu działania narzędzia o funkcje, które zapewnią możliwość sugerowania zmiany kodów SQL służących do zasilenia CAR. Na tym etapie funkcjonalność narzędzia jest ściśle powiązana z dodawaniem nowych SQL Part. . Rozszerzenie obszaru CAR o nowe charakterystyki polega na modyfikacji oraz aktualizacji pliku CMT zgodnie ze ściśle określoną metodologią. Opierając się na tym, co zostało opisane w rozdziale pierwszym rozszerzenie środowiska CAR o nowe charakterystyki wymaga utworzenia kodu SQL, który docelowo powinien znaleźć się w pliku CMT, jako kolejna zakładka. Taki sposób sprawia, że te same tabele i ich atrybuty mogą wielokrotnie stanowić element służący do wyliczania oddzielnych charakterystyk. Jest to dość spora niedogodność, która znajduje odzwierciedlenie we wszelkich wątkach zarówno wydajnościowych, jak i optymalizacyjnych. Zamierzonym efektem prac dla tej grupy jest możliwość analizy wszystkich SQL Part wchodzących w skład środowiska CAR i porównanie ich ze źródłem w postaci kodu SQL odpowiadającym za wyliczanie nowych charakterystyk. Wszystkie wykorzystane wcześniej elementy, które znajdą się w najnowszym SQL Part, a które znalazły zastosowanie w poprzednich zakładkach pliku CMT mogą zostać wykorzystane podczas wyliczania charakterystyk dla nowego SQL Part bez konieczności wielokrotnego implementowania ich do środowiska. Ponadto dzięki graficznemu ujęciu środowiska możliwe będzie nie tylko zidentyfikowanie powtarzających się elementów, ale również ich nawigacja. Węzły grafu odchodzące od danego wierzchołka skierowane będą do wierzchołków zawierających nazwy korespondujących z nimi SQL Part-ów. W związku tym wiadome będzie to, w których kwerendach wykorzystywany jest dany atrybut oraz, do którego pliku CMT należy się odnieść, aby wykonać zamierzone działania. Implementacja tego typu rozwiązania przełoży się na wzrost wydajności oraz optymalizacji środowiska.

Ostatnią wydzieloną grupą wśród funkcjonalności obejmujących opisywane narzędzie jest zastosowanie przydatne przede wszystkim na płaszczyźnie biznesowej. Jest ono ściśle powiązane z funkcjonalnością narzędzia, które zostało ukazane w grupie drugiej, jednak zastosowanie dotyczy odrębnej grupy tematycznej. Ukazanie całościowego przepływu ładunków informacyjnych powinno zdecydowanie ułatwić budowę rozmaitych prezentacji metabazy grafowej CAR. Implementacja takiego rozwiązania powinna przełożyć się w znacznej mierze na ułatwieniu analizy logiki dzięki możliwości ujrzenia przebiegu oraz uczestnictwa poszczególnych elementów na poziomie wszystkich SQL Part. Lineage stworzony dzięki nowemu rozwiązaniu pozwoli zdecydowanie szybciej uzyskać odpowiedzi na konkretne pytania, ponieważ na tym etapie zbędna będzie szczegółowa analiza poszczególnych zakładek plików CMT ze wszystkich obszarów CAR. Dzięki rozbudowanej metabazie grafowej możliwa będzie niemalże natychmiastowa odpowiedź na wiele pytań biznesowych.

Funkcjonalności opisane w ramach każdej grupy stanowią zamierzony efekt prac, którego zastosowanie jest widoczne zarówno pod kątem technicznym, jak i biznesowym. Wdrożenie opisanej koncepcji przekłada się na wzrost wydajności funkcjonowania całego środowiska poprzez ograniczenie równoległego przetwarzania tych samych bloków zapytań, które miałyby miejsce przy obecnej formie rozwoju CAR. Ograniczenie tego typu działań jest możliwe dzięki integracji wszystkich SQL Part będącego podstawą do stworzenia metabazy grafowej ukazującego "lineage" poszczególnych ładunków informacyjnych. Przekłada się to na znaczą optymalizację, zmniejszoną pracochłonność potrzebną do utrzymania środowiska, a także zdecydowanie łatwiejszą ścieżką modyfikacji nowowprowadzanych SQL Part.

# 3. Ogólna koncepcja rozwiązania

Koncepcja rozwiązania obejmuje trzy główne obszary, które na potrzeby dokumentu zostały nazwane warstwami. Działania w pierwszej warstwie obejmują:

* Stworzenie parsera plików CMT, którego główna funkcja sprowadza się do wydzielenia składowych ze wszystkich SQL Part.
* Stworzenia dedykowanej tabeli, do której zapisywane będą wyniki analiz.
* Eksport dedykowanej tabeli do pliku płaskiego.

Działania w drugiej warstwie obejmują:

* Zaimportowanie pliku płaskiego utworzonego w warstwie pierwszej.
* Utworzenie metabazy grafowej na podstawie pliku płaskiego, która będzie graficznym odwzorowaniem struktur wydzielonych w warstwie pierwszej.
* Stworzenie mechanizmu wizualizacji wybranych fragmentów bazy danych.

Działania w trzeciej warstwie obejmują:

* Stworzenie mechanizmu sugerującego zmiany podczas dodawania nowych SQL PART.

## 3.1 Warstwa pierwsza

Warunkiem koniecznym do rozpoczęcia prac w pierwszej warstwie jest możliwość uzyskania dostępu do pliku CMT. W tym celu zastosowano język Python w wersji 3.5. Za kompilator składni języka Python posłużył komponent o nazwie ""SPYDER"" oferowany w pakiecie Anaconda Navigaotor, którego konfiguracja została bliżej opisana w rozdziale czwartym. W celu stworzenia narzędzia zgodnego z zamierzonym efektem prac opisanym w rozdziale 2 niezbędne było zastosowanie technik pozwalających na:

* Pracę z plikami xlsx.
* Tworzenie struktur grafów poprzez kategoryzacje atrybutów zgodnie ze ściśle określonymi warunkami.
* Możliwość dopasowania operacji wykonywanych przy pomocy wyrażen regularnych.
* Zapis utworzonych struktur w dedykowanym zbiorze danych.

Spełnienie wyżej wymienionych wytycznych jest możliwe dzięki bibliotekom oferowanym bezpośrednio przez język Python. Funkcjonalność jaką zapewniają ściśle zazębia się z działaniami jakie powinny zostać podjęte w związku ze stworzeniem narzędzia. Biblioteki użyte w tej warstwie to:

* **Openpyxl** – biblioteka służąca do odczytu plików xlsx
* **Pathlib** – biblioteka oferująca klasy, które reprezentują ścieżki do plików systemowych
* **Networkx** – biblioteka służąca do tworzenia oraz modyfikacji grafów
* **Re** – biblioteka pozwalająca na implementację wyrażen regularnych
* **Matplotlib** – biblioteka służąca do tworzenia wykresów

Działania przeprowadzone w warstwie pierwszej przebiegają zgodnie ze schematem przedstawionym poniżej:

1. Odczyt konkretnych zakładek pliku CMT w formacie xlsx, który jest możliwy dzięki użyciu biblioteki Openpyxl
2. Zdefiniowanie schematu kategoryzacji poszczególnych atrybutów na poziomie pojedynczego SQL Part. Opisane działania są możliwe przy wykorzystaniu biblioteki Networkx. Na tym etapie następuję podział głównych elementów kodu stanowiących warunki selekcji, projekcji, transformacji, sortowania i grupowania. Na podstawie elementów i połączeń zidentyfikowanych w kwerendzie SQL do grafu dodawane są wierzchołki i krawędzie opisane atrybutem ‘sql’. Dodatkowo zachowywana jest kolejność tabel w klauzuli From poprzez zapis liczbowego indeksu w atrybucie ‘From\_id’ odpowiednich krawędzi.

Indeks oznaczeń w atrybucie 'sql' w wierzchołkach grafu:

* *CAR* - wynikowe tabele CAR, w liczbie ponad 30,
* *CAR\_part* - części wynikowych tabel CAR, zakładki w CMT, w tym zakładki techniczne SUB oraz archiwa,
* *Source* - tabela będąca źrodłem zasilania CAR w Korporacyjnej Hurtowni Danych
* *Field* - pole w tabeli

Indeks oznaczeń w atrybucie 'sql' w krawędziach grafu. Podsumowuje działanie obiektu na obiekt:

* *Merge* - połączenie *CAR\_part* do *CAR* (finalna operacja w procesie zasilania CAR, w ramach której wyniki wyliczania poszczególnych zakładek są łączone po wspólnym kluczu do jednej tabeli materializowanej w bazie danych),
* *From* - połączenie miedzy tabelami źródłowymi a tabelą wynikową *CAR\_part*
* *In* - połączenie wskazujące, że pole znajduje się w tabeli
* *Transform* - połączenie miedzy polami źródłowymi a wynikowymi
* *OutSelCond* – relacja od pola do zakładki w CMT: pola służące do ograniczania zawartości tabeli przed złączeniem oraz występujące w połączeniach pomiędzy tabelami,
* *SelCond* – jw., pole znajduje się w warunku where (ograniczające wyniki zapytania po złączeniu),
* *Groupby , Having, Qualify* - jw. – pole znajduje się w odpowiedniej klauzuli zapytania SQL.

Schemat kategoryzacji polega na stworzeniu grafu złożonego z wierzchołków i krawędzi, a następnie przyporządkowaniu poszczególnym atrybutom odpowiadających im wartości. Poniżej zaprezentowane zostały dwa przykłady. Pierwszy z nich ma charakter sztuczny I przedstawia samą koncepcję analizy zapytania SELECT, drugi jest przykładem wziętym z rzeczywistej implementacji CAR oraz z finalnej postaci opracowanego w projekcie narzędzia.

**Select**

T1.A,

T2.B,

Sum(t2.c)

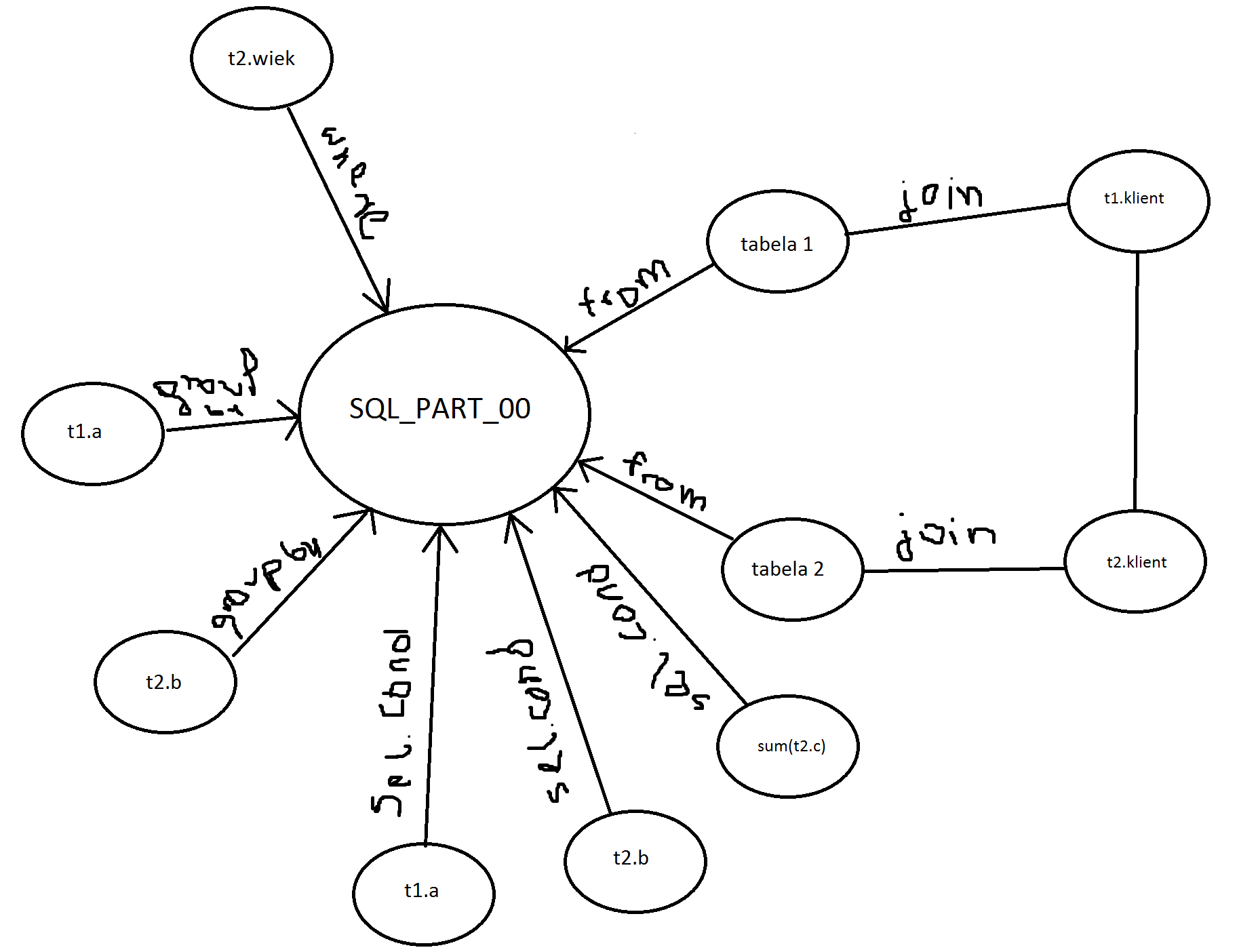
**From** tabela\_1 t1

**Join** tabela\_2 t2 **on** t2.klient = t1.klient

**Where** t2.wiek > 20

**Group** **by** t1.A, t2.B

Rysunek 1 przedstawia szablon stworzonego grafu.



Idea wydzielenia poszczególnych komponentów kodu SQL polega na separacji jego głównych składowych i umieszczeniu ich na oddzielnych wierzchołkach. Krawędzie zaś stanowią opisy operacji wykonywanych na poszczególnych atrybutach i tabelach. Dzięki takiemu zabiegowi możliwe jest natychmiastowe poznanie struktury całego kodu SQL będącego pojedynczym SQL Part. Zgodnie z tym, co widać na rysunku 1 ujęcie graficzne kodu SQL reprezentowane jest przez graf częściowo skierowany, którego główny wierzchołek to nazwa SQL Part poddanego analizie. Na rysunku przyjmuje on nazwę SQL\_PART\_00. Wszelkie warunki selekcji tj. "where", "in", "group by" "order by", itp. stanowią krawędzie skierowane do wskazanego SQL Part. Ich nazwy są ściśle uzależnione od tego jaką klauzulę reprezentują, a nazwy wierzchołków to pola, których dotyczy dana klauzula. Kolejną grupą jest rodzina krawędzi o nazwie "sel. cond.", które są ściśle powiązane z warunkami projekcji. Jako nazwę wierzchołka przyjęto alias tabeli, z której pochodzą oraz jego nazwę. Nieskierowane pozostają natomiast wierzchołki określające warunki łączenia o nazwie "join", przy czym w ostatecznej implementacji oprogramowania zdecydowano się na użycie także tutaj połączenia skierowanego. W tym przypadku zastosowano analogiczne rozwiązanie, ponieważ nazwa wierzchołka określa pole, użyte w klauzuli łączenia. Wartym uwagi jest także fakt, iż podczas graficznego prezentowania warunków selekcji w klauzuli "where" nie jest ujęty szczegółowy warunek ograniczający dany atrybut lub grupę atrybutów. Nie jest to niezbędne na potrzeby wdrażanego rozwiązania, ze względu na to, ,że w tym przypadku najistotniejsza jest struktura samego kodu, która jest widoczna po zastosowaniu powyższych zabiegów.

Zgodnie z tym, co zostało opisane w rozdziale pierwszym do wyliczania charakterystyk CAR wykorzystywane są warunki, atrybuty oraz tabelę, które mogą występować jednocześnie na w wielu SQL Part. W związku z tym krawędzie grafu mogą wychodzić w skali jeden do wielu do innych wierzchołków. Utworzenie metabazy grafowej pozwoli na natychmiastową identyfikację takich przypadków, ponieważ ilość krawędzi wychodząca od danego wierzchołka jest ściśle powiązana z częstotliwością jego wykorzystania w CAR.

Przykład rozbicia pojedynczego SQL Part dla CMT\_CIF\_PROD\_7 oraz jego postać w formie grafu zostały zaprezentowane poniżej.

**SELECT**

TABLE\_MASTER.CIF

,**case**

**when** TABLE\_MASTER.LST\_PREL\_DT **is** **null** **and** TABLE\_1.LST\_PREL\_DT **is** **null**

**then** -1

**when** TABLE\_MASTER.LST\_PREL\_DT **is** **not** **null** **AND** TABLE\_1.LST\_PREL\_DT **is** **null**

**then** **null**

**when** TABLE\_MASTER.LST\_PREL\_DT **is** **null** **and** TABLE\_1.LST\_PREL\_DT **is** **not** **null**

**then** **null**

**when** TABLE\_MASTER.LST\_PREL\_DT **is** **not** **null** **and** TABLE\_1.LST\_PREL\_DT **is** **not** **null**

**then** **cast**(((TABLE\_MASTER.LST\_PREL\_DT - TABLE\_1.LST\_PREL\_DT) month(4)) **as** numeric(4,0))

**end** **as** RENEWAL\_MTH\_CNT

**FROM** ${CAR\_TEMP}.CMT\_CIF\_PROD\_6 **as** TABLE\_MASTER

**LEFT** **JOIN** ${CAR\_TEMP}.CMT\_CIF\_PROD\_6\_AM **as** TABLE\_1

**ON** TABLE\_MASTER.CIF = TABLE\_1.CIF

**AND** TABLE\_1.PERIOD\_DT = (**select** max(period\_dt) **from** ${CAR\_TEMP}.CMT\_CIF\_PROD\_6\_AM )



1. Do przeanalizowania wszystkich SQL Part została wykorzystana pętla, której ilość obrotów jest uzależniona od ilości kwerend biorących udział przy wyliczaniu charakterystyk. Schemat eksploracji podczas każdego obrotu jest analogiczny do tego jaki został zaprezentowany w poprzednim punkcie, a kolejne wyniki zapisywane są do dedykowanej tabeli powodując jej rozszerzenie. Wyniki na tym etapie są przechowywane w pamięci operacyjnej komputera.
2. Ostatnim etapem jest wyeksportowanie uzyskanych wyników do pliku płaskiego. W ramach projektu wykorzystany został format graphml, który odzwierciedla strukturę grafową w xml.

## 3.2 Warstwa druga

Działania w drugiej warstwie polegają na wykorzystaniu pliku wsadowego poprzez przekazanie go do narzędzia, którego funkcje przewidują tworzenie złożonych analiz graficznych. Na jego podstawie możliwe będzie utworzenie metabazy grafowej pokazującej graficzną strukturę przepływu głównych elementów pomiędzy poszczególnymi kwerendami. Narzędziem użytym w tej warstwie jest oprogramowanie GEPHI, którego konfiguracja została bliżej opisana w rozdziale czwartym. Schemat działań w warstwie drugiej przebiega zgodnie z poniższym schematem.

1. Import utworzonego w warstwie pierwszej pliku wsadowego.
2. Wybór obiektu, który będzie przedmiotem analizy (tj. tabeli, zakładki lub pola w CAR) oraz kierunku analizy (analiza zakładki, poprzedników, następników lub wszystkich powiązań).

Zapis wygenerowanego w ten sposób podgrafu w pliku graphml umożliwiającym import i prezentację w Gephi.

## 3.3 Warstwa trzecia

Działania w trzeciej warstwie mają bezpośrednie powiązanie z szeroko pojętym rozwojem środowiska CAR, które jest w tym przypadku rozumiane jako dodawanie nowych charakterystyk. Dotychczasowe działania tego typu ograniczały się jedynie do dokładania nowych SQL Part oraz rozszerzania plików CMT o nowe zakładki. Unikatowość charakterystyk była kluczowa, natomiast mniejszą uwagę poświęcano kwestiom optymalizacyjnym. W rezultacie w całym obrębie środowiska CAR występuję duża ilość wielokrotnie użytych bloków kodów, które przekładają się na spadek wydajności zauważalny w czasie przetwarzania. Mechanizm narzędzia stworzony w trzeciej warstwie ma celu wyeliminowanie dodawania do CAR kwerend o podobnej strukturze, co finalnie będzie skutkowało optymalnym rozszerzaniem środowiska o nowe charakterystyki. Ponadto w nowych kodach identyfikowane będą sekwencje, warunki, transformacje, tabele oraz atrybuty, które wykorzystano wcześniej w bardzo podobny lub identyczny sposób. Dzięki temu dodawanie charakterystyk będzie możliwe na poziomie wykorzystywanego SQL Part, który zostanie zaktualizowany o dany blok kodu, a nowe kwerendy będą zawierały unikatowe bloki kodów.

W wersji oprogramowania przygotowanej na potrzeby projektu zilustrowano możliwość automatycznego podpowiadania pięciu najbardziej pasujących w CAR zakładek, do których można dodać nowe charakterystyki opisane nowym SQL. Prace w warstwie pierwszej były przeprowadzane przy użyciu języka python w wersji 3.5. Schemat działań w warstwie trzeciej przebiega zgodnie z poniższym schematem.

1. Przyjęcie jako parametr trzech ciągów opisujących nową kwerendę SQL do zaimplementowania w CAR:
   1. tablic w klauzuli From (gdzie kolejność ma istotne znaczenie ze względu na posługiwanie się wszędzie w CAR metodą *Left join*),
   2. pól w klauzuli Where,
   3. pól w klauzuli Group by.
2. Import utworzonego w warstwie pierwszej pliku wsadowego ze strukturą grafową CAR,
3. Wyliczenie w pętli dla każdej zakładki w CAR poniżej zdefiniowanego rankingu wg prostego, zbudowanego ekspercko klasyfikatora regułowego,
4. Prezentacja pięciu najlepiej dopasowanych zakładek w CAR wraz z wartością rankingu.

Przebieg wyznaczania rankingu opisany jest funkcją score\_calc, której kod zaprezentowany jest poniżej. Należy zaznaczyć, że treść tej funkcji jest niezależnie modyfikowalnym, wyodrębnionym elementem całej aplikacji, a sam algorytm ma charakter jedynie przykładowy i w kolejnych wersjach aplikacji wymaga dostosowania do wymagań funkcjonalnych właściciela systemu.

**def score\_calc**(l\_from, sg\_from,l\_group\_by, sg\_gb,l\_where, sg\_where): skoringu

**if** from\_not\_fit(l\_from, sg\_from) == True:

**return** -1

score = score\_from(l\_from, sg\_from)

**if** score == -1:

**return** score

score += score\_gb(l\_group\_by, sg\_gb)

score += score\_where(l\_where, sg\_where)

**return** score

W pierwszym kroku uruchamiana jest funkcja from\_not\_fit do sprawdzenia, czy którakolwiek tablica From ze starej zakładki pokrywa się z tablicą z nowego zapytania. Jeśli żadna się nie pokrywa, wyliczanie funkcji score\_calc jest kończone z wartością -1.

Drugi krok procedury wyliczania rankingu polega na wyliczeniu wartości funkcji *score\_from*. Funkcja ta odpowiada na pytanie, czy zbiór pasujących tabel jest w tej samej kolejności. Jeśli tak nie jest, funkcja zwraca -1, natomiast w przeciwnym wypadku zwracana jest liczba pasujących do siebie tabel.

Trzecim krokiem jest wyliczenie funkcji score\_gb, czyli weryfikacja, czy obiekty w klauzuli Group\_by obu zapytań pokrywają się ze sobą. Funkcja zwraca 0 lub 1 punkt w zależności od tego, czy listy obiektów są puste, różnej długości lub na podstawie szczegółowego porównania elementów obu list.

Czwartym krokiem jest weryfikacja funkcji *score\_where*, gdzie do wyniku dodawany jest jeden punkt pod warunkiem, że elementy klauzuli where nowej kwerendy pojawiają się wśród warunków zapytania w danym CAR\_part.

# 4. Wykorzystanie narzędzi/komponentów

W ramach budowy opisywanego mechanizmu użyto dwóch głównych komponentów, których synergia pozwoliła na uzyskanie narzędzia w ostatecznej formie. Opisywane w poprzednim rozdziale warstwy koncepcji rozwiązania zostały wyraźnie podzielone ze względu na funkcjonalność, ale także na narzędzia jakie użyto podczas prac nad każdą z nich. Poszczególne warstwy charakteryzują się brakiem przenikania narzędzi na poziomie każdej z nich. Zastosowane narzędzia musiały gwarantować możliwość użycia poniższych technik:

* Techniki programistyczne,
* Techniki wizualizacyjne

Jako narzędzie pozwalające na zastosowanie technik programistycznych użyto języka Python w wersji 3.5 dostarczanego w pakiecie typu open source o nazwie Anaconda Navigator. Edytorem modułów i skryptów powstałych w projekcie jest program Spyder 3.1.4. Jest to graficzny interfejs użytkownika (GUI) wchodzący w skład głównego pakietu Anaconda umożliwiającego proste uruchamianie wbudowanych aplikacji i łatwe zarządzanie pakietami oraz środowiskiem bez potrzeby używania linii poleceń/wiersza poleceń. Stanowi fundament prac w pierwszej i trzeciej warstwie dotyczącej koncepcji rozwiązania prezentowanej w rozdziale trzecim. Zalety zastosowania języka Python dotyczyły możliwości korzystania z szeregu bibliotek, które poprzez swoją różnorodność dostarczały niezbędnych funkcjonalności. Dokładniejszy opis użytych bibliotek został przedstawiony w rozdziale trzecim. Dzięki zaimplementowanym w bibliotekach gotowym funkcjom możliwe było stworzenie kodu spełniające kryteria koncepcji w wyżej wymienionych warstwach.

Na potrzeby projektu zbudowana została aplikacja automatyzująca wywoływanie poszczególnych skryptów w języku Python i ułatwiająca wprowadzanie parametrów do wywołań poszczególnych funkcji. Makieta aplikacji została przygotowana z wykorzystaniem QT Designer 5.6.0. w pakiecie Anaconda, natomiast kod służący do obsługi zdarzeń został napisany w Spyder po zaimportowaniu do niego dodatkowej biblioteki PyQt5.

Zastosowanie technik wizualizacyjnych miało bezpośrednie przełożenie na działania podjęte w drugiej warstwie dotyczącej koncepcji rozwiązania prezentowanej w rozdziale trzecim. Narzędziem użytym na tym etapie jest program Gephi 0.9.1, który znajduje się w grupie narzędzi typu open-source służącym do badania powiązań pomiędzy obiektami oraz ich wizualizacji. Wskazane funkcjonalności były kluczowe w kontekście wyboru narzędzia. Program został napisany w języku JAVA na platformie NeTBeans. Jego użycie oraz zastosowanie jest kluczowe w kontekście budowy narzędzia, ponieważ sprowadza się do stworzenia interaktywnego grafu spajającego wszystkie SQL Part, co jest jedną z głównych funkcjonalności dostarczanych przez opisywane narzędzie. Ponadto jest to jedno z niewielu darmowych narzędzi pozwalających na tworzenie zaawansowanych analiz graficznych, które w czytelny sposób pozwoliło zwizualizować rozbudowaną strukturę środowiska CAR.

## 4.1 Budowa środowiska

Aby zbudować kompletne środowisko umożliwiające wdrożenie przedstawionej w rozdziale trzecim koncepcji należy zapewnić oprogramowanie obsługujące dwie kluczowe warstwy, tj:

1. Warstwa programistyczna – polega na przeprowadzeniu niezbędnych procedur związanych z kompletną instalacją pakietu Anaconda Navigator. Kolejnym etapem jest walidacja aplikacji Spyder i weryfikacja możliwości korzystania z niezbędnych bibliotek i instalacji w razie ich braku w pakiecie instalacyjnym zapewnianym przez dostawcę oprogramowania. Brakujące biblioteki można doinstalować w razie potrzeby korzystając z opcji Environments w programie Anaconda, wpisując nazwę niezbędnej biblioteki i klikając Install.

2. Warstwa wizualizacji – polega na poprawnej instalacji oprogramowania Gephi. Wszelkie procedury konfiguracyjne mogą zostać pominięte, ponieważ po poprawnym przejściu procesu instalacji opisanym w rozdziale 4.2.2. narzędzie jest kompletne i nie wymaga podejmowania żadnych dodatkowych akcji mających na celu konfigurację.

Budowa opisywanego środowiska dotyczy systemu Windows. Inne systemy operacyjne mogą charakteryzować się koniecznością podjęcia odmiennych działań zarówno na etapie instalacji, jak i konfiguracji.

### 4.1.1 Instalacja Anaconda Navigator w systemie Windows

W celu uzyskania pełnej wydajności na poziomie komponentów wchodzących w skład oprogramowania Anaconda Navigator najlepszym rozwiązaniem jest instalacja z poziomu użytkownika lokalnego, co stanowi pełną rekomendację dostawcy oprogramowania dla wszystkich trzech systemów, czyli:

* MS Windows
* MacOS
* Linux

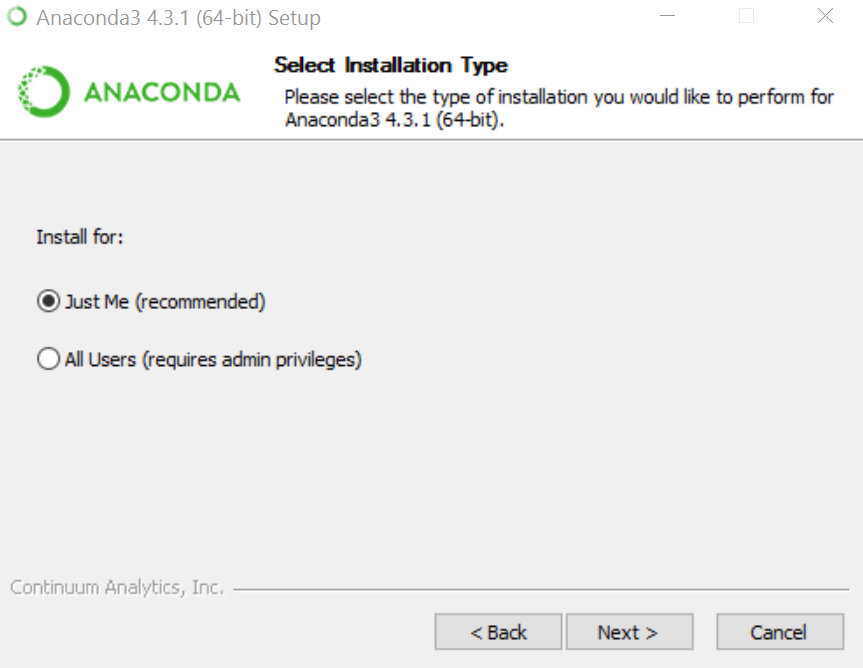
Możliwe są dwa typy instalacji oprogramowania. Pierwszą z nich jest instalacja standardowa, natomiast druga polega na instalacji w tzw. trybie cichym "silent mode" lub "silent installation". Kompletną procedurę instalacji standardowej można przedstawić w 6 krokach:

1. Należy pobrać paczkę z plikami instalacyjnymi ze strony dostawcy.

2. Po kliknięciu na ikonkę "Setup.exe" uruchomi się plik instalacyjny.

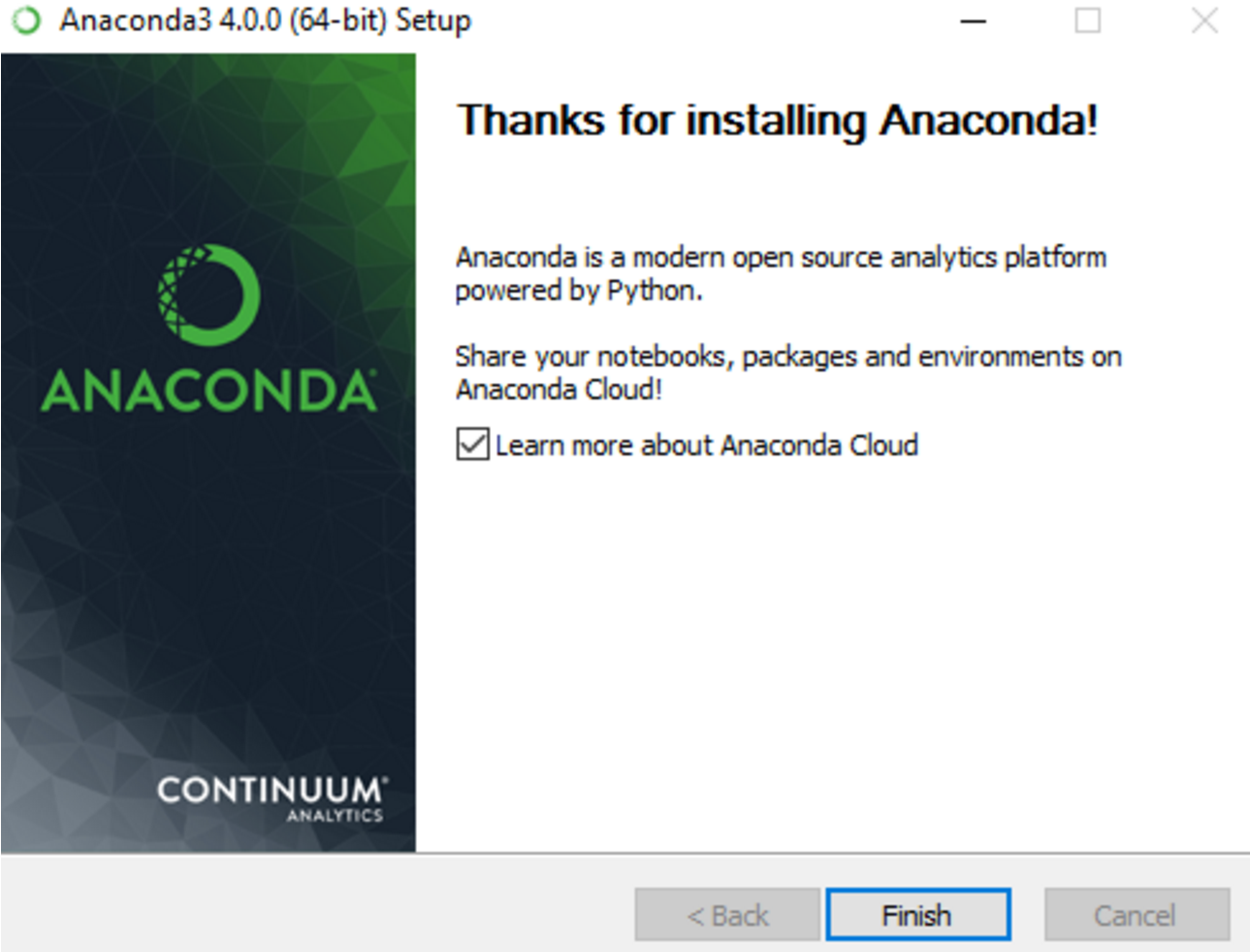
3. Należy określić lokalizację, na której ma zostać zainstalowane oprogramowanie z jednoczesnym uwzględnieniem braku spacji oraz wszelkich znaków w formacie UNICODE.

4. Instalacja powinna odbywać się z poziomu użytkownika lokalnego, a wszelkie odstępstwa są rekomendowane jedynie w momencie, w którym niezbędny jest administrator.



5. Jeśli program anty-wirusowy zwraca komunikaty o zagrożeniu należy *tymczasowo* go wyłączyć i uruchomić ponownie po przejściu procesu instalacji.

6. Po poprawnym przejściu procesu instalacji wyświetlony komunikat powinien być zgodny z rysunkiem 2.



Drugim możliwym sposobem jest instalacja w tzw. trybie cichym gdzie nie wyświetlane są żadne komunikaty lub okna w sposób, w jaki przebiegało to podczas instalacji standardowej. Ponadto wszelkie domyślne ustawienia są automatycznie akceptowane. Przebieg procesu instalacji cichej w systemie Windows przebiega w następujący sposób.

Instalacja oprogramowania Anaconda Navigator w trybie cichym może zostać uruchomiona poprzez użycie argumentu /s w wierszu poleceń. Możliwa jest również modyfikacja części domyślnie ustawionych opcji.

 /InstallationType=[JustMe|AllUsers], domyślnie: JustMe

 /AddToPath=[0|1], domyślnie: 1

 /RegisterPython=[0|1], sposób instalacji intepretera Python: 0 (Just me), 1 (All users)

 /S

 /D=&lt;installation path>;

Wszystkie argumenty są wrażliwe na małe i wielkie litery. Ścieżka do instalacji pliku musi być na ostatnim miejscu i nie może być zawierać się w apostrofach lub cudzysłowach. W momencie, w którym korzysta się z instalacji w trybie cichym z wywołując argument /s wymagane jest również podanie ścieżki instalacyjnej po argumencie /D. Przy pomocy poniższej komendy możliwe jest wywołanie instalacji oprogramowania Anaconda Navigator w trybie cichym.

start /wait "" Miniconda4-latest- Windows-x86\_64.exe

/InstallationType=JustMe

/RegisterPython=0

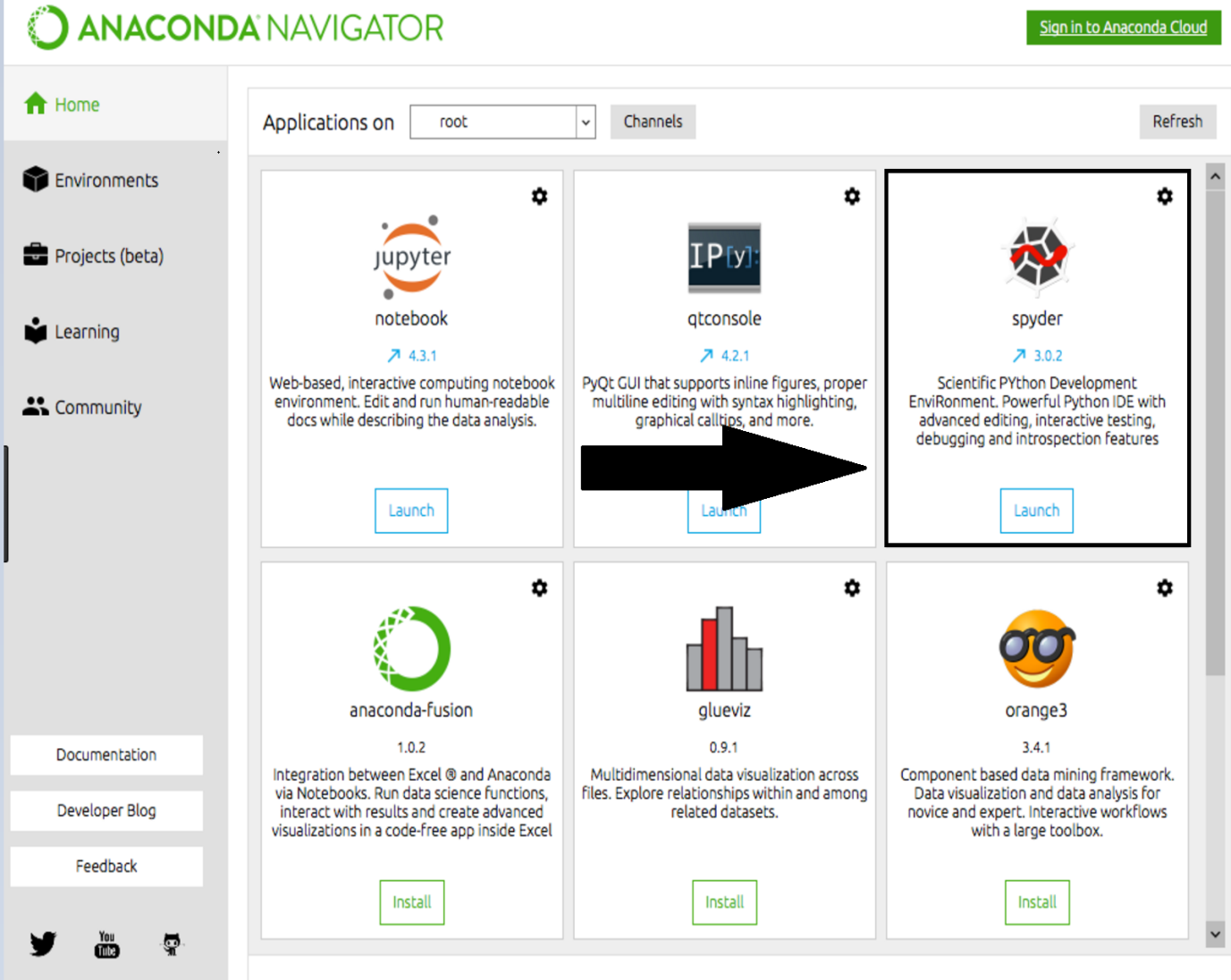
/S

/D=%UserProfile%\Miniconda3

Po poprawnej instalacji oprogramowania możliwy jest dostęp do wszystkich komponentów wchodzących w skład pakietu Anaconda Naviagtor i są to np:

* Jupyter Notebook
* Python QTCONSOLE
* Anaconda-Fusion
* Glueviz
* QT Designer

Przykład dostępnych w Anaconda Navigator komponentów widoczny jest na rysunku 3. Jednak jak już wcześniej wspomniano na potrzeby budowy narzędzia niezbędny jest interpreter języka Python, "Spyder" wraz z poszczególnymi bibliotekami opisywanymi w rozdziale trzecim.



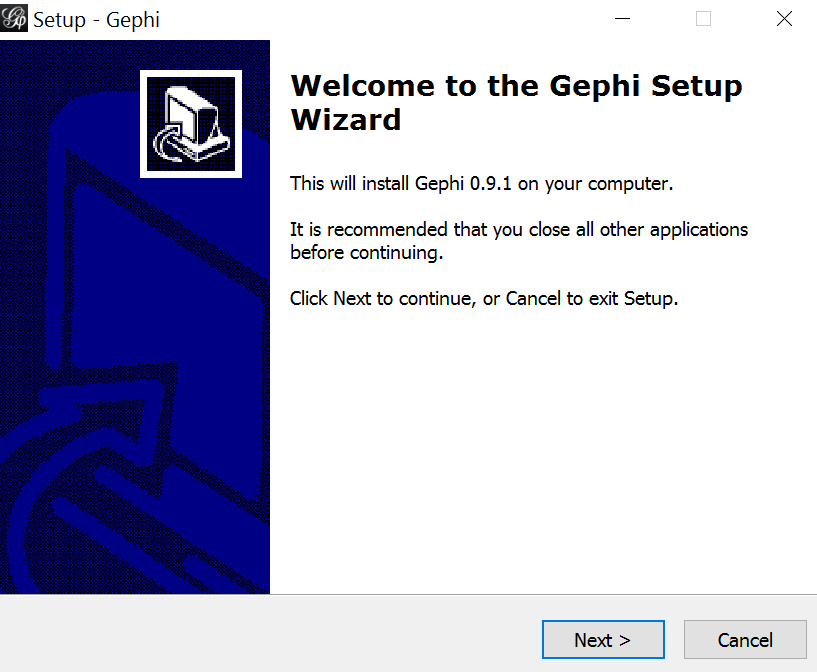
Dostawca deklaruję, iż wszystkie ww. Biblioteki są dostępne w standardowej wersji oprogramowania, jednak na etapie konfiguracji warto upewnić się czy środowisko do pracy w warstwie pierwszej jest kompletne. Przekrojowy proces konfiguracji w wypadku braku wymaganych bibliotek został opisany w podrozdziale 4.3.

Po zainstalowaniu pakietu należy przeprowadzić aktualizację poszczególnych komponentów pakietu wybierając opcję Update Application w prawym górnym narożniku karty komponentu.

4.2.2 Instalacja Gephi 0.9.1 w systemie Windows

Instalacja Gephi w systemie Windows przebiega zgodnie z poniższą procedurą.

* Pierwszym krokiem jest weryfikacja kompatybilności wersji JAVA JRE z programem Gephi. Oprogramowanie współpracuje z JAVA JRE w wersji 7 oraz 8. Jeśli okażę się, że zainstalowana wersja odbiega od standardów wskazanych przez producenta należy dokonać jej aktualizacji. Aby sprawdzić zainstalowaną wersję JAVA JRE w systemie Windows należy postępować zgodnie z poniższym schematem:
  1. Kliknąć prawym przyciskiem myszy w dolnym lewym rogu ekranu, po czym wybrać z menu podręcznego pozycję **Panel sterowania**.
  2. W wyświetlonym Panelu sterowania wybrać kategorię **Programy**.
  3. Wybrać opcję **Programy i funkcje**.
  4. Na liście są uwzględniane zainstalowane wersje oprogramowania Java.
* Kolejnym krokiem jest pobranie paczki z plikami instalacyjnymi Gephi ze strony producenta.
* Następnie należy uruchomić kreator instalacji "Setup.exe", który będzie skutkował rozpoczęciem procesu instalacji.



* Instalacja oprogramowania przebiega w sposób standardowy i polega na przejściu przez poszczególne etapy tj:
  1. Zaakceptowanie warunków umowy określonych przez producenta
  2. Wskazanie docelowej ścieżki instalacyjnej
  3. Finalizacja procesu instalacji

## 4.3 Konfiguracja narzędzi

Ostatnim etapem podczas budowy kompletnego środowiska jest konfiguracja poszczególnych komponentów. Idea tego typu działań sprowadza się do maksymalizacji funkcjonalności oferowanej przez dane narzędzie jak również do weryfikacji, dostępności wszystkich niezbędnych elementów. Zastosowanie procesu konfiguracji odbywa się w przełożeniu na każde z opisanych w rozdziale 4.1 narzędzi.

### 4.3.1 Konfiguracja komponentu "Spyder"

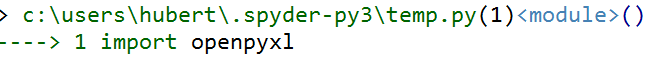
Aby mieć pewność, że komponent "Spyder" został prawidłowo skonfigurowany należy przeprowadzić działania weryfikacyjne mające na celu sprawdzenie możliwości korzystania z bibliotek, które są niezbędne do pracy w ramach budowy narzędzia w pierwszej warstwie oprogramowania. Sprawdzenie zainstalowanych wraz z pakietem Anaconda Navigator bibliotek Python może zostać przeprowadzone poprzez użycie:

* Oprogramowania Anaconda Navigator
* Z poziomu wiersza poleceń systemu Windows

Przeprowadzenie weryfikacji przy użyciu oprogramowania Anaconda Navigator przebiega zgodnie z poniższym schematem, w którym należy

* Uruchomić zainstalowane wcześniej oprogramowanie Anaconda Navigator
* Uruchomić komponent Spyder
* Przeprowadzić walidacje zwracanych komunikatów podczas próby importu poszczególnych bibliotek zgodnie z następującą składnią:
* import openpyxl
* import pathlib
* import network
* import re
* import matplotib
* import pyqt5

Wywołanie pojedynczej komendy w przypadku istnienia biblioteki skutkuje komunikatem mówiącym o pomyślnym przebiegu procedury "import" i jest zgodne z rysunkiem 4. Przedstawiony poniżej przykład dotyczy próby zaimportowania biblioteki openpyxl.

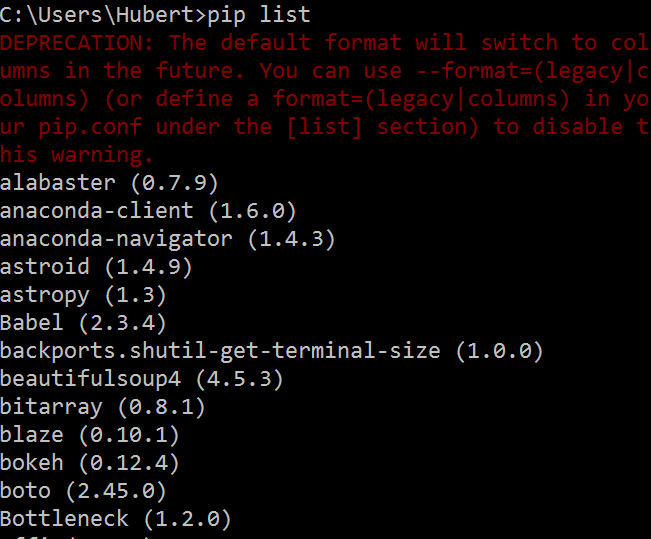


Jeśli podobny komunikat zostanie wyświetlony podczas próby importu pozostałych bibliotek, wówczas użytkownik ma pewność, że środowisko w pierwszej warstwie zostało poprawnie zainstalowane i skonfigurowane.

Komunikat informujący o tym, że dana biblioteka nie znajduję się w zainstalowanym pakiecie jest zgodny z rysunkiem 5.



Drugim sposobem na sprawdzenie dostępności wszystkich niezbędnych bibliotek w systemie Windows jest użycie linii komend (wiersza poleceń). Procedura weryfikacyjna jest zgodna rysunkiem 6 i polega na wylistowaniu zainstalowanych modułów. Aby uzyskać zamierzony efekt w wierszu poleceń należy wpisać komendę *pip list.*



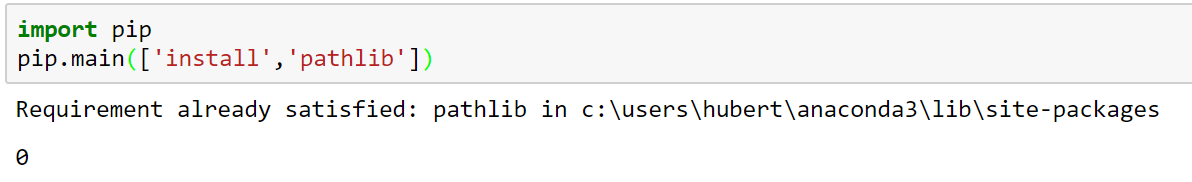
Brak na liście jakiejkolwiek biblioteki wskazanej w rozdziale drugim jest jednoznaczny z tym, że środowisko wymaga dodatkowej konfiguracji.

#### 4.3.1.1 Instalacja bibliotek Python

Jeżeli wynik uzyskany na etapie walidacji dostępności bibliotek wykazał jakiekolwiek braki niezbędne jest podjęcie kroków związanych z zapewnieniem komplementarności środowiska w warstwie pierwszej. Polega ona na instalacji brakujących komponentów. Schemat instalacji bibliotek może przebiegać na kilka sposobów, jednak na potrzeby dokumentu zostaną przedstawione dwa najbardziej popularne.

Pierwszym sposobem jest instalacja biblioteki bezpośrednio przez oprogramowanie Anaconda Navigator. Instalacja przebiega zgodnie z poniższym schematem:

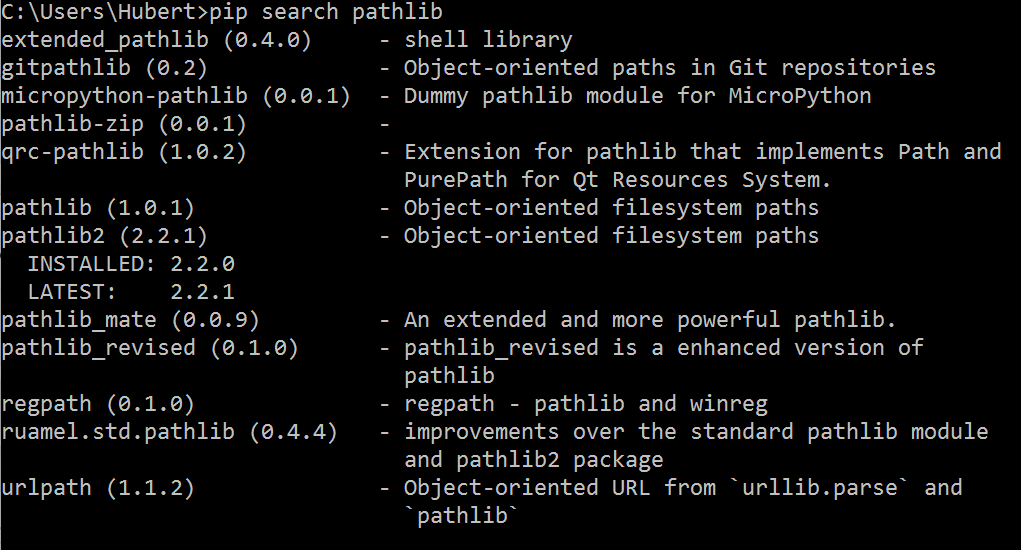
1. Należy uruchomić oprogramowanie Anaconda Navigator
2. Należy uruchomić komponent Spyder. (alternatywnym rozwiązaniem jest uruchomienie komponentu Jupyter Notebook)
3. Należy wprowadzić blok kodu pozwalający na przeprowadzenie instalacji brakującej biblioteki. Rysunek 7 przedstawia blok kodu, którego użycie przekłada się na zainstalowanie biblioteki pathlib.



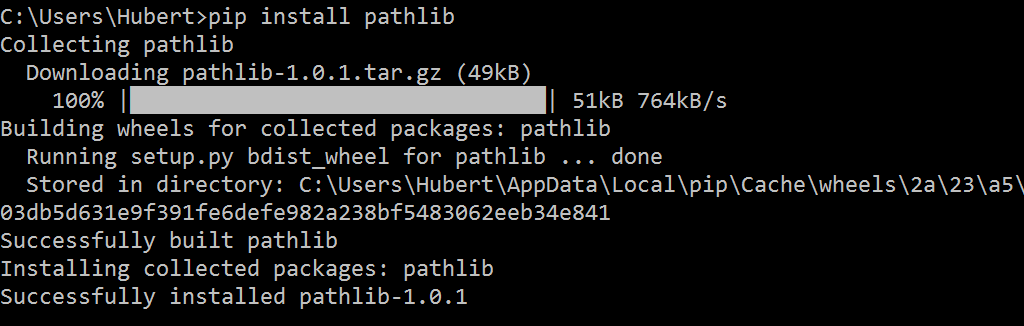
1. Procedurę opisaną w punkcie trzecim należy przeprowadzić dla każdej brakującej biblioteki.

Drugim sposobem jest instalacja bibliotek przy użyciu linii komend (wiersza poleceń). Instalacja przebiega zgodnie z poniższym schematem.

1. Należy uruchomić linię komend (wiersz poleceń) systemu Windows
2. Opcjonalnie można sprawdzić listę dostępnych do zainstalowania bibliotek. Jest to możliwe poprzez wywołanie komendy *pip search <nazwa\_biblioteki>.* Wynik zapytania dla biblioteki pathlib został zaprezentowany na rysunku 8 i zawiera wszystkie biblioteki, które spełniają zadane kryteria wyszukiwania.



1. Należy wywołać komendę pozwalającą na instalacje założonej biblioteki. Składnia odpowiadająca za tego typu operacje to: *pip install <nazwa\_biblioteki>.* Przykład dotyczący instalacji biblioteki pathlib został przedstawiony na rysunku 9. Po wywołaniu komendy następuję zainicjowanie procesu instalacji, który polega na pobraniu wskazanej biblioteki oraz umieszczeniu jej w dedykowanym katalogu. Finalnym komunikatem mówiącym o pomyślnym przebiegu instalacji jest informacja widoczna w ostatniej linijce rysunku 9, a jej treść brzmi: Successfull*y installed <nazwa\_biblioteki> - <wersja\_biblioteki>.*

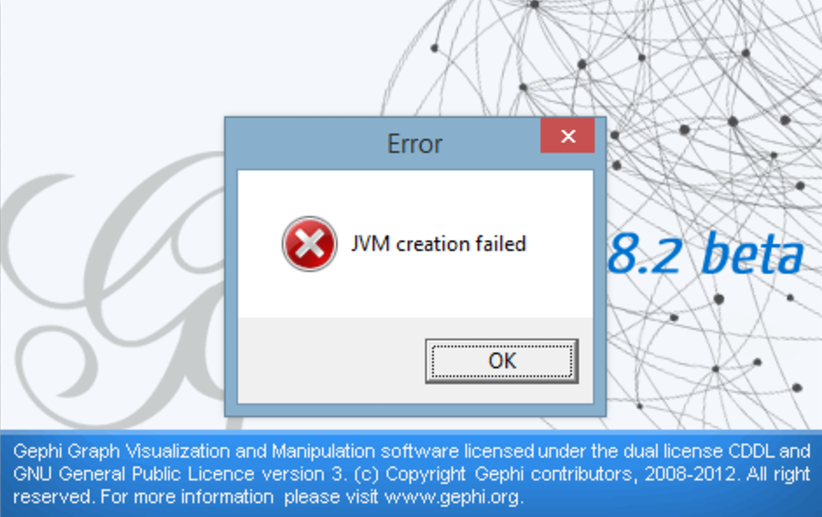


Po tak przeprowadzonych działaniach użytkownik ma pewność, iż pierwsza warstwa środowiska została poprawnie skonfigurowana.

### 4.3.2 Konfiguracja oprogramowania Gephi

Proces konfiguracji oprogramowania Gephi jest konieczny jedynie w momencie, w którym podczas uruchamiania lub pracy zostaje zwracany komunikat o treści "JVM creation failed". Przyczyną pojawiającego się błędu są ustawienia pamięci determinowane przez pakiet JAVA JRE. Maksymalna pamięć wynikająca z domyślnych ustawień to 512 mo. Powstały błąd może wynikać z dwóch powodów.

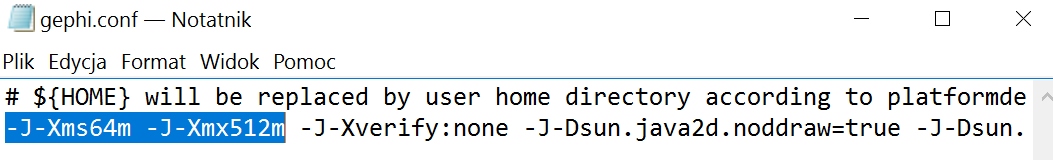
1. Pierwszym z nich jest przypadek, w którym alokowana jest zbyt mała ilość pamięci. Skutkuje to zatrzymaniem pracy narzędzia w momencie osiągnięcia limitu, co wiąże się z utratą wyników sesji.
2. Drugi przypadek ma miejsce w momencie, w którym alokowana ilość pamięci wynikająca z ustawień jest większą niż ilość pamięci w jaką wyposażony jest system. Wówczas pakiet JAVA IRE nie będzie w stanie wykonać startu, a komunikat zostanie zwrócony już podczas próby uruchomienia programu.



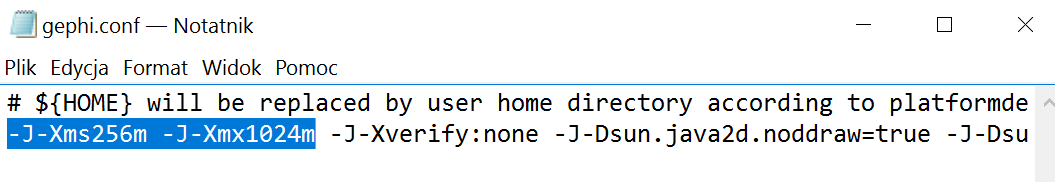
Ilość alokowanej pamięci może zostać skonfigurowana poprzez edycję pliku z ustawieniami narzędzia o nazwie gephi.conf. Procedura konfiguracji wygląda następująco:

Należy uruchomić plik gephi.conf. w dowolnym edytorze tekstowym. Można go odnaleźć zgodnie ze ścieżką menu Start -->GEPHI --> gephi.conf.

* Należy zmodyfikować wartość występującą po opcji **–Xms oraz -Xmx**, które są wartościami granicznymi w przestrzeni wykorzystywanej pamięci.



* Aby użyć przestrzeni pamięci w zakresie 256-1024 Mb należy ustawić opcję -Xms256m oraz Xmx1024m. Najbardziej optymalnym rozwiązaniem dla komputerów z pamięcią 2 Gb jest ustawienie opcji –Xmx1400. Po dokonanych modyfikacjach należy zapisać plik gephi.config.



* W przypadku kiedy zmiana ustawień w pliku gephi.config miała miejsce podczas otwartej sesji narzędzia, wówczas należy je zamknąć oraz uruchomić ponownie.

Wszystkie działania opisane w rozdziale czwartym przekładają się na budowę w pełni komplementarnego oraz skonfigurowanego środowiska, które jest przystosowane do wdrożenia założonej koncepcji.

### 4.3.3 Konfiguracja oprogramowania CMT\_Widget

Wszystkie działania opisane w rozdziale czwartym przekładają się na budowę w pełni komplementarnego oraz skonfigurowanego środowiska, w którym można uruchomić przygotowane oprogramowanie. Składa się ono z plików:

* CMT\_parser\_Graph\_constructor.py
* Graph\_analysis.py
* CAR\_table\_ranking.py
* project\_testing.py
* parser\_app\_v2.ui
* CMT\_Widget.py

Pliki te należy wgrać do ustalonego katalogu roboczego, np. *„d:\projekty\zajęcia na uczelni\!tpd\”*, do podfolderu o nazwie np. *„py”*. Dodatkowo, oprócz podfolderu zawierającego modułu należy utworzyć podfolder *„logs”* do którego program parsujący wgrywa logi z przetwarzania poszczególnych zakładek CMT. Z kolei w podfolderze o nazwie „CMT” lub podobnej, parametryzowalnej z poziomu CMT\_Widgetu, należy umieścić źródłowe pliki CMT. Poniżej podany jest szczegółowy opis poszczególnych modułów oprogramowania:

CMT\_parser\_Graph\_constructor.py jest oprogramowaniem realizującym pierwszą warstwę analizy opisanej w rozdziale 3 niniejszego dokumentu. Wynikiem działanie jest zapisanie w katalogu roboczym bazy CAR w formie xml, np. pliku car.graphml.

Graph\_analysis.py realizuje drugą warstwę analizy. W wyniku działania tego modułu w katalogu roboczym zapisywane są pliki xml o nazwach w formacie *nazwa\_kierunek.graphml,* gdzie nazwa jest nazwą obiektu, z którą wywołana jest funkcja analizy, zaś kierunek opisuje jeden z czterech możliwych, wymienionych wyżej, przekrojów analizy grafu.

CAR\_table\_ranking.py realizuje trzecią warstwę analizy, w ramach której powstaje pięć najbliższych dopasowani zakładek CAR do obiektów nowego zapytania SQL.

project\_testing.py to moduł roboczy służący testom jednostkowym budowanego oprogramowania. Ma jedynie charakter pomocniczy i będzie służyć wsparciu w przyszłym rozwoju aplikacji.

parser\_app\_v2.ui to plik zawierający definicję zbudowanej w QT Designer makiety aplikacji zawierającej trzy zakładki odpowiadające trzem warstwom analizy.

CMT\_Widget.py jest głównym programem, po uruchomieniu którego pojawia się aplikacja. Widżet został zaprojektowany tak, aby prowadzić analityka przez kolejne etapy analizy i powinien być na tyle intuicyjny, aby nie wymagać szczegółowego pliku pomocy. Program uruchomić można otwierając treść modułu w aplikacji Spyder i naciskając F5. Alternatywnie, aplikację można uruchomić otwierając Wiersz Polecenia w systemie Windows, nawigując do katalogu roboczego i folderu z kodami Python poleceniem np.

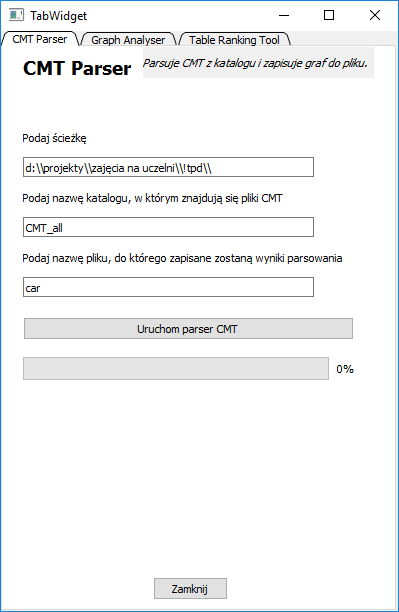
*Cd „d:\projekty\zajęcia na uczelni\!tpd\py”*

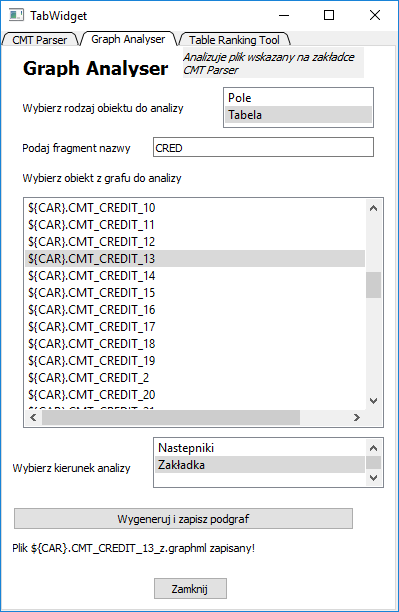
A następnie wpisując polecenie:

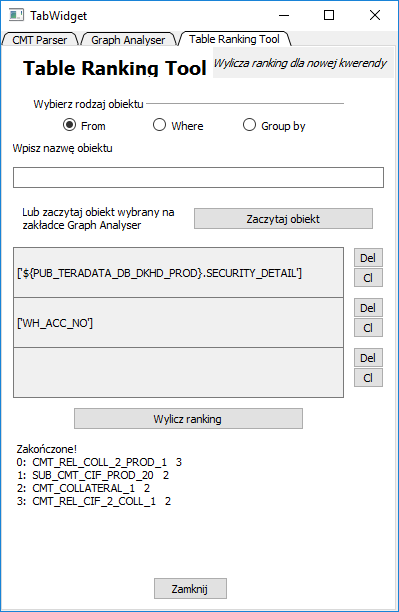
*python CMT\_Widget.py*

Poniżej zaprezentowane są zrzuty ekranu z aplikacji. Po otwarciu aplikacji pojawia się okno zatytułowane TabWidget w trzema zakładkami. Domyślnie aplikacja otwarta jest na trzeciej zakładce.

* Zakładka CMT Parser zawiera zaimplementowane funkcje modułu CMT\_parser\_Graph\_constructor.py
* Zakładka Graph Analyser zawiera front-end do analiz zakodowanych w module Graph\_analysis.py
* Zakładka Table Ranking Tool zawiera pola do wprowadzania parametrów dla funkcji z modułu CAR\_table\_ranking.py.







# Podsumowanie

Rozwiązanie zaprezentowane w niniejszym dokumencie ma na celu zilustrowanie możliwości automatycznej analizy etapu transformacji rozumianego jako litera T w tradycyjnych procesach ETL służących do zasilania hurtowni danych. Przedmiotem badań w niniejszym opracowaniu był proces zasilania wydzielonego Data Marta Korporacyjnej Hurtowni Danych BZ WBK o nazwie Repozytorium Analityczne Klienta (w skrócie CAR). Zaprezentowana tu analiza jest krokiem naprzód w stronę rozwiązania problemu milenijnego polegającego na braku oprogramowania pozwalającego na automatyczne uwzględnianie w procesie zasilającym zmian w źródłach danych lub nowych wymagań odnośnie struktur i treści danych. Wprawdzie przygotowane oprogramowanie nie automatyzuje samego procesu przygotowywania czy rozbudowy istniejących zapytań SQL, niemniej może służyć pomocą w analizie i przyspieszać ją oferując przydatne wskazówki odnośnie tych obszarów istniejącego systemu (zakładek CAR), w których takie zmiany należy wprowadzić, aby osiągnąć najwydajniej działający system.