AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ
KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

Porównanie szybkości języka zapytań SQL z inteligentnym asocjacyjnym wyszukiwaniem danych wykorzystującym asocjacyjne grafowe struktury danych AGDS

Performance comparison between SQL and intelligent associative data search using associative graph data structures AGDS.

Autor: *Kamil Makarowski*
Kierunek studiów: Automatyka i Robotyka
Opiekun pracy: *dr hab. Adrian Horzyk*

Kraków, 2017

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a tak że uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.)

„Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

……..

Podpis dyplomanta

Spis treści

Wykaz ważniejszych oznaczeń

AGDS (*ang. Associative Graph Data Structure*) – Asocjacyjna Grafowa Struktura Danych
DBMS (*ang. Database Management System)* – System Zarządzania Bazą Danych
BCNF (*ang. Boyce-Codd Normal Form)* – Postać normalna Boyce'a-Codda
SZBD – System zarządzania bazą danych
DDL – (*ang. Data Definition Language)* – Język definicji danych
DML – (*ang. Data Manipulation Language)* – Język manipulowania danymi
DQL – (*ang. Data Query Language)* – Język formułowania zapytań baz danych
DCL – (*ang. Data Control Language)* – Język kontroli danych
CRUD – (*ang Create-Read-Update-Delete)* – Podstawowe funkcje aplikacji korzystających z pamięci trwałych
SQL (*ang. Structured Query Language)* – Strukturalny język zapytań
RBO (*ang. Rules-Based Optimization)* – Optymalizator oparty na regułach
CBO (*ang. Cost-Based Optimization*) – Optymalizator oparty na kosztach
AVB Tree (*ang. Aggregated-Values B-Tree*) – Drzewo agregujące wartości
NF (*ang. Normalized Form)* Postać znormalizowana bazy danych
BAS (*ang. Biological Associative System)* – Biologiczny system skojarzeniowy
AAS (*ang. Artificial Associative System)* – Sztuczny system skojarzeniowy
AANG *(ang. Active Associative Neural Graph)* – Aktywny asocjacyjny graf neuronowy
BST (*ang. Binary Search Tree)* – Binarne drzewo poszukiwań
RBT (*ang. Red-Black Tree)* – Drzewo czerwono-czarne

1. Wstęp

1.1. Cel pracy

Celem pracy jest przeprowadzenie analizy porównawczej języka zapytań SQL pomiędzy relacyjną bazą danych, a asocjacyjnymi grafowymi strukturami danych AGDS. Zostanie wykonana implementacja asocjacyjnego modelu danych, do którego transformowane będą struktury tabelaryczne. Zakres pracy obejmuje porównanie szybkości działania operacji na tych strukturach, wraz z analizą oraz wyprowadzeniem wyników odnośnie złożoności obliczeniowej. Zaprojektowany system ma za zadanie obsługiwać większość poleceń zaimplementowanych w języku SQL ze szczególnym naciskiem na łączenie. Oprócz ogólnego porównania szybkości przeprowadzona również zostanie analiza wewnętrznej struktury oraz zalet wykorzystania struktur drzewiastych i grafowych.

1.2. Wprowadzenie

W dzisiejszych czasach Big Data daje światu nieograniczone możliwości we wszystkich branżach od opieki zdrowotnej, po finanse czy produkcje. Obecnie, każde urządzenie podłączone do Internetu generuje ogromne ilości danych, które niosą ze sobą niekiedy ważną informację. Nieustanne zmiany w technologii i Big Data powodują, że wszystkie firmy, agencje rządowe starają się przetwarzać jak najwięcej istotnych danych w jak najkrótszym czasie. We współczesnych systemach komputerowych, które są obecnie jedną z najszybciej rozwijających się branż na świecie, wykorzystywane są nowe materiały oraz powstają nowe rodzaje pamięci, aby tylko zwiększyć prędkość przetwarzania. Od dawna procesory są wielordzeniowe, przy czym w tym wypadku częściowo zrezygnowano z dokładania kolejnych rdzeni do procesora ze względu na niewspółmierne korzyści w stosunku do kosztów. Od dawna na rynku figurują już układy FPGA, które pozwalają na zrównoleglenie obliczeń. Przy połączeniu układów CPU i GPU możliwe jest wykonywanie skomplikowanych operacji w krótszym czasie dzięki przetwarzaniu głównego programu w CPU i zrównoleglenie części obliczeń w GPU [1]. Dodatkowo rozwijane jest oprogramowanie informatyczne, które pozwala efektywniej przyjmować i przetwarzać dane. Dzięki zastosowaniu modelu relacyjnego i nowoczesnych języków programowania systemy są coraz lepiej skalowalne, a dzięki skomplikowanym algorytmom optymalizacyjnym potrafią zapamiętywać i odtwarzać podobne operacje.

Niestety, nawet pomimo tak szybkiego rozwoju technologii niemożliwe jest optymalne analizowanie tak ogromnych porcji danych, których wciąż przybywa. Obecnie jedyną „maszyną”, która potrafi przetworzyć tak ogromną ilość jednocześnie

przychodzących danych jest mózg ludzki. Naukowcy od dawna prowadzą badania nad nim oraz zagadnieniami sztucznej inteligencji, która umożliwiłaby komputerom reakcje na nadchodzące bodźce, jak również automatyzację procesów, które obecnie są niemożliwe do wykonania na komputerach ze względu na zbyt wysoką złożoność obliczeniową istniejących algorytmów. Wiele zagadnień związanych z biologiczną budową mózgu jest jeszcze nieznanych, co uniemożliwia pełne odtworzenie inteligentnej sieci neuronowej [1][2].

Niniejsza praca dotyczy niewielkiej części związanej z zagadnieniem sztucznej inteligencji, jaką są sztuczne systemy skojarzeniowe. Skupiono się w niej na pasywnych asocjacyjnych strukturach grafowych, które umożliwiają szybkie przetwarzanie i przeszukiwanie skomplikowanych zbiorów danych w postaci grafu. Dokument przybliża czytelnikowi budowę oraz cechy relacyjnych baz danych, które są obecnie najpopularniejszą formą łączenia i wiązania podobnych kontekstowo zbiorów danych. Wprowadzono pojęcia asocjacyjności oraz omówiono zagadnienie sztucznych systemów skojarzeniowych, kładąc główny nacisk na wyszczególnioną w tytule strukturę AGDS [1]. Przybliżono metody optymalizacji baz danych i wykorzystywane algorytmy, jak B-Drzewa oraz AVB-Drzewa [2][7][8]. Zgodnie z celem pracy wprowadzono czytelnika w zagadnienia języka SQL [11], a następnie przy pomocy własnoręcznie zaprojektowanego mechanizmu przetwarzania zapytań porównano działanie obu struktur (AGDS i RDBMS) na większej bazie danych.

1.3. Plan pracy

Praca została podzielona na cztery części, z których każda kolejna ma swoje odniesienie do poprzedniej. W pierwszym rozdziale omówiono obecny stan wiedzy i kierunki rozwoju standardowych technologii obliczeniowych i Big Data [13] oraz przybliżono czytelnikowi zagadnienie sztucznych systemów skojarzeniowych.

Drugi rozdział przedstawia rozkład na czynniki modelu relacyjnego baz danych. W pierwszej części przybliżono temat baz danych i omówiono podstawowe jego składniki. Następnie przedstawiono podstawę systemu relacyjnego, jaką są postacie normalne i relacje. Użytkownik na rzeczywistym przykładzie dowiaduje się, w jaki sposób przebiega proces przekształcania pojedynczej skomplikowanej tabeli do postaci relacyjnej [11][12]. Na koniec przedstawiono dwa główne sposoby przetwarzania zapytań SQL – RBO i CBO oraz omówiono różnice występujące pomiędzy nimi [17][18][19].

W trzecim rozdziale omówione zostało zagadnienie sztucznych systemów skojarzeniowych, które w niniejszej pracy zostanie porównane z RDBMS. W pierwszej części rozdziału czytelnik zostaje zaznajomiony z pojęciami związanymi ze sztuczną inteligencją, rodzajami relacji pomiędzy poszczególnymi częściami grafu, a następnie wprowadzone zostanie pojęcie struktury AGDS, która jest główną częścią pracy magisterskiej. Oprócz definicji i cech AGDS zostanie przeprowadzony pełny proces przetworzenia rzeczywistej relacyjnej bazy danych. Następnie wprowadzona zostanie jedna z najpopularniejszych struktur wykorzystywana do optymalizacji baz danych, jaką są B-Drzewa [7], a następnie opisany zostanie sposób ich wstrzyknięcia

do grafu. Kolejną omówioną strukturą będą AVB drzewa, które są rozszerzeniem B-Drzew ze specjalnym zastosowaniem do omawianych w niniejszej pracy struktur grafowych. W podsumowaniu zaprezentowany zostanie główny kierunek rozwoju grafów AGDS poprzez przekształcenie ich do struktury DASNG [1][2].

Rozdział czwarty rozpocznie część praktyczną pracy, jaką jest rzeczywista implementacja i późniejsze porównanie obu struktur. Omówiona zostanie warstwa danych wraz z dostawcą bazy danych, który umożliwi zunifikowanie połączenia do jednego miejsca w aplikacji. Przedstawione zostaną decyzje projektowe oraz sposób implementacji kluczowych części systemu [9][10]. Na koniec przedstawiony został parser języka SQL, który umożliwi jednoczesne przeprowadzenie testów obu platform.

Piąty rozdział zawiera wyniki i porównania. Zawarto w nim różnice w sposobie przetwarzania i wyświetlania danych. Przeprowadzono badania oparte na zwykłych zapytaniach, jak również złączeniach. Przeprowadzono wiele testów na tej samej próbce danych, aby zaobserwować szybkość zmniejszania się czasu przetwarzania w RDBMS ze względu na mechanizm *cache'u* oraz wyeliminować ewentualne odchylenia [14].

W ostatnim rozdziale zawarte zostało podsumowanie wraz omówieniem dalszych możliwych kierunków rozwoju prac, jak również omówiono problemy, które napotkano podczas implementacji. Szczególnie został omówiony kierunek dalszych badań, który posiada nieograniczone możliwości ze względu na różnorodność algorytmów dostępu do danych.

2. Model relacyjnych baz danych

Bazą danych nazywamy zbiór uporządkowanych danych, z których każda niesie pewną informację o otaczającym świecie. Bazą danych w świecie wirtualnym jest analogią do zbioru dokumentów w formie papierowej. Niniejszy rozdział dotyczyć będzie baz danych w ujęciu informatycznym.

Definicja 2.1.

„Baza danych jest tematycznie wyodrębnionym, logicznie zintegrowanym i odpowiednio uporządkowanym oraz utrwalonym zbiorem danych.”[11]

Do zarządzania bazą danych służy specjalnie przystosowane do tego celu oprogramowanie komputerowe zwane systemem zarządzania bazą danych SZBD – *DBMS Database Management System*. Większość obecnie wykorzystywanych SZBD działa na zasadzie relacji klient-serwer. Polega ona na podziale zadań w zależności od przydzielonych ról. Dane przechowywane są na serwerze, który jest odpytywany przez klientów. Do najpopularniejszych systemów należą obecnie [12]:

- Microsoft SQL Server
- MySQL
- Oracle
- Postgre SQL

2.1. Cechy i budowa bazy danych

Każdy z przedstawionych powyżej systemów powinien realizować następujące funkcjonalności [11][15]:

- Język definiowania danych (*DDL*) – pozwala użytkownikowi wykonywanie podstawowych operacji (usuwanie, modyfikowanie, dodawanie) na strukturach danych oraz bazach danych;
- Język manipulowania danymi (*DML*) – umożliwia przeprowadzanie operacji na danych zgodnie z zasadą *CRUD* (*Create-Read-Update-Delete*). Jego głównym zastosowaniem jest manipulacja danymi;
- Język kontroli danych (*DCL*) – wykorzystywany jest do ustawiania autoryzacji dostępu do danych;
- Zapewnia spójność bazy danych, umożliwia jej odtworzenie w przypadku awarii sprzętowej, gwarantuje bezpieczeństwo danych;

- Gwarantuje jednoczesny dostęp do danych w przypadku użytkowania bazy przez wielu użytkowników poprzez mechanizm transakcji;
- Umożliwia gromadzenie dużych ilości danych;
- Udostępnia wiele interfejsów do obsługi bazy danych.

Wraz z rozwojem systemów bazodanowych i stale rosnącą liczbą przechowywanych danych rozpoczęto prace nad różnymi modelami bazodanowymi, które umożliwiałyby efektywniejszy i szybszy dostęp do danych. Do najpopularniejszych obecnie wykorzystywanych należą:

- kartotekowa baza danych,
- bazy relacyjne,
- hierarchiczny model baz danych,
- bazy obiektowe,
- nierelacyjne bazy danych (NoSQL).

Niniejsze opracowanie dotyczyć będzie w głównej mierze modelu relacyjnych baz danych, który jest obecnie najbardziej rozpowszechnionym i wykorzystywanym modelem. Podstawą relacyjnych baz danych jest teoria relacyjna. Za jej twórcę i prekursora uważa się Franka Edgara Codda, który przedstawił swoją pracę w 1970 roku [11]. Z relacyjnym modelem baz danych ściśle związana jest nomenklatura podstawowych pojęć wykorzystywanych na każdym etapie projektowania. Poniżej przedstawiono opis poszczególnych obiektów w różnych terminologiach związanych z bazami danych (*Tabela 2.1*).

Tabela 2.1. Nomenklatura obiektów bazodanowych

Teoria relacyjna	Model ER	Relacyjne bazy	Aplikacja
Relacja	Encja	Tabela	Klasa
Krotka	Instancja	Wiersz	Rekord
Atrybut	Atrybut	Kolumna	Pole
Dziedzina	Dziedzina	Typ danych	-

Poniżej szczegółowo opisano elementy modelu relacyjnego [12][15]:

- Tabela (encja, klasa obiektów) – jest to podstawowa struktura występująca w bazach danych, która służy do modelowania niezależnych od siebie obiektów. Każda tabela powinna zawierać jedynie dane, których dotyczy. W związku z tym rekordy zawarte w tabeli powinny być jednorodne, co do typu obiektu, którego dotyczą.
- Wiersz (rekord, krotka) – rekordem nazywamy obiekt, który opisany jest wszystkimi atrybutami w obrębie danej relacji. Każda tabela składa się ze zbioru wierszy, który zgodnie z teorią baz danych jest nieuporządkowany. Każdy wiersz musi być natomiast unikalny w stosunku do reszty zbioru. Ma to uzasadnienie w przypadku chęci zmodyfikowania jednego z nich.
- Kolumna (Atrybut) – Każda tabela opisana jest przez zbiór kolumn ściśle opisujących cechy obiektów należących do zbioru. Każdy z atrybutów określony jest przez typ danych, który musi być spełniony przy wypełnianiu

wiersza danymi. Nazwa kolumny musi być unikatowa w skali tabeli, aby silnik bazodanowy mógł jednoznacznie odwołać się do określonego atrybutu. Kolejność kolumn w tabeli jest bez znaczenia i to od użytkownika zależy ich uporządkowanie (w większości systemów domyślną kolejnością jest czas dodania kolumny do tabeli).

2.2. Klucze i relacje w modelu relacyjnych baz danych

Z pojedynczym rekordami w tabeli ściśle związany jest unikalny identyfikator pozwalający na jednoznaczne rozróżnienie poszczególnych wierszy. W terminologii baz danych przyjęto nazwę *klucza*. Rozróżnia się kilka ich rodzajów, z których każdy ma swoje zastosowanie w wykonywanych na rekordach operacjach. Przed kluczami stawiane są ograniczenia, które wymagają, aby kolumna/kolumny będące kluczami posiadały unikatowe wartości pozwalające na jednoznaczną identyfikację oraz, aby wartości w nich przechowywane były atomowe (niepodzielne). Klucze można podzielić na dwa główne rodzaje:

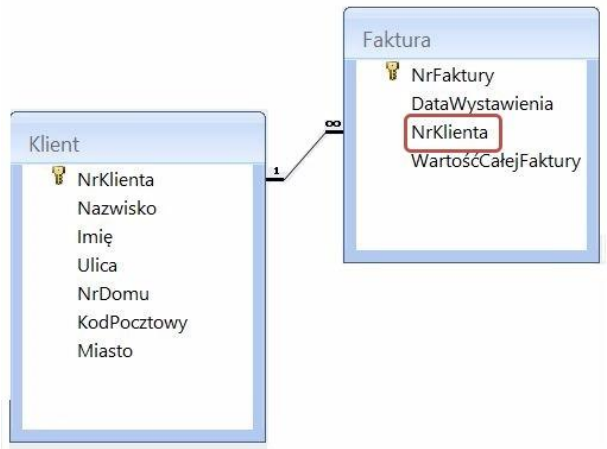
- klucze proste – składające się z pojedynczej kolumny, zawierającej tylko unikatowe wartości;
- klucze złożone – składające się z więcej niż jednej kolumny.

Według przyjętej nomenklatury kolumny, które wchodzą w skład kluczy nazywa się *atrybutami podstawowymi*, natomiast pozostałe kolumny nazywane są *atrybutami opisowymi [16]*. Znajomość pojęcia klucza jest niezbędne do odpytywania bazy danych, gdzie dane pochodzą z wielu różnych tabel.

Poniżej opisane zostały rodzaje kluczy, które znajdują zastosowanie w relacyjnych bazach danych [12]:

- Superklucz – superkluczem nazywamy dowolny zbiór atrybutów, który może jednoznacznie identyfikować wiersz w tabeli. Każda tabela może zawierać wiele takich kluczy, a jednym ze szczególnych przypadków superklucza jest klucz składający się ze wszystkich atrybutów tabeli.
- Klucz kandydujący – ten rodzaj klucza występuje, jako założenie teoretyczne. Jest to jeden z superkluczy, który brany jest pod uwagę przez projektanta bazy danych jako przyszły klucz główny.
- Klucz podstawowy (klucz główny) – jest to najpopularniejszy i najważniejszy rodzaj klucza w teorii baz danych. Jest to wybrany przez projektanta bazy danych klucz będący jednym z kluczy kandydujących, który ostatecznie będzie identyfikatorem pojedynczego rekordu w tabeli. W pracy z obiektami, które mają odniesienie w rzeczywistości, można rozróżniać pojęcie klucza naturalnego oraz sztucznego. Kluczem naturalnym nazywamy identyfikator występujący w świecie rzeczywistym, jak np. numer PESEL osoby lub numer NIP firmy. Klucze sztucznym jest natomiast identyfikator wybrany przez twórcę lub będący po prostu inkrementowaną liczbą całkowitą. Kluczem głównym może być każdy zbiór kolumn, aby spełniony był warunek o jednoznacznej unikalności zbioru kolumn tworzących klucz.

- Klucz obcy – klucz obcy ściśle związany jest z teorią relacyjnych baz danych. Jest to zbiór atrybutów w tabeli będący jednoznacznym wskazaniem na klucz główny innej tabeli. Dlatego też dobrą praktyką jest, aby klucz główny był możliwie jak najkrótszy, co później pozwala na przedstawienie go w innej tabeli jako klucza obcego w możliwie jak najkrótszej postaci. Zbiór atrybutów wybrany, jako klucz główny dba o to, aby w poszczególnych kolumnach znajdowały się tylko te wartości, które istnieją w tabeli docelowej, jako klucze podstawowe. Należy pamiętać, że klucz obcy może się odnosić do obiektów należących do tej samej tabeli. Oznacza to, że może istnieć połączenie węzłów w obrębie tej samej warstwy, gdzie odniesieniem jest inny obiekt reprezentowany w tej encji. Dobrym przykładem jest zbiór uczniów oraz ich kolegów z ławki, gdzie kluczem obcym będzie identyfikator ucznia, który siedzi z uczniem z tabeli źródłowej.



Rys. 2.1 Klucz obcy ID Klienta wraz z relacją (źródło: opracowanie własne)

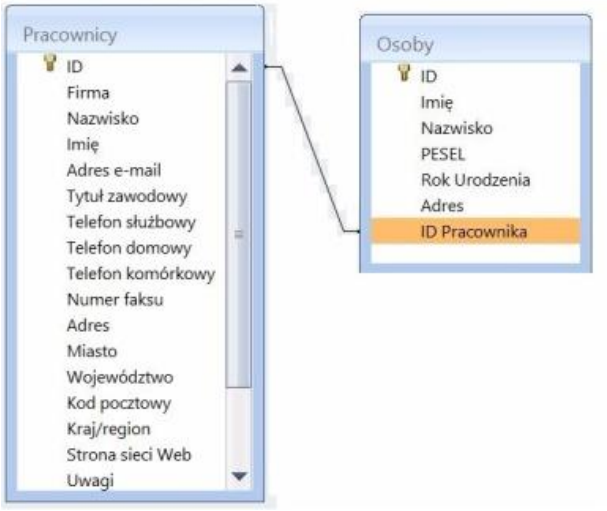
Omówione powyżej zagadnienia są ściśle związane ze strukturą bazy danych i tabeli. Opisana została budowa, a także przedstawione zostały pojęcia kluczy pozwalające na identyfikowanie rekordów. Najważniejszym pojęciem w teorii relacyjnych baz danych jest relacja. Relacją nazywamy związek pomiędzy dwoma tabelami pozwalający na płynne przechodzenie pomiędzy nimi. W przypadku teorii relacyjnych baz danych występuje wiele skomplikowanych struktur jak np. teatralne, obowiązkowe. W poniższym rozdziale przedstawione zostaną struktury dwuargumentowe (binarne), łączące ze sobą dwie tabele. Rozróżniamy trzy główne rodzaje związków [11][12]:

- związek 1:1 (jeden do jeden),
- związek 1:N (jeden do wielu),
- związek N:M (wiele do wielu).

Poniższy podrozdział szczegółowo opisuje przedstawione rodzaje związków wraz z przykładami pozwalającymi na ich wyobrażenie.

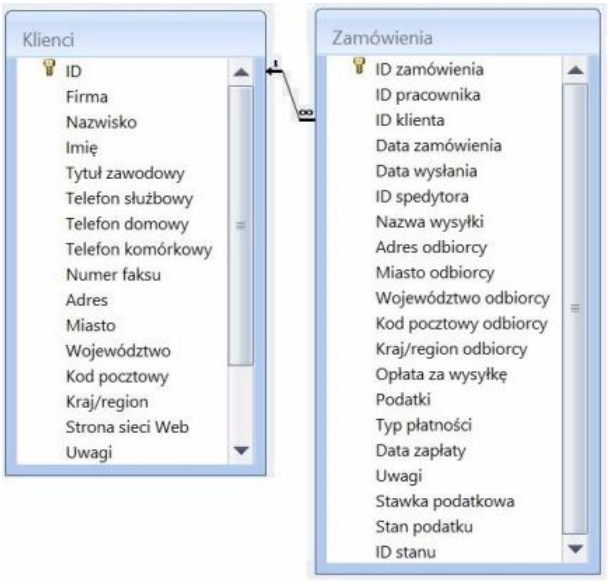
- **Związek 1:1 (Jeden do Jeden)**
  Relacja 1:1 jest najczęściej relacją rozszerzającą, która dostarcza dodatkowych informacji o obiekcie. Najlepszym przykładem jest tabela będąca relacją

główną, która przechowuje osoby, natomiast relacją pochodną jest tabela, która przechowuje pracowników. W tym wypadku związek 1:1 jest zachowany, ponieważ jeden pracownik nie może być dwiema osobami, natomiast jedna osoba nie może być dwoma pracownikami. Model ten ma zastosowanie w odczytywaniu rzadko używanych danych. Jeśli niektóre dane są rzadko odpytywane to można je przenieść do osobnej tabeli i odpytywać tylko w razie potrzeby na podstawie klucza obcego. Innym zastosowaniem jest dodatkowa ochrona, która pozwala na wydzielenie wrażliwych danych do innej tabeli (zarobki pracownika) i zastosowanie wobec niej dodatkowych zabezpieczeń (szyfrowanie, częstszy backup).



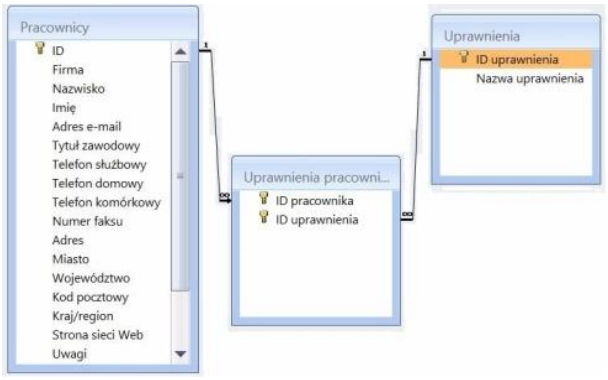
Rys. 2.2 Relacja 1:1 Pracownicy-Osoby (źródło: opracowanie własne)

- **Związek 1:N (Jeden do Wielu)**
  Związek jeden do wielu jest najczęściej używaną spośród relacji. Pozwala na zamodelowanie relacji rodzic-dziecko. Relacja ta pozwala na przyporządkowanie jednego obiektu do wielu innych. Jako przykład można podać klasę w szkole. Do jednej klasy może uczęszczać wielu uczniów natomiast uczeń może być przypisany tylko do jednej klasy.



Rys. 2.3 Relacja 1:N (źródło: opracowanie własne)

- **Związek N:M (Wiele do Wielu)**
  Relacja ta jest rozszerzeniem relacji 1:N, ponieważ jest realizowana zawsze, jako złączenie dwóch takich operacji. Przy modelowaniu relacji N:M konieczne jest dodanie trzeciej tabeli, będącej tabelą łącznikową. Przykładem zastosowania takiej relacji są nauczyciele uczący w szkole oraz przedmioty, do których są przypisani. Wielu nauczycieli może być przypisanych do jednego przedmiotu oraz wiele przedmiotów może być prowadzonych przez jednego nauczyciela.



Rys. 2.4 Relacja N:M (źródło: opracowanie własne)

W relacyjnych bazach danych często pojawia się pojęcie normalizacji [11][12][15]. Jest to jedna z metod wykorzystywana do tworzenia relacyjnych baz danych. Wykorzystuje strategię projektowania wstępującego (bottom-up design). Polega on na dekompozycji większych tabel na mniejsze oraz logicznego ich ekstrahowania w obrębie struktur. Normalizacja jest wielokrotnym procesem dekompozycji struktur według określonych wymagań. Proces ten często opiera się na wyobraźni i intuicji, ponieważ projektant musi łączyć kolejne tabele w logiczną całość. Należy dokładnie przestrzegać zasad wprowadzanych przez kolejne postacie normalne, ze względu na trudność wycofania wprowadzonych zmian. Dobrą metodą sprawdzenia poprawności zaprojektowanej bazy jest test na złączenie bezstratne.

2.3. Postaci normalne bazy

Historia postaci normalnej bazy danych sięga 1972 roku, kiedy E.F. Codd zdefiniował je w swojej pracy [11]. Dwa lata później w 1974 wprowadzono definicje silniejszej trzeciej postaci normalnej zwanej postacią Boyce'a-Codda. Kilka lat później R. Fargin zdefiniował czwartą, a potem piątą postać normalną. W obecnych systemach bazodanowych funkcjonuje już szósta postać normalna, gdzie dane są mocno uwarunkowane od czasu.

Głównym założeniem stawianym przed procesem normalizacji jest potrzeba usunięcia nadmiarowości danych i ułożenia ich w logiczną całość poprzez zastosowanie relacji pomiędzy tabelami. W teorii baz danych wyróżniamy 3 podstawowe postacie normalne, które będą rozważane w ramach niniejszej pracy:

- 1-NF (First Normal Form)
- 2-NF (Second Normal Form)
- 3-NF (Third Normal Form)

Jedną z najważniejszych cech, która świadczy o błędnie zaprojektowanej bazie jest jej nadmiarowość. Poszczególne rekordy posiadają wciąż powtarzające się dane. Redundancja występująca w tym wypadku ma negatywny wpływ na zwiększone wykorzystanie pamięci oraz możliwość powstania niespójności występującej w obrębie baz. Odpowiednim przykładem jest chęć dodania kolumny z pseudonimem autora w przypadku rozważania tabeli odnoszącej się do książek w bibliotece. Każdy autor w tabeli występuje wiele razy, co zmusza projektanta do wypełnienia każdego rekordu wielokrotnie tym samym pseudonimem.

Kolejnym z rodzajów błędów jest anomalia wprowadzania oraz usuwania danych. W przypadku chęci wprowadzenia nowego autora do zbiorów biblioteki konieczne jest dodanie wraz z nim którejś z jego książek. Analogiczna sytuacja występuje przy usuwaniu, gdzie w przypadku chęci usunięcia autora projektant traci informacje o wszystkich jego książkach. Z postaciami normalnymi bazy danych ściśle związana jest zależność funkcyjna.

Definicja 2.2.

Zależność funkcyjna, oznaczona jako $A \rightarrow B$*, jednoznacznie przyporządkowuje do każdej wartości atrybutu A jedną określoną wartość atrybutu B [11]*

Zastosowanie powyżej definicji można zauważyć w przypadku relacji PESEL-Nazwisko. Konkretna wartość numeru PESEL determinuje jedno nazwisko, natomiast nie działa to w drugą stronę, ponieważ z konkretnego nazwiska nie wynika jednoznacznie numer PESEL. Oznacza to więc, że zależność jest jednokierunkowa. Wyróżnia się trzy główne typy zależności funkcyjnych $A_1, A_2, A_3, \dots, A_N \rightarrow B_1, B_2, B_3, \dots, B_M$:

- Trywialne – jeśli zbiór atrybutów $B_1, B_2, B_3, \dots, B_M$ jest podzbiorem $A_1, A_2, A_3, \dots, A_N$;

- Nietrywialne – jeśli co najmniej jeden z atrybutów B_i nie należy do zbioru atrybutów $A_1, A_2, A_3, \dots, A_N$;
- Całkowicie nietrywialnie – jeśli żaden z atrybutów B_i nie należy do zbioru atrybutów $A_1, A_2, A_3, \dots, A_N$;

Zbiór $A_1, A_2, A_3, \dots, A_N$ nazywa się zbiorem determinującym (wyznacznikiem), natomiast zbiór $B_1, B_2, B_3, \dots, B_M$ jest zbiorem zależnym.

Poniżej przedstawiona została przykładowa tabela *Zamówienia*, która zostanie poddana procesowi normalizacji:

Zamówienia

NrFak	DataWystaw	NrKlien	NazwaKlienta	AdresKlienta	KodTowaru	NazwaTowa	CenaTowaru	NrPozycji	IlośćSprzeda	WartośćFakt
4	01.03.2015	Firma A	Szypuła Anna	ul. Jasna 2/12, 42-504 Będzin	D01	Długopis	1,70 zł	1	5	15,40 zł
4	01.03.2015	Firma A	Szypuła Anna	ul. Jasna 2/12, 42-504 Będzin	G01	Gumka	2,30 zł	2	3	15,40 zł
5	17.05.2017	Firma C	Szymczak Tomasz	ul. Krzywa 18/55, 05-800 Pruszków	D01	Długopis	1,70 zł	1	2	4,60 zł
5	17.05.2017	Firma C	Szymczak Tomasz	ul. Krzywa 18/55, 05-800 Pruszków	Z01	Zeszyt	1,20 zł	2	1	4,60 zł
6	07.02.2017	Firma A	Szypuła Anna	ul. Jasna 2/12, 42-504 Będzin	P01	Piórnik	7,50 zł	1	1	11,70 zł
6	07.02.2017	Firma A	Szypuła Anna	ul. Jasna 2/12, 42-504 Będzin	Z01	Zeszyt	1,20 zł	2	2	11,70 zł
6	07.02.2017	Firma A	Szypuła Anna	ul. Jasna 2/12, 42-504 Będzin	L01	Linijka	0,90 zł	3	2	11,70 zł
7	14.09.2016	Firma B	Chmiel Antoni	ul. Długa 13, 30-23 Kraków	P01	Piórnik	7,50 zł	1	1	7,50 zł
8	15.08.2017	Firma C	Szymczak Tomasz	ul. Krzywa 18/55, 05-800 Pruszków	G01	Gumka	2,30 zł	1	2	4,60 zł

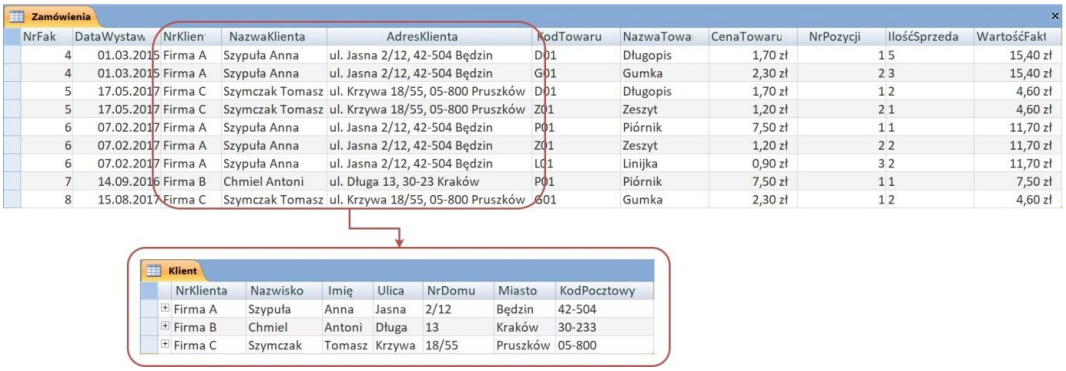
Rys. 2.5 Tabela zamówienia (źródło: opracowanie własne)

2.3.1. Pierwsza postać normalna

Definicja 2.3.

Relacja jest w pierwszej postaci normalnej wtedy i tylko wtedy, gdy każdy atrybut niekluczowy jest funkcjonalnie zależny od klucza głównego. [11]

Główną zasadą pierwszej postaci normalnej bazy jest atomowość jej danych. Każde z pól powinno przechowywać pojedynczą informację, aby możliwe było wykonywanie efektywnych zapytań. Na tym etapie normalizacji bazy pojawia się również pojęcie klucza głównego, który identyfikuje każdy z rekordów.



Rys. 2.6 Pierwsza postać normalna – Tabele (źródło: opracowanie własne)

Można zauważyć, że w tabeli *Zamówienia* informacje o kliencie są „zbite" i nieczytelne. Na szczególną uwagę zasługuje kolumna *AdresKlienta*, która zawiera wszystkie informacje o adresie. Aby tabela była w 1NF należy rozdzielić kolumny *NazwaKlienta* i *AdresKlienta* tak, aby przechowywały w pełni atomowe dane (rys. 2.6). Pierwsza postać normalna nie przekształca tabel na mniejsze.

2.3.2. Druga postać normalna

Definicja 2.4.

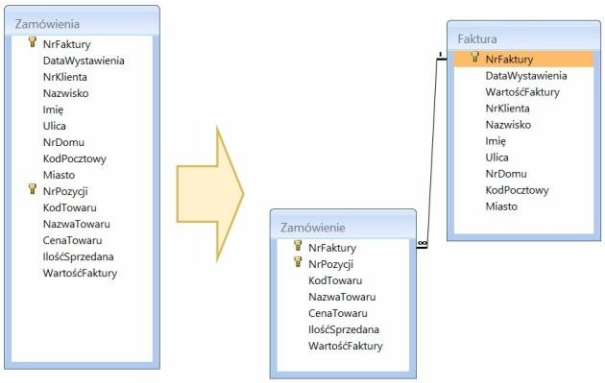
Relacja jest w drugiej postaci normalnej, jeśli jest już w pierwszej postaci normalnej i każdy atrybut niekluczowy jest w pełni funkcjonalnie zależny od klucza głównego. [11]

Proces normalizacji tablicy z 1NF do 2NF polega na znalezieniu, a następnie usunięciu wszystkich częściowych zależności funkcyjnych. Przy znalezieniu niepełnej zależności funkcyjnej, należy przenieść do innej relacji wszystkie atrybuty niekluczowe wraz z częścią klucza głównego, od którego zależą. Z drugą postacią normalną ściśle związane jest pojęcie pełnej zależności funkcyjnej.

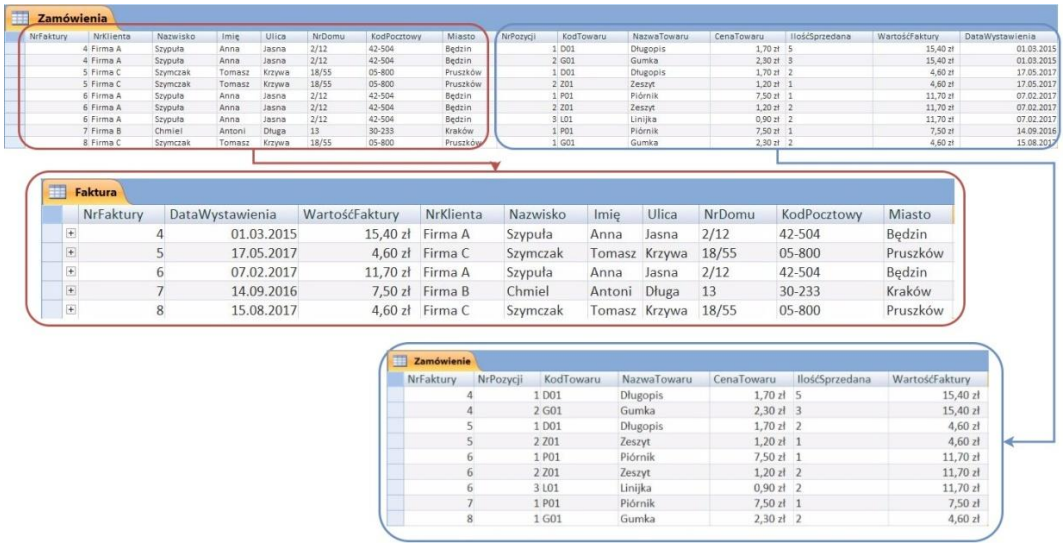
Definicja 2.5.

Pełna zależność funkcyjna $A \rightarrow B$ jest oceniana, w przypadku gdy klucz A jest określony na zbiorze atrybutów $A_1, A_2, A_3, \ldots, A_N$ i polega ona na tym, że klucz B jest funkcjonalnie zależny od całego zbioru atrybutów A, lecz nie jest on funkcjonalnie zależny od właściwego podzbioru A. [11]

Każda tabela w 1NF zawierająca klucz prosty (tzn. kluczem jest pojedyncza kolumna) jest automatycznie w drugiej postaci normalnej. W tabeli *Zamówienia* występuje częściowa zależność tranzytywna. Kolumny *DataWystawienia*, *NrKlienta*, *WartośćFaktury* zależą od *NrFaktury* i w związku z tym powinny być przeniesione do osobnej tabeli, która została nazwana *Faktura*.



Rys. 2.6 Druga postać normalna – Relacje (źródło: opracowanie własne)



Rys. 2.7 Druga postać normalna – Tabele (źródło: opracowanie własne)

Przekształcanie tabeli do drugiej postaci normalnej skutkuje zwiększeniem ilości tabel poprzez wydzielenie zależnych od siebie kolumn z tabeli głównej.

2.3.3. Trzecia postać normalna

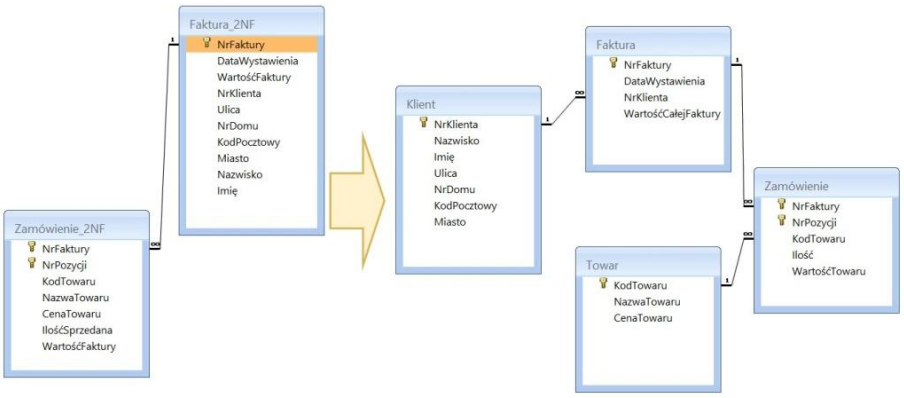
Definicja 2.6.

Relacja jest w trzeciej postaci normalnej wtedy i tylko wtedy, gdy jest w drugiej postaci normalnej i każdy atrybut niekluczowy jest bezpośrednio (nietranzytywnie) zależny od klucza głównego. [11]

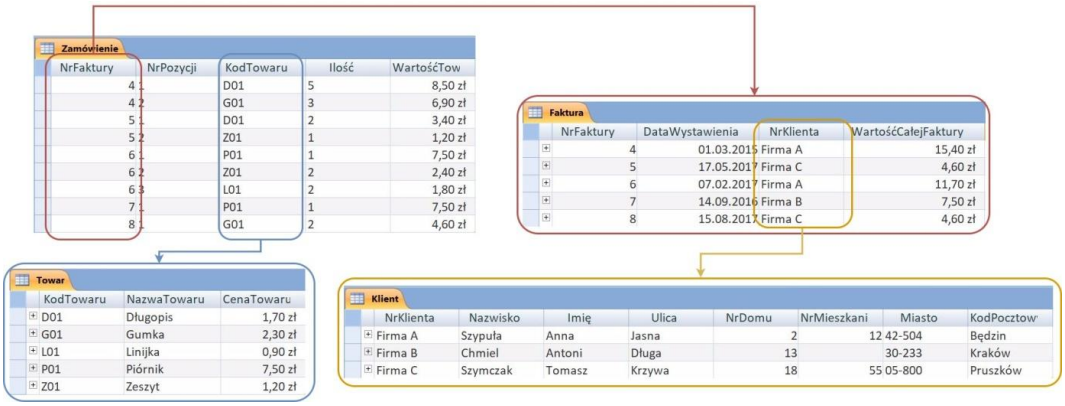
Z trzecią postacią normalną związane jest pojęcie zależności tranzytywnej. Zależnościa tranzytywna występuje w przypadku, jeśli pomiędzy atrybutami X, Y, Z będącymi w relacji R występują odpowiednie zależności funkcyjne, tj. $X \rightarrow Y$ i $Y \rightarrow Z$. Mówi się wtedy, że atrybut Z jest tranzytywnie zależny od atrybutu X, ponieważ jest od niego przechodnio zależny.

Proces normalizacji z *2NF* do *3NF* wykonuje się poprzez usunięcie zależności przechodnich występujących w danej relacji. Pierwszą tabelą, która zostanie poddana normalizacji, jest *Faktura*. Można zauważyć, że kolumny *Nazwisko, Imię, Ulica, NrDomu* są zależne tranzytywnie od *NrFaktury* poprzez kolumnę *NrKlienta*. W celu wyeliminowania tej zależności należy wyodrębnić tabelę *Klient*, gdzie przechowywane będą informacje o klientach.

Drugą tabelą, która wymaga przekształcenia, jest *Zamówienie*, gdzie *CenaTowaru* i *NazwaTowaru* zależą tranzytywnie od klucza głównego poprzez *KodTowaru*. Efekt przekształceń oraz powstałych relacji został przedstawiony poniżej (*Rys. 2.8*) i (*Rys. 2.9*).



Rys. 2.8 Trzecia postać normalna - Relacje



Rys. 2.9 Trzecia postać normalna - Tabele

2.3.4. Postać normalna Boyce'a-Codda (BCNF)

Definicja 2.7

Relacja R jest w postaci Boyce'a-Codda wtedy i tylko wtedy, gdy dla każdej nietrywialnej zależności funkcyjnej $A_1, A_2, A_3, \dots, A_N \rightarrow B_1, B_2, B_3, \dots, B_M$ *spełnionej w R zbiór* $\{A_1, A_2, A_3, \dots, A_N\}$ *jest nadkluczem R [11]*

W celu stwierdzenia, czy relacja jest w tej postaci, konieczne jest sprawdzenie czy wszystkie zależności funkcyjne są spełnione i czy wyznaczniki są kluczami lub zawierają klucz. Poniżej przedstawiona została tabela, która jest w *3NF*, ale nie spełnia *BCNF*. W tabeli występuje zależność funkcyjna poprzez $Miasto \rightarrow KodPocztowy$, kolumna *KodPocztowy* jest zależna tylko od części klucza, co skutkuje niespełnieniem warunku na postać normalną Boyce'*a-Codda*.

Rys. 2.10 Postać normalna Boyce'a-Codda – Adres (źródło: opracowanie własne)

2.3.5. Czwarta postać normalna

Definicja 2.8.

Relacja R jest w czwartej postaci normalnej wtedy i tylko wtedy, gdy jest w postaci BCNF i nie zawiera żadnych nietrywialnych zależności wielowartościowych, które to zależności muszą być wyłączone do oddzielnych schematów. [11]

Aby przedstawić tabelę w czwartej postaci normalnej konieczne jest znalezienie zależności wielowartościowej, w której nie są spełnione wymagania 4NF, np. $A_1, A_2, A_3, \dots, A_N \rightarrow B_1, B_2, B_3, \dots, B_M$, gdzie $A_1, A_2, A_3, \dots, A_N$ jest nadkluczem, co powoduje podzielenie relacji R na dwa schematy:

- Pierwszy schemat zawierający wszystkie atrybuty *A* i *B*.
- Drugi schemat składający się z wszystkich atrybutów typu *A* oraz wszystkich atrybutów z relacji *R*, które nie występują ani w *A* ani w *B*.

2.3.6. Piąta postać normalna

Definicja 2.9.

Relacja jest w piątej postaci normalnej, jeśli jest w czwartej postaci normalnej i nie istnieje jej rozkład odwracalny na zbiór mniejszych tabel, czyli nie zawiera zależności połączeniowych. [11]

Rozkład odwracalny (połączeniem bezstratnym) oznacza taką dekompozycję tablicy na mniejsze, aby przy złączeniu naturalnym otrzymanych tablicy zachowana została relacja pierwotna. Wymagane jest, aby żaden z rekordów nie został utracony, oraz żeby po złączeniu nie wystąpiły żadne nieautentyczne wiersze.

Proces projektowania baz danych jest zadaniem nietrywialnym i wymaga specjalistycznej wiedzy oraz informacji na temat modelowanego procesu. Konieczne są tutaj wymagania sprzętowe, jak również wymagania biznesowe, jakie są stawiane przed projektowaną bazą, aby była ona jak najefektywniejsza. Mimo że pojęcie normalizacji baz danych wprowadza wiele uproszczeń oraz daje ogromne możliwości, to jest ono często nadużywane. Projektanci często próbują za wszelką cenę zaprojektować bazę spełniającą wszystkie postacie normalne. Prowadzi to w wielu przypadkach do nadmiernej złożoności operacji złączeń (JOIN) w następstwie dzielenia tabel w trakcie normalizacji. Poniżej przedstawione zostały zalety i wady normalizacji oraz najczęstsze błędy, których należy się wystrzegać.

Zalety normalizacji:

- Rozwiązanie problemu anomalii dodawania, modyfikowania oraz usuwania rekordów z bazy;
- Zmniejszenie ogólnej liczby danych przechowywanych w bazach;
- Przyspieszenie wykonywania określonych zapytań – bezpośrednio odwoływanie się do tabeli, więcej indeksów klastrowych;

- Bardziej efektywne składowanie danych na dysku, ponieważ wąskie tabele wymagają mniej operacji I/O;
- Lepsze zarządzanie transakcjami;
- Przejrzystość w bazach.

Wady normalizacji:

- Zmiana struktur baz danych;
- Modyfikowanie relacji pomiędzy tabelami;
- Duże koszty odwrócenia procesu normalizacji;
- Zwiększone skomplikowanie wykonywanych zapytań ze względu na konieczność wykorzystywania złączeń (tzw. *JOIN-ów*);
- Niższa efektywność indeksów w tabelach ze względu na wszechobecne łączenie tabel;
- Konieczna jest znajomość całego procesu biznesowego ad hoc, aby wykonać normalizację.

2.4. Optymalizacja zapytań SQL

Wszystkie systemy baz danych muszą być w stanie odpowiadać na żądania informacji od użytkownika, tzn. wykonywać zapytania użytkowników i uzyskiwać wymagane informacje z systemu w oczekiwany i niezawodny sposób. Zapytania do bazy danych definiowane są zazwyczaj w sposób deklaratywny, a system jest odpowiedzialny za wybór odpowiedniej drogi wykonania [17].

Optymalizator zapytań (optymalizator kwerend) jest częścią relacyjnego oprogramowania baz danych przeznaczonym do analizy zapytania SQL, a następnie decyduje, którą ścieżkę wybrać, aby zapytanie było w pełni i jak najlepiej zoptymalizowane.

Wyróżnia się dwa główne rodzaje optymalizatorów [17]

- Optymalizator oparty na regułach (RBO – Rules Based Optimization),
- Optymalizator oparty na kosztach (CBO – Cost Based Optimization).

Optymalizator oparty na regułach

Optymalizatorem opartym na regułach nazywamy optymalizator, którzy wykorzystuje zestaw odgórnie zdefiniowanych reguł do przetworzenia instrukcji SQL, zamiast wyszukiwać różnych dróg otrzymania wyniku. Jest to stosunkowo stara metoda wykonywania zapytań i powoli jest ona wypierana przez CBO. Regułą może być wykorzystanie indeksu, tzn. jeśli dla danej tabeli dostępny jest indeks, to zostanie on zawsze wykorzystany. Wiązało się to z dodatkowymi kosztami, ponieważ użycie indeksu w zapytaniu nie zawsze jest wskazane. Dobrym przykładem jest kolumna, która określa płeć i posiada jedną z dwóch wartości *mężczyzna* i *kobieta*. Użytkownik wysyła następujące zapytanie do bazy danych:

*Select * from Osoby Where płeć = 'mężczyzna'*

W przypadku zwrócenia przez powyższe zapytanie około 50% rekordów szybko okazuje się, że użycie indeksu spowolniło działanie zapytania. W powyższym przypadku szybsze byłoby przeszukanie całej tabeli i sprawdzenie wartości w kolumnie płeć, a następnie zwrócenie wyniku. Inżynierowie Oracle w trakcie badań doszli do wniosków, że użycie indeksu nabiera sensu, kiedy liczba zwracanych wierszy nie przekracza 5-10% całej tabeli. Jak można zauważyć RBO nie zawsze podejmował trafne decyzje, a poprzez zdefiniowany zestaw dyskretnych reguł nie miał takiej możliwości.

Optymalizator oparty na kosztach (CBO)

Optymalizatory kosztów korzystają z pewnych statystyk gromadzonych w bazach danych. Niektóre spośród nich obejmują np. liczbę unikatowych wartości kolumny indeksowanej lub liczby wierszy w tabeli. Z optymalizacją względem kosztów ściśle związane jest pojęcie selektywności. Przykład działania CBO można odnieść do powyższego zapytania dla RBO. W przypadku rozważania grupy osób, z których 95% stanowią kobiety można wydzielić dwie odrębne drogi przetwarzania. W przypadku wyszukiwania mężczyzn korzystamy z indeksów, natomiast w przeciwnym wypadku przeszukujemy całą tabelę. Optymalizator oparty na kosztach posiada w wewnętrznej strukturze tego typu informacje i pozwala na podjęcie optymalnej decyzji.

Z optymalizatorem ściśle powiązane jest pojęcie *selektywności* i *unikalności*, a dotyczy indeksów bazodanowych. Selektywność wyrażamy następującym wzorem [21]:

$$selektywność = \frac{unikalność}{liczba\,rekordów} \times 100\%$$

Unikalnością jest liczba unikalnych wartości kolumny w obrębie tabeli. Dla przykładu z *Osobami* unikalność wynosi 2, ponieważ atrybut rozróżnia dwie wartości (mężczyzna, kobieta).

Selektywność określa procentową ilość różnych wartości w danym zestawie próbek. Niska wartość oznacza, że w obrębie kolumny nie występuje zbyt wiele zmian, co pozwala na podjęcie odpowiednich kroków przez silnik bazodanowy. Zgodnie z powyższym można założyć, iż nie powinno się używać indeksu dla niskiego współczynnika selektywności, co w połączeniu ze statystykami odnośnie procentowego rozłożenia wartości w kolumnie silnik bazodanowy otrzymuje pełny przekrój przypadków, co pozwoli mu na podjęcie optymalnej decyzji.

Zasadniczo Optymalizator CBO jest uogólnieniem optymalizator RBO, którego użycie z wraz z kolejnymi wersjami baz danych zanika.

2.5.Podsumowanie

Relacyjne bazy danych dają ogromne możliwości dla projektów aplikacji, ale niestety wiążą się z nimi pewne restrykcje, które w niektórych przypadkach mogą być powodem wielu problemów. Poniżej przedstawiono najważniejsze ograniczenia oraz zagrożenia, na jakie narażona jest aplikacja korzystająca z relacyjnej bazy danych:

- Złożoność danych – Dane w RDBMS znajdują się w wielu tabelach, które połączone są ze sobą poprzez współdzielone wartości kluczy. Błędy popełnione w czasie projektowania relacyjnej bazy danych objawiają się najczęściej w przyszłości podczas dalszego rozwoju. Niekiedy projektanci mogą wybrać nieodpowiedni typ danych lub wykonać zbyt złożony projekt, który będzie wykazywał się sztywnością i brakiem elastyczności przy rozwoju.
- Złamane klucze i rejestry – Relacyjne bazy danych wymagają współdzielonych kluczy do łączenia rozłożonych na kilka tabel informacji. Problemem mogą się w tym wypadku okazać używane typy danych. Dla przykładu można podać zamówienie połączone z numerem klienta. Jeśli klucz obcy w tabeli zamówień jest innego typu niż klucz główny w tabeli z klientami, to nie ma fizycznej możliwości połączenia ze sobą obu rekordów bez dodatkowej pracy wykonanej przez programistę. Stwarza to ogromne ryzyko błędu podczas rozwoju bazy danych.
- Doświadczenie programisty – czynnik ludzki jest jednym z najniebezpieczniejszych podczas rozwoju i projektowania bazy danych. Niekiedy projekt wykonywany jest przez różne osoby o różnym poziomie zaawansowania i znajomości dobrych praktyk. Prowadzi to do różnic w poszczególnych częściach bazy, co w późniejszym etapie może prowadzić do załamania zapytań i trudności w odczycie danych.
- Wydajność sprzętowa – Relacyjna baza danych charakteryzuje się niekiedy dużą złożonością ze względu na dużą liczbę tabel oraz składa się głównie ze skomplikowanych zapytań odczytujących i łączących dane z wielu tabel. Wiąże się to ze zwiększonymi kosztami użytkowania ze względu na potrzebę zakupu lepszych serwerów i dokładniejszej ich konfiguracji.

Optymalizatory zapytań są kluczowym czynnikiem odpowiadającym za wydajność przetwarzania zapytań. Implementacja optymalizatora RBO jest znacznie prostsza, ponieważ nie występują w niej czynniki zmienne. W niniejszej pracy zaimplementowany zostanie mechanizm optymalizacji oparty na regułach, który każde zapytanie przetwarzać będzie w identyczny sposób.

3. Sztuczne systemy skojarzeniowe ASS

W ciągu ostatnich lat nastąpił ogromny wzrost ilości dostępnych danych cyfrowych pochodzących z Internetu, smartfonów, mediów społecznościowych oraz innych źródeł. Pomimo stałego wzrostu mocy obliczeniowej oraz stosowania coraz bardziej skomplikowanych algorytmów relacyjne bazy danych stają się niewystarczające do rozwiązania wszystkich stawianych przed nimi problemów. Z modelem relacyjnym związany jest duży narzut związany z obsługą wielu tabel oraz skalowaniem w przypadku zarządzania Big Data [13]. Rozwiązaniem mogłoby być dodanie kolejnych klastrów, które jednak nie rozszerzają się dynamicznie, dodatkowo duże rozwiązania danych nie skalują się wystarczająco dobrze przy użyciu RDBMS. Największą natomiast wadą systemów bazodanowych jest brak mechanizmów uczenia i wnioskowania. Relacyjne bazy danych poprzez swoje zachowania imitują część zachowań inteligentnych, jednak jest to wciąż niewystarczające.

W 1956 r. przedstawiona została pierwsza definicja sztucznej inteligencji, a zagadnienie to rozwijane jest do dnia dzisiejszego. Obecne mechanizmy znajdują zastosowania w wielu dziedzinach techniki jak eksploracja danych (*data mining*) [2], czy inteligencji obliczeniowej (*computational intelligence*) [1][5]. W niniejszej pracy omówiono jeden z działów sztucznej inteligencji, jakim są sztuczne systemy skojarzeniowe, które świetnie nadają się do analizy dużych zbiorów danych i z powodzeniem mogą zastąpić model relacyjny. Nieodłączną strukturą związaną ze „strukturami asocjacyjnymi" są grafy. Pozwalają one na implementację mechanizmów wykorzystywanych w sieciach neuronowych oraz swoją budową przypominają sieć neuronów w mózgu [3]. Dzięki odpowiedniemu powiązaniu możliwa jest natychmiastowa możliwość odczytu danych. W niniejszym rozdziale omówiona została Grafowa Asocjacyjna Struktura Danych (AGDS), której twórcą jest dr. hab. Adrian Horzyk [1]. Zbiór ten łączy dane w zależności od podobieństwa oraz sąsiedztwa, co pozwala na określenie pewnych wzorców, jak również wnioskowanie o podobieństwach. Struktura ta idealnie nadaje się do przetwarzania danych dowolnego rodzaju ze względu na możliwość szybkiego dodawania lub usuwania danych. Istotną zaletą omawianych struktur grafowych jest szybki czas zmieniania istniejących rekordów ze względu na zmianę połączenia pomiędzy wartością parametru i referencją do rekordu.

W niniejszym rozdziale zostaną przedstawione podstawy teoretyczne, z których wynika model asocjacyjny. Następnie omówione zostaną zasady asocjacji wraz z typami relacji asocjacyjnych. Następnie omówiona zostanie struktura AGDS, a wraz z nią przedstawiony zostanie prosty przykład jej tworzenia. Kolejnym krokiem będzie zaproponowanie sposobu optymalizacji grafu poprzez zastąpienie posortowanej listy elementów B-Drzewem. Na koniec omówiony zostanie proces przetwarzania modelu

relacyjnego bazy danych składającego się z tabel złożonych i wirtualnych wraz z algorytmem parsowania zapytania SQL i odczytu danych z wykorzystaniem słowa kluczowego JOIN. Jako podsumowanie omówione zostaną sposoby rozszerzenia grafu AGDS poprzez zastosowanie AVB Drzew, które są strukturą specjalnie dopasowaną do tego typu grafów. Dzięki wykorzystaniu AVB Drzewa oraz dodaniu czynnika czasowego omówiony zostanie graf DASNG [2], będący naturalnym neuronowym rozszerzeniem grafu AGDS.

3.1. Wprowadzenie teoretyczne

Najważniejszym terminem związanym ze sztucznymi systemami skojarzeniowymi jest pojęcie **asocjacji (skojarzenia)**.

Definicja 3.1

Asocjacją nazywane jest pewne powiązanie ze sobą dwóch lub kilku zdefiniowanych wcześniej elementów, np. wzorców, danych, zbiorów lub klas. [1]

Głównym celem stosowania asocjacji pomiędzy obiektami jest potrzeba szybkiego przeszukiwania zbiorów danych oraz potrzeba eliminacji instrukcji warunkowych i iteracyjnych w celu przyspieszenia procesów przetwarzania danych. Ludzki mózg dzięki dostarczonym danym na podstawie ich kontekstu potrafi przeprowadzić proces wnioskowania w sposób naturalny i podjąć odpowiednie działania w oparciu o wydedukowane informacje. Imitacja ludzkiego mózgu jest głównym celem, który stawiany jest przed systemami skojarzeniowymi. Kolejnym pojęciem charakteryzującym sztuczne systemy skojarzeniowe jest tzw. **aktywna skojarzeniowość** (zwana krótko skojarzeniowością**)** [1][3][4].

Definicja 3.2

„Skojarzeniowością systemu nazywamy zdolność do dynamicznego, plastycznego, aktywnego, reaktywnego i grafowego wiązania ze sobą danych, ich kombinacji, a także układów. Informacją zwrotną jest podobieństwo, zależność oraz kontekst ich występowania w przekazywanym układzie, jak również w kontekście innych danych.”[1]

Dzięki skojarzeniowości możliwe jest przekształcanie zbudowanych układów w sposób dynamiczny poprzez automatyczne dopasowanie nowych wzorców do istniejących. Dzięki temu nowe dane dopasowywane są w sposób efektywny, a proces ich asocjacji charakteryzuje się szybkością i oszczędnością w dalszym przetwarzaniu.

Wobec powyższych definicji i różnic występujących pomiędzy strukturami biologicznymi i sztucznymi można wyróżnić dwie główne grupy systemów skojarzeniowych [1]:

- Biologiczne systemy skojarzeniowe (BAS) – są to biologiczne układy nerwowe, z których zbudowany jest ludzki mózg, należą do nich: neurony, receptory, synapsy, połączenia i efektory.
- Sztuczne systemy skojarzeniowe (AAS) – ich głównym celem jest odwzorowanie systemów BAS poprzez wykorzystanie występujących w nich odpowiedników funkcjonalnych przeniesienie ich na platformę sprzętową.

System skojarzeniowy często porównywany jest do relacyjnych baz danych, ze względu na swoje zastosowania. W bazach danych informacje powiązane są ze sobą w luźny sposób, dzięki czemu możliwe jest szybkie dodawanie jak i ich usuwanie. Systemy skojarzeniowe budowane są systematycznie, a każde przychodzące dane są wiązane z istniejącą już reprezentacją poprzednich danych. Wraz z rozrostem systemu pewne asocjacje występujące pomiędzy danymi są wzmacniane i uwidaczniają się, co nie występuje w przypadku baz danych. System skojarzeniowy nie jest pamięcią, którą można dowolnie manipulować. Całkowita dowolność byłaby wysoce nieefektywna ze względu na potrzebę przebudowywania części grafu, która jest połączona z usuwanym rekordem. Główną zaletą systemów skojarzeniowych są informacje odwzorowane bezpośrednio w zbudowanej strukturze. Zostało to przedstawione na poniższym prostym przykładzie.

Przykład 3.1.
Na zbiorze studentów wyszukać nazwisko ostatniej osoby na podstawie kolejności alfabetycznej.

W przypadku sztucznego systemu skojarzeniowego wszystkie wartości atrybutu *Nazwisko* są połączone wzajemnie w kolejności alfabetycznej, więc wyszukanie ostatniego nazwiska sprowadza się do odczytania węzła, który nie ma następnika będącego obiektem zawierającym nazwisko.

Skojarzenia, jakie powstają pomiędzy węzłami nie są przypadkowe i modelowane są w oparciu o zdefiniowane relacje zgodnie z przyjętymi zasadami. Zostały nazwane zasadami asocjacji i zostały omówione w kolejnej sekcji.

3.2. Relacje asocjacyjne

Najważniejszą częścią tworzonego systemu skojarzeniowego jest zastosowanie odpowiednich nieprzypadkowych połączeń między danymi. Przy przetwarzaniu kolejnych porcji danych połączenia byłyby wzmacniane, bądź osłabiane. Głównym celem powiązań jest naturalne tworzenie podobieństw pomiędzy obiektami na podstawie zdefiniowanych wcześniej własności. W niniejszym rozdziale relacje asocjacyjne odwzorowywane są w strukturach grafowych poprzez powiązania asocjacyjne.

Wyróżniamy następujące rodzaje powiązań asocjacyjnych [1]:

- Relacja Asocjacyjnego Podobieństwa (ASIM – associative similarity),
- Relacja Asocjacyjnego Następstwa (ASEQ – associative sequence),
- Relacja Asocjacyjnego Kontekstu (ACON – associative context),
- Relacja Asocjacyjnego Definiowania (ADEF – associative definition),
- Relacja Asocjacyjnego Tłumienia (ASUP – associative suppression).

Relacja Asocjacyjnego Podobieństwa (ASIM)

Relacją tą nazywamy naturalne połączenia pomiędzy bliskimi, podobnymi danymi. W grafie powiązanie to występuje na skutek występowania bliskich siebie danych w kontekście pewnej określonej metryki. Podobieństwo może być na podstawie zdefiniowanych wag, co w przypadku pobudzenia jednego węzła skutkuje oddziaływaniem na sąsiadów siła wprost proporcjonalna do przyjętych wag.

Relacja Asocjacyjnego Następstwa (ASEQ)

Relacją typu ASEQ są odwzorowaniem chronologii występującej pomiędzy danymi. Powstają one na skutek pojawiania się podobnych sekwencji danych w systemie, docierających do odpowiednich węzłów (neuronów). Skutkuje to utrwaleniem informacji i możliwym wzmocnieniem takiego połączenia. W przyszłości może to powodować pobudzenie neuronów na podstawie utrwalonych w przeszłości skojarzeń. Ich głównym zadaniem jest wiązanie ze sobą potencjalnie odrębnych części systemu, co prowadzi do utrwalenia pewnego kontekstu, wspomagając przy tym wywołania w przyszłości.

Relacja Asocjacyjnego Kontekstu (ACON)

Powyższa relacja jest pochodną powiązania typu ASEQ. Charakteryzuje się głównie opóźnieniem w czasie wykonania i wykorzystaniem kontekstu wcześniejszych aktywacji neuronów w sieci [1][6]. Są one odtwarzane, co pewien okres czasu na podstawie pobudzania neuronów, od których zależą. Czynnikiem wpływającym na siłę ich oddziaływania na system jest odstęp czasu pomiędzy kolejnymi wywołaniami. Dzięki swojej strukturze mogą zarówno hamować jak i pobudzać zależne od siebie neurony.

Relacja Asocjacyjnego Definiowania (ADEF)

Powiązanie zachodzi podczas równoczesnego pobudzenia wielu neuronów w różnych częściach systemu. Ich powstawanie pozwala na łączeniu wielu różnych węzłów, ich zakresów, jak również kombinacji i układów, co skutkuje powstawaniem całkiem nowych kombinacji. Powstawanie relacji jest głównie zależne od typów połączeń sąsiadujących oraz kolejności aktywacji poszczególnych neuronów.

Relacja Asocjacyjnego Tłumienia (ASUP)

Powiązaniem ASUP nazywamy relacje, które łączą ze sobą dane i ich kombinacje, skutkując przy tym wyostrzeniem lub dyskryminacją obiektów reprezentowanych przez określone neurony. Skutkuje to utworzeniem w grafie danych pewnych kombinacji, które tworzą mapę kombinacji danych. Połączenia te wykluczają istnienie redundantnych danych, co umożliwia budowanie reprezentacji

powtarzających się kombinacji. Głównym zadaniem połączeń typu ASUP jest uniemożliwienie aktywacji podobnych neuronów zapobiegając przy tym redundancji działań i pobudzenia neuronów, które w skutek wnioskowania zostały stłumione. Wielkość i dokładność pokrycia przestrzeni zależy od ilości neuronów jak również liczby podobnych połączeń.

Przedstawione powyżej połączenia mogą znaleźć zastosowanie w strukturach grafowych występujących w obrębie różnych sztucznych systemów skojarzeniowych. Przykładem ich zastosowania będzie omówiona w kolejnym rozdziale grafowa asocjacyjna struktura danych AGDS.

3.3. Grafowa asocjacyjna struktura danych AGDS

Jedną ze struktur należących do grupy sztucznych systemów skojarzeniowych są grafy AGDS. Są to obiekty modelujące grupę luźno powiązanych ze sobą danych poprzez określenie dodatkowych mechanizmów poszerzających możliwości interakcji pomiędzy nimi. Głównym celem ich zastosowania jest umożliwienie efektywnego przeszukiwania i wnioskowania. Na podstawie zdefiniowanego kontekstu tworzą one mapę powiązań, która daje bezpośredni dostęp do parametrów i osadzonych na nich wartości wraz z przyłączonymi do nich danymi. Umożliwia uzyskiwanie informacji w czasie stałym do większości często poszukiwanych informacji wynikających z relacji pomiędzy danymi i obiektami. Niniejszy rozdział składa się na definicję grafu AGDS, a także omówienie jego budowy. Następnie przeprowadzony zostanie proces przekształcania prostej relacyjnej bazy danych na strukturę skojarzeniową. W podsumowaniu przedstawione zostaną zalety i wady zastosowanego rozwiązania, a także możliwości jego dalszego rozszerzenia [1].

3.3.1. Definicja

Grafowa Asocjacyjna Struktura Danych należy do skojarzeniowych reprezentacji danych charakteryzującą się możliwością horyzontalnego i wertykalnego powiązania pomiędzy danymi. AGDS są rodzajem grafowych baz danych implementujących relacje pomiędzy nimi w postaci asocjacji [1].

Na strukturę grafu składają się powiązania:

- asocjacyjnego podobieństwa (ASIM),
- asocjacyjnego następstwa (ASEQ),
- asocjacyjnego definiowania (ADEF).

AGDS nie odwzorowuje właściwości czasowych, w związku z tym pozostałe powiązania omówione w poprzedniej sekcji nie mają tutaj zastosowania. Główną cechą omawianej struktury jest jej pasywność, czyli zbudowana jest z węzłów, które nie posiadają dynamicznych, reaktywnych właściwości neuronów i nie pozwalają na ich pobudzenie [3]. Oprócz węzłów w skład grafu wchodzą krawędzie, które mogą posiadać wagi wewnętrzne określające np. ilość połączeń do danego węzła, jak

również mogą faworyzować wartości średnie bądź graniczne. Utworzona struktura podlega standardowym zasadom przeszukiwania jak w zwykłych grafach. Dodatkową zaletą AGDS jest możliwość przekształcenia jej w aktywny asocjacyjny graf neuronowy (AANG) [1] czy też DASNG [2], które udostępniają wszystkie możliwości związane z wykorzystaniem neuronów w miejsce zwykłych węzłów w grafie.

Struktury grafowe AGDS są bardzo dobrą alternatywą dla szeroko wykorzystywanych relacyjnych baz danych ze względu na eliminację duplikatów, przez co zajmują mniej pamięci oraz dodatkowo dostarczają informacji o różnych powiązaniach (np. korelacjach i podobieństwach) pomiędzy danymi i obiektami. Składają się zarówno z krawędzi skierowanych jak i nieskierowanych.

Definicja 3.3:

Grafem AGDS nazywamy uporządkowaną siódemkę

$$AGDS = (VV, VR, VS, VC, ESIM, ESEQ, EDEF)$$

gdzie, VV, VR, VS, VC są różnymi typami wierzchołków, natomiast ESIM, ESEQ, EDEF są krawędziami [1].

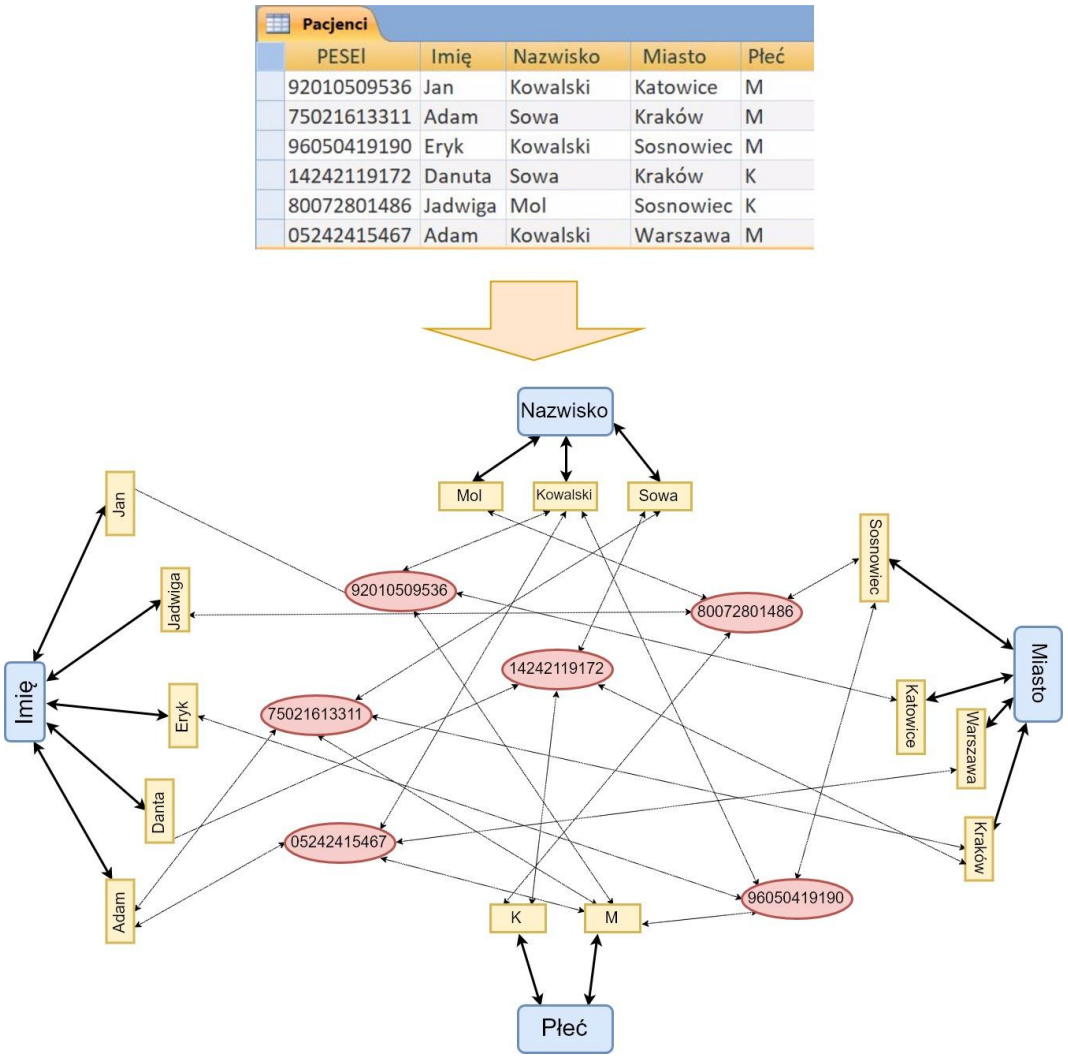
Poniżej zostały omówione znaczenia poszczególnych części grafu:

- *VV* (*value vertex*) – zbiór wierzchołków reprezentujący pojedynczą wartość,
- *VR (range vertex)* – zbiór wierzchołków reprezentujących przedział wartości,
- *VS (subset vertex)* – zbiór wierzchołków reprezentujący podzbiór wartości,
- *VC (combination vertex)* – zbiór wierzchołków reprezentujących kombinację wartości,
- *ESIM (similarity edge)* – zbiór krawędzi nieskierowanych łączących asocjacyjnie podobne wierzchołki,
- *ESEQ (sequence edge)* – zbiór krawędzi skierowanych łączących asocjacyjnie następne wierzchołki,
- *EDEF (defining edge)* – zbiór krawędzi dwustronnie skierowanych łączących wierzchołek definiujący z wierzchołkiem definiowanym.

Charakterystyczną cechą grafu jest występowanie krawędzi dwustronnie skierowanych, które posiadają po obu stronach wartości wag zależne od kierunku poruszania się po nich. W celu lepszego zrozumienia występujących krawędzi należy przyjrzeć się pewnym zależnością zachodzącym pomiędzy nimi. Jeśli pewna krawędź ESEQ definiuje następstwo asocjacyjne ASEQ lub asocjacyjny kontekst definiuje któryś z wierzchołków, to wtedy jest wewnątrz zbioru krawędzi EDEF, co umożliwia równoczesne odwzorowanie następstwa w postaci pojedynczej wartości etykiety.

3.3.2. Przekształcenie RDBMS na AGDS

Poniżej przedstawiona została prosta transformacja Tabeli *Pacjenci* o kluczu głównym PESEL (rys. 3.1).



Rys. 3.1 Przekształcenie RDBMS do AGDS – Tabela *Pacjent (źródło: opracowanie własne)*

Można zaobserwować, że wykonywanie operacji na zwykłej tabeli jest stosunkowo kosztowne, gdzie przeszukiwanie powyższej nieposortowanej tabeli ma koszt *O(n)*, natomiast tabeli posortowanej *O(log n)*, co daje całkowity koszt (sortowanie+przeszukanie) na poziomie *O(n log n)* [7]. Kolejnym ograniczeniem jest możliwość sortowania tylko po jednej kolumnie. Dane w tabeli dodawane są w sposób chronologiczny (kolejności wkładania), wiec nawet sortowanie po jednym atrybucie wymaga indeksowania. Można również zauważyć występowanie redundantnych danych, które dodatkowo zabierają miejsce w pamięci.

Utworzenie struktury AGDS pozwala na zachowanie wszystkich danych oraz uzyskanie dodatkowych informacji poprzez ich łączenie relacjami następstwa. Na tak wykonanym grafie czas przeszukiwania jest często stały i wynosi *O(1), przy czym zależne jest to od rodzaju wyszukiwania, jak również ilości wartości różnych*. Istnieją różne sposoby definiowania grafu i są zależne od ilości kolumn składających się na

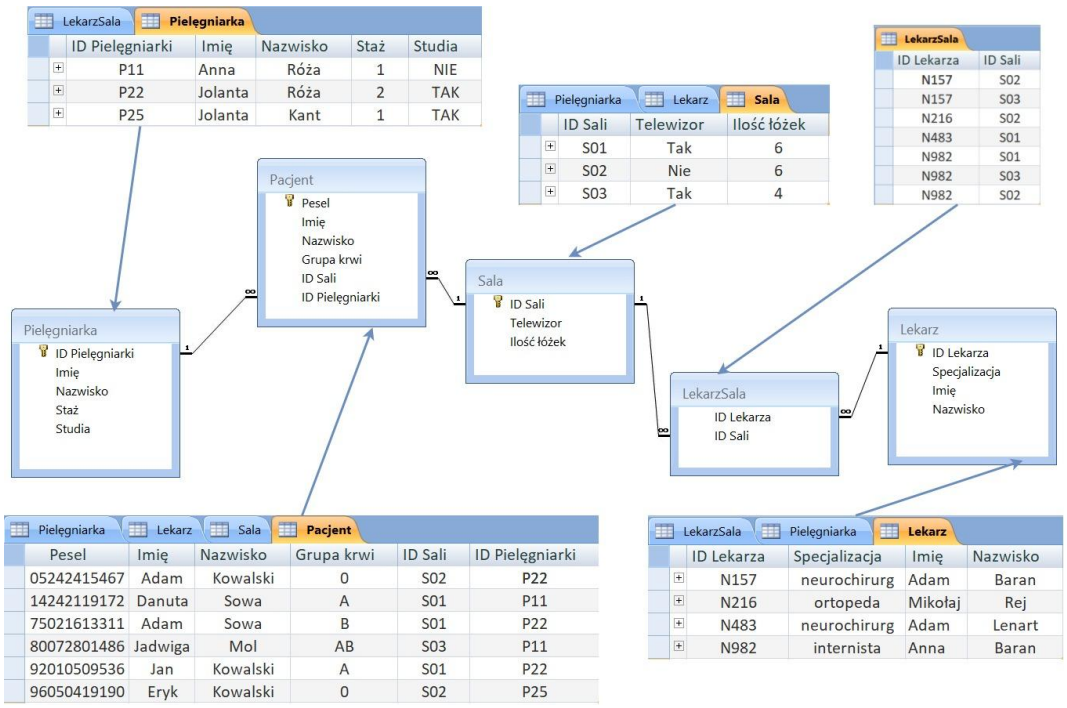
klucz główny. W powyższym przypadku nie wykonano włączenia klucza głównego (PESEL) do grafu, jako parametru ze względu na unikalność danych, lecz wykorzystano go jako nazwę węzła identyfikującego poszczególne osoby. Asocjacyjny graf AGDS umożliwia również eksplorację danych w zakresie utworzonych danych. Poprzez możliwość zastosowania wag można wyszukiwać rekordy „podobne" względem wybranego kryterium. Struktury asocjacyjne powinny być oceniane głównie przez pryzmat operacji wyszukujących i kompresujących dane. Na podstawie rysunku 3.1 można zauważyć większe skomplikowanie struktury oraz wyciągnąć wnioski na temat potencjalnego tworzenia takiej struktury. Początkowy narzut wykorzystany na budowę grafu zwraca się z nawiązką podczas późniejszego wykorzystania AGDS do wnioskowania.

Przykład przedstawiony powyżej nie wykorzystuje pełnego potencjału struktur asocjacyjnych. Pełne możliwości można zaobserwować w trakcie przetwarzania większych zbiorów w ramach relacyjnych baz danych. Poniższy przykład przedstawia proces tworzenia grafu na podstawie bazy danych „Oddział szpitalny", która składa się z kilku tabel:

Baza danych Oddział Szpitalny:

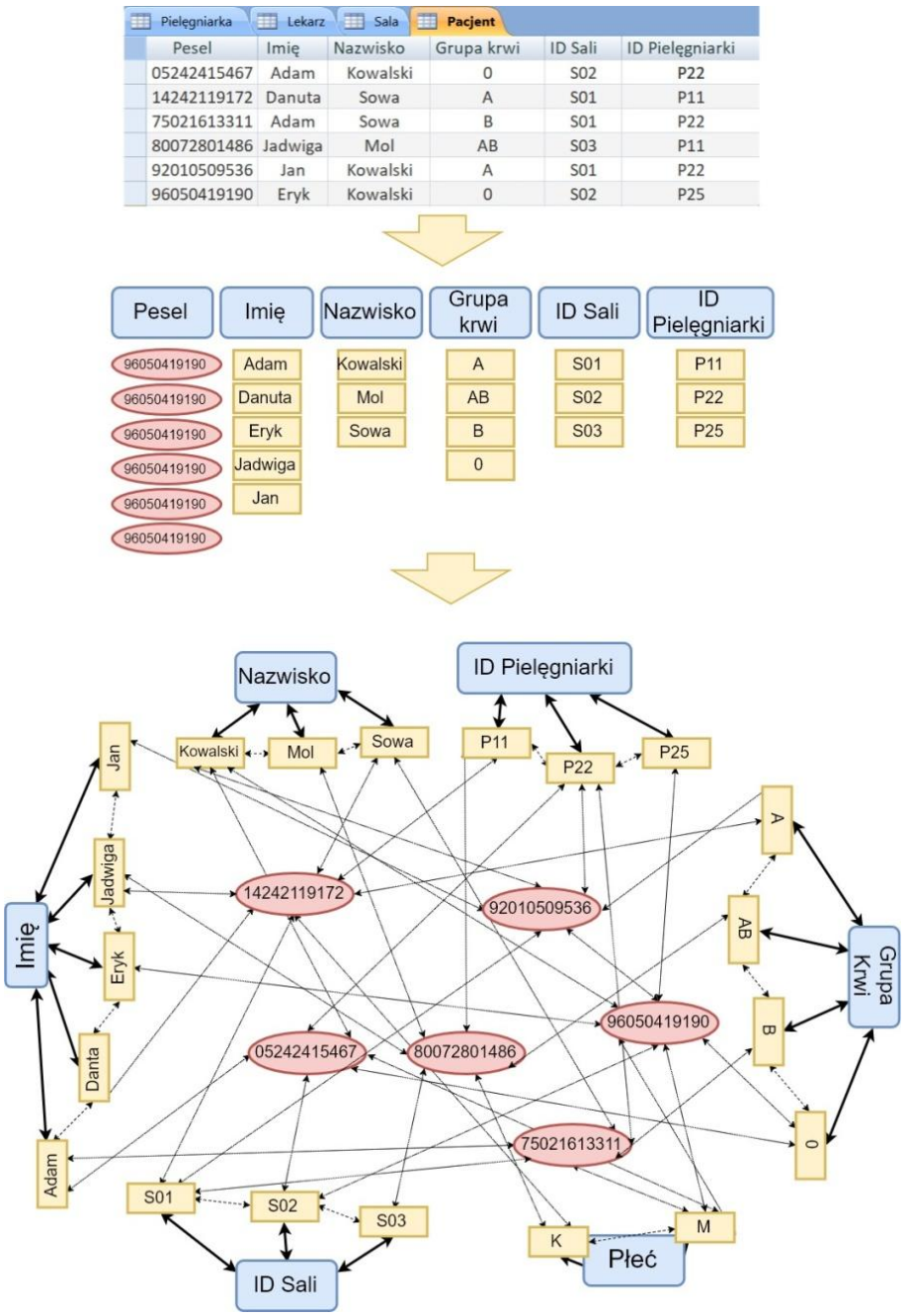
- Pacjenci
- Lekarze
- Pielęgniarki
- Sale

Przedstawiona w przykładzie baza danych (rys. 3.2) opisuje wewnętrzny oddział szpitalny, który składa się z sal (Sala) o określonych cechach. Na każdej sali leżą pacjenci, z których każdy ma przypisaną do siebie jedną pielęgniarkę. Dodatkowo dyżury na oddziale odbywają lekarze, którzy są przypisani do wielu sal w zależności od dyżuru.



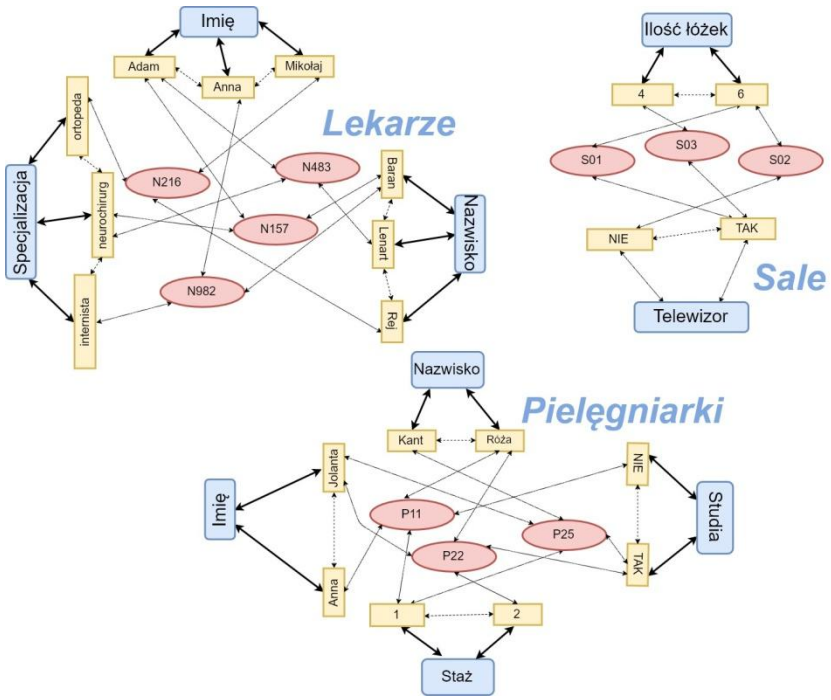
Rys. 3.2 Relacja bazy danych *Oddział Szpitalny (źródło: opracowanie własne)*

Proces przetworzenia przedstawiony zostanie na tabeli *Pacjenci*. Pierwszym krokiem jest wydzielenie z każdego atrybutu wszystkich wartości bez duplikatów. Kolejnym krokiem jest posortowanie wartości w każdej kolumnie względem wybranego wcześniej kryterium, w tym wypadku, wykonano sortowanie alfabetycznie. Następnie na podstawie kluczy głównych tworzy się referencje do rekordów i odwzorowywane są odpowiednie połączenia na podstawie wartości w tabeli. Powiązania te są typu EDEF [1] i są one dwustronne, co umożliwia wyszukiwanie rozpoczynając od parametru jak również od referencji do rekordu. Ostatnim krokiem jest połączenie ze sobą sąsiadujących wartości w obrębie parametru (powiązania typu ESIM), które umożliwią szybkie obliczenia, wyznaczanie wartości granicznych lub wnioskowanie i określanie podobieństwa.



Rys. 3.3 Struktura grafu AGDS po przekształceniu tabeli *Pacjent (źródło: opracowanie własne)*

Wiedząc jak przekształcać pojedyncze tabele, można przystąpić do budowania złożonej struktury AGDS na podstawie relacyjnej bazy danych. Pierwszym krokiem jest wydzielenie wszystkich tabel wirtualnych (składających się wyłącznie z kluczy obcych). Najlepszą metodą jest wyodrębnienie tabel, które nie posiadają kluczy obcych i od nich rozpocząć tworzenie grafu. Następnie należy stopniowo dołączać tabele posiadające klucze obce (rozpoczynając od tych, które zostały już wcześniej przetworzone na struktury AGDS) (*rys. 3.4)*.



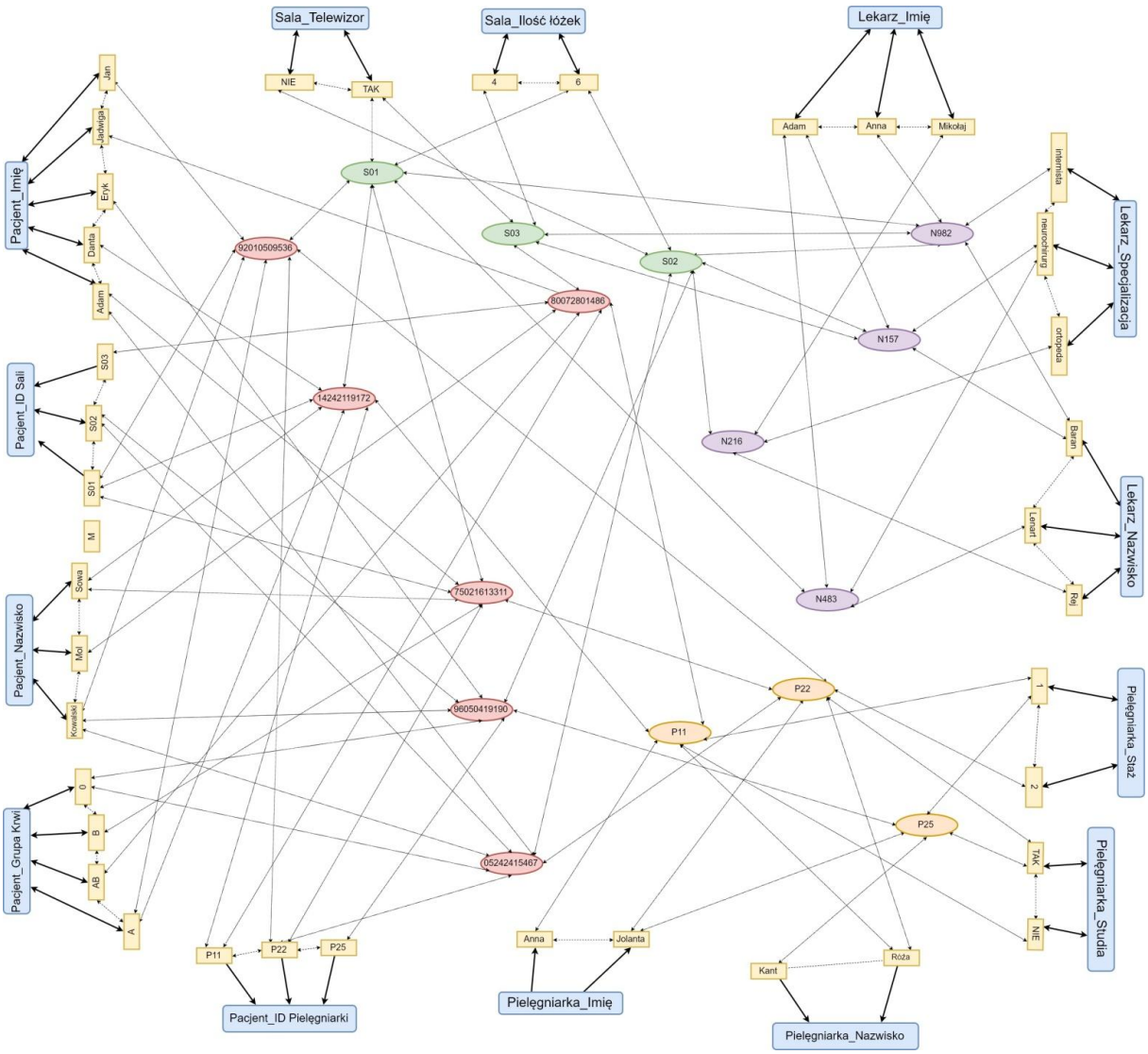
Rys. 3.3 Rozłączne grafy AGDS – Baza *Oddział Szpitalny (źródło: opracowanie własne)*

Kolejnym krokiem jest wyodrębnienie relacji pomiędzy tabelami na podstawie kluczy obcych w tabelach (uwzględniając tabele wirtualne - *vrt*) (*tabela 3.1.)*

Tabela 3.1. Relacje pomiędzy tabelami w bazie *Oddział Szpitalny*

Tabela źródłowa	Klucz obcy	Tabela docelowa	Atrybut docelowy
Pacjent	ID Pielęgniarki	Pielęgniarka	ID Pielęgniarki
Pacjent	ID Sali	Sala	ID Sali
LekarzSala (vrt)	ID Sali	Sala	ID Sali
LekarzSala (vrt)	ID Lekarza	Lekarz	ID Lekarza

Na podstawie wartości w tabelach i znanych powiązań Klucz Obcy-Atrybut docelowy można połączyć ze sobą powiązane rekordy. Należy pamiętać o kolejności przetwarzania tabel rozpoczynając od tabel prostych i stopniowo zmniejszać liczbę tabel do przetworzenia poprzez dołączanie kolejnych tabel złożonych, których odnośnik w postaci tabeli prostej został już przekonwertowany na graf AGDS. Powiązania powinny być typu EDEF ze względu na możliwości dwustronnego przemieszczania po grafie pomiędzy różnymi „częściami grafu". Posiadając już połączenia pomiędzy tabelami rzeczywistymi, można przetworzyć tabelę wirtualną. Jest to typ tabeli, która jest niezbędna do definiowania relacji wiele do wielu (rozdział 2). Należy przejść po tabeli, łącząc ze sobą odpowiadające sobie referencje do rekordów. Dla tabeli wirtualnych nie tworzy się grafu AGDS ze względu na brak kluczy głównych, które umożliwiłyby powiązanie wartości z konkretną referencją (*rys. 3.4).*



Rys. 3.3 Pełny graf AGDS – Baza *Oddział Szpitalny (źródło: opracowanie własne)*

Dla poniższego grafu wykonano też przykładowe zapytanie SQL z koniecznością wykorzystania łączenia wewnętrznego tabel (*join)*:

SELECT *pac.Pesel,*
pac.Imię,
pac.Nazwisko,
piel.Imię,
piel.Nazwisko,
FROM *Pacjent as pac* ***INNER JOIN*** *Pielęgniarka as piel*
ON *Pac.Pielęgnierka_ID = Piel.Pielęgniarka_ID*
WHERE *pac.Grupa_Krwi LIKE (‘A’, ‘AB’)*

Pierwszym krokiem jest przetworzenie warunku WHERE i wstępna selekcja rekordów, które spełniają warunek. Szukamy pacjentów, którzy mają grupę krwi *‘A’* lub *‘AB’*. Wychodzimy od parametru Pacjent_Grupa_Krwi i wyszukujemy referencje do rekordów, do których prowadzą wartości parametru z warunku. Wynikowo otrzymano 3 pacjentów (dwóch dla grupy A i jeden dla grupy AB). Kolejnym krokiem

jest odczytanie wartości parametrów z klauzuli SELECT, które odnoszą się do tabeli Pacjent (tabela 3.2).

Tabela 3.2. Klauzula SELECT tabeli *Pacjent*

Pacjent		
Pesel	Imię	Nazwisko
14242119172	Danuta	Sowa
80072801486	Jadwiga	Mol
92010509536	Jan	Kowalski

Następnym krokiem jest wyszukanie referencji do rekordów tabeli Pielęgniarka, które są połączone bezpośrednio z rekordami tabeli Pacjent. Następnie poprzez te rekordy odczytano wartości z parametrów dla pielęgniarek, a następnie połączono z wyszukanymi już pacjentami (tabela 3.3).

Tabela 3.3. JOIN tabel *Pacjent* oraz *Pielęgniarka*

SELECT pac.Pesel, pac.Imię, pac.Nazwisko, piel.Imię, piel.Nazwisko, FROM Pacjent as pac INNER JOIN Pielęgniarka as piel ON Pac.Pielęgnierka_ID = Piel.Pielęgniarka_ID WHERE pac.Grupa_Krwi LIKE (‘A’, ‘AB’)				
Pacjent Pesel	Pacjent Imię	Pacjent Nazwisko	Pielęgniarka Imię	Pielęgniarka Naz.
14242119172	Danuta	Sowa	Anna	Róża
80072801486	Jadwiga	Mol	Anna	Róża
92010509536	Jan	Kowalski	Jolanta	Kant

Wykonanie zapytania zawierającego klauzulę *JOIN* wymagało bardzo małej ilości przejść pomiędzy grafami. Odczyt wszystkich informacji odbywał się poprzez pojedyncze przejście pomiędzy węzłami. Przy niewielkich strukturach danych jak powyżej graf AGDS może wydawać się skomplikowany ze względu na dużą ilość połączeń i asocjacji. Należy jednak zwrócić uwagę na korzyści, jakie dzięki niemu otrzymujemy. Przetwarzanie skomplikowanych zapytań *SQL* przez grafy w bardzo krótkim czasie jest tylko jedną z wielu zalet. Moglibyśmy wnioskować podobieństwa pomiędzy rekordami lub w prosty sposób wyszukiwać wartości graniczne. Graf AGDS jest bardzo dobrym sposobem na kompresję danych i ich kombinację. Pomimo wielu zalet struktury AGDS posiadają pewne ograniczenia, które dyskwalifikują ich zastosowanie w pewnym przypadkach. Do największych wad należy działanie w pamięci RAM, co zmusza użytkownika do każdorazowego odtwarzania struktury przy uruchamianiu aplikacji. Drugą wadą, która została również omówiona na początku jest niezmienność danych. Każda modyfikacja, która nie jest przyrostowa wiąże się z koniecznością przebudowy grafu. Należy jednak wspomnieć, że operacje modyfikacji danych są i tak znacznie mniej kosztowne niż w przypadku indeksowanych atrybutów w relacyjnych bazach danych, ze względu na wykorzystanie struktur drzewiastych oraz szybkiej identyfikacji powiązanych rekordów. Kolejna sekcja opisuje główną metodę na optymalizację struktury grafowej poprzez zmniejszenie narzutu obliczeniowego przy wyszukiwaniu odpowiednich wartości parametrów. Do tej pory w omawianych przykładach

stosowana była posortowana lista wartości, której przeszukiwanie wykonywane było w czasie *O(n)*. W niniejszej pracy zaproponowano zastąpienie takiej listy poprzez strukturę drzewiastą, a konkretnie AVB-Drzewa , specjalną odmianę B-Drzew, , które są podstawą do stworzenia grafu DASNG będącego rozszerzeniem AGDS.

3.4. B-Drzewa jako metoda optymalizacji AGDS

B-Drzewa (*drzewa m-narne*) są strukturą drzewiastą, która jest uogólnieniem drzew binarnych. Jej głównym celem była początkowo optymalizacja dostępu do pamięci o dostępie bezpośrednim (np. dysków magnetycznych). B-Drzewo zaprojektowane zostało w latach sześćdziesiątych XX wieku przez Bayera i McCreighta [1]. Obecnie znajdują szerokie zastosowanie w bazach danych ze względu na szybkość przeszukiwania, poprzez niewielką (logarytmiczną) ilość operacji dostępu do danych. B-Drzewa są uogólnieniem drzew binarnych poprzez wykorzystanie struktury węzła wewnętrznego, który pozwala na przechowywanie większej liczby kluczy w węźle (w przypadku drzew binarnych przypadał jeden klucz na węzeł) [8].

3.4.1. B-Drzewa kontra inne struktury drzewiaste

Struktura B-Drzew ma swoje początki wywodzące się od drzew poszukiwań binarnych (BST). Zbudowane są one z węzłów, z których każdy posiada swój klucz wewnętrzny. Wymaganiem stawianym przed kluczem jest zdolność porównywania go z innymi w obrębie wybranego kontekstu. Drzewa BST umożliwiają przeszukiwanie danych w czasie wprost proporcjonalnym do wysokości drzewa, co przy n węzłach w najgorszym przypadku umożliwia przeszukiwanie z logarytmiczną złożonością obliczeniową *O(log n),* co w porównaniu do list, których złożoność wynosi *O(n)* jest dużym przeskokiem. Główną wadą struktur BST był brak reakcji na nadchodzące dane, co niekiedy powodowało budowę drzewa, o dużej wysokości (w pesymistycznym przypadku wysokość drzewa wynosiła n, co implikowało złożoność obliczeniową operacji przeszukiwania na poziomie *O(n)) [8].*

W związku z wadami związanymi z brakiem samoorganizacji i stale rosnącą wysokością drzewa rozpoczęto prace badawcze nad bardziej „zrównoważoną" strukturą. Stworzone zostały drzewa czerwono-czarne (RBT), które umożliwiają kontrolę wysokości drzewa poprzez wewnętrzną samoorganizację pod wpływem przychodzących danych. Drzewa RBT posiadają taką samą złożoność przeszukiwania (*O(log n)),* przy czym znacząco ograniczają wysokość drzewa, co uniemożliwia osiąganie wartości granicznych wyszukiwania.

Jedną z głównych różnic w budowie węzła w stosunku do BST jest występujący dodatkowy bit informacji, który imituje kolor czerwony/czarny. Dzięki wewnętrznym restrykcjom dotyczącym położenia kolorów zapewniony został warunek, iż każda ścieżka w grafie może być co najwyżej dwukrotnie dłuższa niż każda inna. Umożliwia to pewne utrzymanie zrównoważonego drzewa. Operacja „rotacji" (równoważenia wewnętrznej struktury) jest wykonywana po przyjściu jednostki danych, a przed umieszczeniem jej w docelowym drzewie. Na podstawie

przeprowadzonych badań udowodniono, że drzewo RBT ma wysokość co najwyżej *2 log(n+1)*. Po za drzewami czerwono-czarnymi występują również inne wariacje jak 2-3 drzewa, bądź AVL drzewa, lecz nie zostały one omówione w niniejszej pracy [7][8], gdyż nie mają większego znaczenia z punktu widzenia realizacji tej pracy.

Główną motywacją wprowadzenia struktur B-Drzew była możliwość wykonywania efektywnych operacji na dyskach magnetycznych, z których odczyt był operacją kosztowną. Struktura B-Drzew jest uogólnieniem drzew czerwono-czarnych, ale posiadająca lepsze właściwości. Główną cechą charakterystyczną jest zmienna ilość kluczy przechowywanych w pojedynczym węźle. Częścią wspólną obu rodzajów struktur jest wysokość, która wynosi *O(log n)* (ze względu na zrównoważenie), przy czym ilość rozgałęzień wychodzących z pojedynczego węzła może być znacznie większa i ograniczona do stopnia drzewa, co implikuje znacznie mniejszą wysokość. Decyzja o przejściu do kolejnego węzła odbywa się na podstawie porównania kluczy wewnętrznych pamiętanych w danym węźle.

3.4.2. Definicja

B-Drzewo T jest drzewem o poniższych własnościach:
Każdy węzeł x ma następujące pola:
$n[x]$– liczba kluczy aktualnie pamiętanych węźle x porządku niemalejącym

$$key_1[x] \leq key_1[x] \leq \cdots \leq key_{n[x]}[x]$$

Jeśli x jest węzłem wewnętrznym to zawiera także $n[x] + 1$ wskaźników do swoich synów $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$. Liście nie posiadają synów, więc ich pola są niezdefiniowane.
Klucze $key_i[x]$ rozdzielają przedziały kluczy pamiętanych w poddrzewach, jeśli k_i jest dowolnym kluczem poddrzewa o korzeniu $c_i[x]$ to:

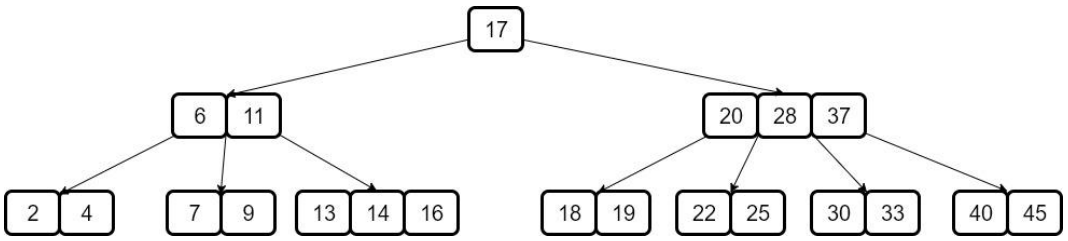
$$k_1 \leq key_1[x] < k_2 \leq key_2[x] < \cdots \leq key_{n[x]}[x] < k_{n[x]+1}$$

Wszystkie liście leżą na tej samej głębokości równej wysokości drzewa [8].

Istnieją dolne i górne ograniczenia na liczbę kluczy w danym węźle. Ograniczenia te zależą od ustalonej liczby całkowitej $t \geq 2$, którą nazywamy minimalnym stopniem drzewa:

- Każdy węzeł różny od korzenia musi mieć, co najmniej $t - 1$ kluczy, każdy węzeł wewnętrzny różny od korzenia ma, zatem t synów (w przypadku niepustego drzewa korzeń musi mieć, co najmniej jeden klucz).
- Każdy węzeł może zawierać, co najwyżej $2t - 1$ kluczy. Mówimy, że węzeł jest pełny, jeśli zawiera dokładnie $2t - 1$ kluczy.

Najprostszym drzewem jest 2-3-4 drzewo (wspomniane w sekcji 3.4.1), które posiada stopień $t = 2$ [8] (rys. 3.4).



Rys. 3.4 B-Drzewo 2-3-4 o głębokości 2

Wysokość drzewa jest kluczowym parametrem, który wpływa na szybkość przeszukiwania, zgodnie poniższym wzoremwysokość drzewa nigdy nie przekracza:

$$h \leq \log_t \frac{n+1}{2}$$

gdzie:
h – wysokość drzewa (głębokość liści),
t – minimalny stopień drzewa ($t \geq 2$),
n – liczba węzłów.

Na tej podstawie można zaobserwować korzyści, jakie płyną ze stosowania struktur B-Drzew. Przy porównaniu obu wysokości można stwierdzić, że wysokość B-Drzewa jest w przybliżeniu wprost proporcjonalnie mniejsza o log_t w stosunku do RBT [7].

3.4.3. Operacje na B-Drzewach

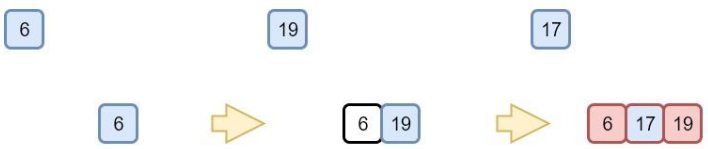
B-Drzewo umożliwia przeprowadzenie podstawowych operacji takich jak:

- Wstawianie (Insert)
- Wyszukiwanie (Search)
- Usuwanie (Remove)

Wstawianie

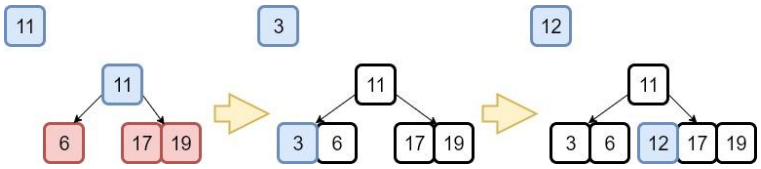
Wstawianie nowej wartości do B-Drzewa odbywa się w czasie jednego przebiegu, czyli będzie wymagać maksymalnie $O(t\ log_t n)$ operacji (na podstawie maksymalnej wysokości drzewa). Proces wstawiania odbywa się rekurencyjnie, a drzewo rośnie w dół to znaczy, że kolejne wartości wstawiane są w liściach na samym dole drzewa. Przebieg operacji wstawiania przedstawiony został na podstawie przykładu wstawiania liczb, gdzie ustalono kontekst, jako porównanie standardowe.

Przykład 3.1
Rozważano puste 2-3-4 drzewo, do którego należało wstawić następujące liczby:
6, 19, 17, 11, 3, 12, 8, 20, 22, 23, 21, 26



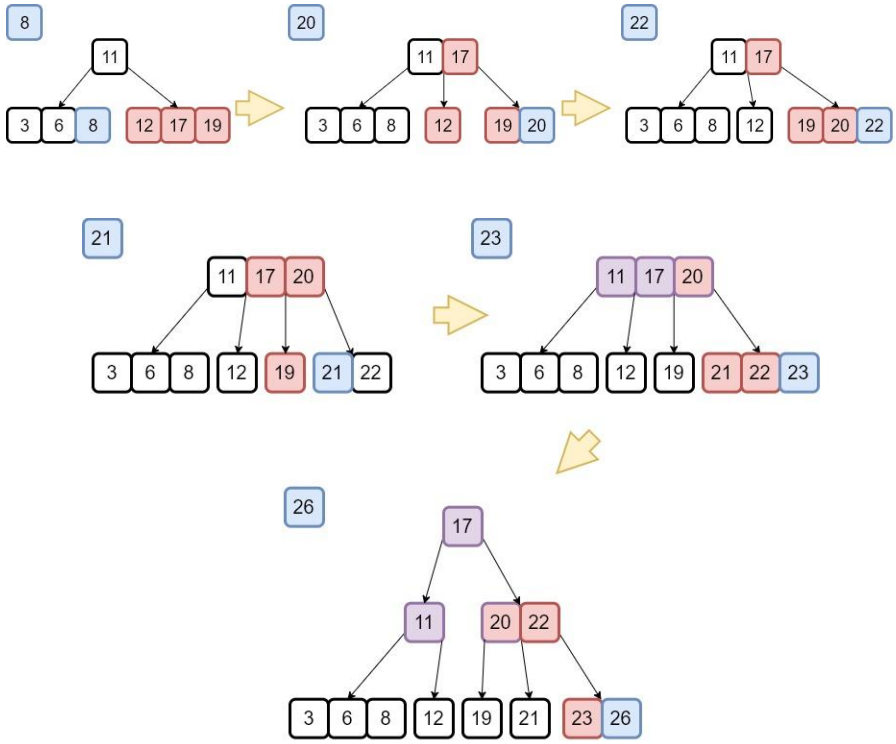
Rys. 3.5 Operacje wstawiania do B-Drzewa (6, 19, 17)

Rozważane B-Drzewo posiada minimalny stopień 2, co znaczy, że ilość dzieci wychodzących z poszczególnego węzła nie może przekroczyć 4. W przypadku dodania do pustego drzewa liczb 6,19, 17 są one dodawane do korzenia.



Rys. 3.6 Operacja rozdzielenia węzłów

Przy wstawianiu liczby 11 można zaobserwować przepełnienie węzła wewnętrznego. Powoduje to podzielenie węzła na mniejsze części. Gwarantuje to, że w przypadku zejść rekurencyjnych do kolejnych węzłów nigdy nie napotkamy węzła pełnego. Węzły uczestniczące w operacji zostały oznaczone kolorem czerwonym.



Rys. 3.7 Operacje wstawiania wraz z operacją podwójnego rozdzielenia

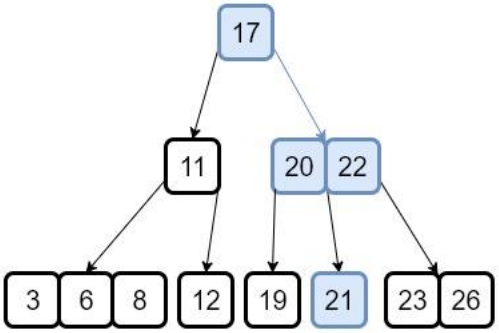
W trakcie dodawania liczby 26 można zaobserwować dwukrotny podział rekurencyjny. W pierwszym kroku rozbijany jest węzeł [21,22,23], z którego w górę drzewa przechodzi wartość 22 (kolor czerwony), która powoduje przepełnienie węzła wyższego i wykonanie kolejnego rozdzielenia węzłów (kolor fioletowy)

Wyszukiwanie

Operacja wyszukiwania danych w B-Drzewie jest analogiczna do standardowej operacji wyszukiwania w drzewie binarnym, przy czym różnica polega na tym, że

w każdym węźle dokonywany jest wybór poddrzewa (kierunku), w którym nastąpią dalsze poszukiwania w zależności od rezultatu porównania wewnątrz węzła.

Przykład 3.2
Znaleźć wartość 21 w B-Drzewie z przykładu 3.1



Rys. 3.8 Wyszukiwanie w B-Drzewie

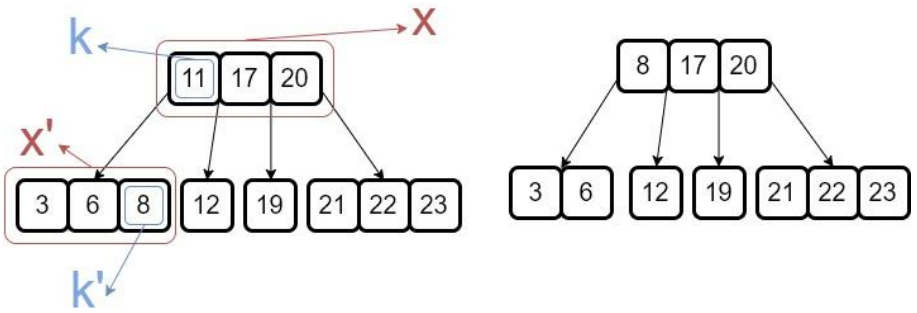
Można zaobserwować, że operacja wyszukania wymagała wykonania dwóch operacji porównania i dwóch przejść, co w porównaniu z listą daje wynik znacznie szybciej.

Usuwanie

Operacja usuwania jest podobna do operacji wstawiania i również realizowana przy pomocy rekurencji. Poniżej przedstawiony został algorytm postępowania:

- Jeśli usuwany klucz k jest w węźle *x* i *x* jest liściem, to usuwamy klucz *k*.
- Jeśli usuwany klucz należy do węzła x i x jest węzłem wewnętrznym, to:
    - *x*' niech będzie synem *x*, poprzedzającym *k*. Jeśli *x'* posiada co najmniej *t* kluczy, to należy wyznaczyć poprzednika klucza *k* jako *k'*. Następnie przesuwamy klucz *k'* w miejsce klucza *k*.

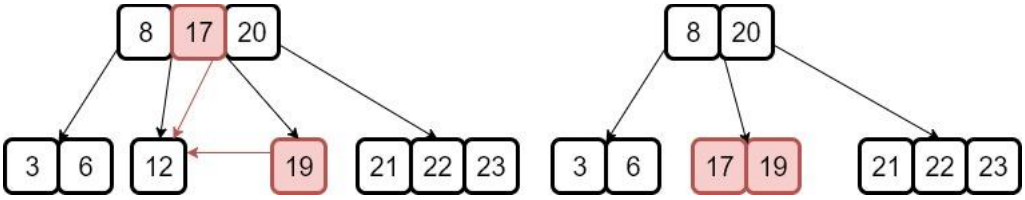
Przykład 3.3
Usunąć 11



Rys. 3.9 Usuwanie z liścia

- Jeśli węzeł będący poprzednikiem nie posiada co najmniej t kluczy sprawdzenie wykonuje się na węźle, który jest następnikiem. W przypadku spełnienia warunku stosujemy analogiczną operację jak w poprzednim punkcie, przy czym przenosimy klucz będący następnikiem usuwanego klucza.
- Jeśli obaj synowie *y* będący poprzednikiem i *z* będący następnikiem mają *t-1* kluczy, to należy wykonać operację łączenia. Należy przenieść cały węzeł będący następnikiem i wraz z kluczem *k* wpisać w węźle *y*, a następnie rekurencyjnie usunąć klucz *k* z węzła *y*.

Przykład 3.4
Usunąć 12



Rys. 3.10 Usuwanie z operacją łączenia

3.5. AVB Drzewa rozszerzeniem B-Drzew

AVB-Drzewa należą do struktur drzewiastych i zostały zaprojektowane przez dr. hab. Adriana Horzyka [2]. Są one naturalnym uogólnieniem B-Drzew specjalnie zaprojektowanych dla asocjacyjnych struktur grafowych. AVB-Drzewa można wyrazić jako B-Drzewa zawierające tylko unikalne wartości. Dodatkowo struktura automatycznie agreguje i zlicza pojawiające się duplikaty, co jest wykorzystywane przy zapytaniach agregujących, takich jak *COUNT*. Przechowywane informacje o duplikatach nie zmniejszają ilości informacji dotyczącej węzła, natomiast umożliwiają w późniejszym czasie jego poprawne usunięcie. AVB-Drzewa są zwykle mniejsze niż B-Drzewa oraz B+Drzewa [2][8], które nie agregują duplikatów ze względu na zwiększoną liczbę połączeń pomiędzy węzłem, a docelowymi referencjami do obiektów. Przypisanie do pojedynczej wartości znacząco przyspiesza odwołanie podczas wykonywania zapytań, co jest głównym celem tej struktury.

3.5.1. Operacje na AVB-Drzewach

Operacje wykonywane na AVB-Drzewach są bardzo podobne do standardowych wykonywanych na B-Drzewach. W przypadku przeszukiwania wykorzystywany jest standardowy mechanizm omówiony w poprzedniej sekcji, natomiast operacjami, które wymagają dodatkowego omówienia jest wstawianie oraz usuwanie elementów. Procesy te zostały omówione w postaci opisowej. Na koniec sekcji omówione zostaną główne różnice pomiędzy obiema zaproponowanymi strukturami [2].

Wstawianie

Operacja wstawiania jest bardzo podobna do tej, która wykonywana jest na B-Drzewach, przy czym uwzględnia ona dodatkowo zliczanie duplikatów, co umożliwi w późniejszym czasie efektywne usuwanie wartości bez utraty informacji. Algorytm zostanie omówiony na podstawie AVB-Drzewa o stopniu 2 (maksymalnie 2 klucze w węźle oraz maksymalnie 3 synów).

Wstawianie klucza *k* należy rozpocząć od korzenia, a następnie rekurencyjnie schodzić w dół zgodnie z następującymi zasadami [2]:

- Jeśli *k* jest mniejszy niż lewy klucz rodzica, należy przejść do lewego syna;
- Jeśli *k* jest większy niż prawy klucz rodzica, należy przejść do prawego syna;
- Jeśli *k* znajduje się pomiędzy dwoma ww. kluczami, to należy przejść do syna środkowego;
- Jeśli *k* jest równy jednemu z kluczy w węźle, to zwiększyć licznik tego klucza i zakończyć operację;

W przypadku osiągnięcia liścia, gdy węzeł będący liściem nie posiada maksymalnej liczby kluczy, to należy dodać *k* do liścia.

Jeśli w węźle-liściu jest już przechowywana maksymalna liczba kluczy, to należy zasymulować dodanie klucza *k* do węzła, a następnie podzielić węzeł na pół w taki sposób, że środkowa wartość trafia jako klucz do rodzica natomiast węzły lewy oraz prawy stają się automatycznie jego lewym i prawym synem.

W przypadku, gdy w rodzicu, do którego dodawany jest środkowy klucz, istnieje już maksymalna liczba kluczy, to należy powtórzyć operacje, analogicznie do punktu 3. Operacja dzielenia wykonywana jest do skutku.

W przypadku, gdy w liściu istnieje już dodawany klucz *k*, to należy zwiększyć licznik [2].

Jak można zaobserwować zaprezentowane ścieżki wstawiania dla AVB-Drzew, jak również B-Drzew są podobne, a różnica polega głównie na istnieju licznika kluczy, które się powtarzają. Obydwa rodzaje drzew dbają o zachowanie balansu w drzewie.

Usuwanie

Usuwanie w AVB Drzewie odbywa się analogicznie jak w przypadku B-Drzew, przy czym uwzględniane są dodatkowo liczniki duplikatów, co powoduje jedynie dekrementację licznika, przy zachowaniu obecnej struktury. Dopiero usunięcie

ostatniej istniejącej wartości klucza (ostatniego duplikatu wartości) uruchamia mechanizm usuwania analogiczny do B-Drzew.

3.5.2. AVB Drzewa w strukturze AGDS

Po wprowadzeniu do B-Drzew oraz AVB-Drzew można przystąpić do zamiany użytych list ww. struktury. Wykonanie takiej transformacji jest obarczone pewnym początkowym kosztem, który przy dużej ilości danych zwraca się z nawiązką. Kluczową kwestią jest wybranie stopnia drzewa. Przekształcenie relacyjnych baz danych z użyciem grafów AGDS oraz struktury AVB-Drzew i wykorzystanie neuronów asocjacyjnych ASN w miejsce węzłów prowadzi do powstania głębokiego asocjacyjnego grafu neuronowego (DASNG), który zostanie omówiony w kolejnym podrozdziale. Graf ten przy zaimplementowanych dodatkowo wagach umożliwia nie tylko przeszukiwania horyzontalne, ale również wertykalne. Dzięki agregacji duplikatów i wagowego połączenia pomiędzy węzłami można definiować własne zapytania oparte o podobieństwo, gdzie np. dla wcześniej rozważanej bazy danych Oddziału specjalistycznego możliwe byłoby zdefiniowanie zapytania:

Przykład 3.5
Znajdź mi wszystkich lekarzy, którzy mają dyżury w podobnych salach.

Każda wartość byłaby powiązana wagą, np. „sala żółta jest podobna do pomarańczowej, a podobnieństwo to jest wyrażone przez wagę x”.

3.6. Głęboki asocjacyjny graf neuronowy (DASNG)

Niniejsza sekcja opisuje głęboki asocjacyjny graf neuronowy, który jest rozszerzeniem AGDS poprzez zastosowanie współczynnika czasu i utworzenie struktury neuronowej. Struktura ta nie kompensuje informacji o przetwarzanych relacjach. Graf DASNG [2] może być skonstruowany z dowolnej struktury bazodanowej na podstawie relacji klucz podstawowy – klucz obcy, które opisują semantyczne podobieństwo pomiędzy rekordami. Dzięki powyższej konstrukcji możliwe jest tworzenie relacji podobieństwa nie tylko ze względu na bezpośrednie połączenia, lecz również w oparciu o podobne cechy pomiędzy połączeniami pośrednimi. Umożliwia to znaczne rozszerzenie funkcjonalności w obrębie metod inżynierii wiedzy, jak również sztucznej inteligencji. W rezultacie użytkownik może dowolnie zdefiniować dowolny podobny zbiór wartości, który zostanie odtworzony w strukturze grafowej. W konsekwencji każda klasa obiektów może zostać szybko odnaleziona w DASNG, ponieważ poprzez stymulację podgrupy węzłów powodują one aktywację podobnych neuronów najbardziej spełniających zadane relacje i ograniczenia.

Model grafu DASNG może być również reprezentowany przez inne relacje, oparte na czasie lub współrzędnych, gdzie dwa porównywane obiekty znajdują się w niewielkiej odległości od siebie lub obiekty występują w niedalekim odstępie czasowym. W głębokim asocjacyjnym grafie możliwe jest również zastosowanie

współczynnika wagowego, który pozwoli na jednoznaczność powiązań w zależności od wybranego kontekstu. W przypadku modelu DASNG neurony są połączone z cechami, do których się odnoszą oraz innymi neuronami podobnymi. Dzięki zastosowaniu dodatkowo omówionego wcześniej czynnika czasowego możliwe jest określenie siły relacji, co również dostarcza informacji o sile powiązania klucz obcy-klucz podstawowy w określonym kontekście.

Zastosowanie sensorów reaktywnych oraz neuronów zamiast biernych rejestrów bazy danych jak to zostało zrobione w przypadku zwykłego grafu AGDS pozwala na szybkie i automatyczne uzyskiwanie informacji w kontekście stymulowanym przez wybrany podzbiór neuronów. Przy budowie grafu DASNG nie ma tworzenia węzłów dla tabel złączeniowych, ponieważ są one wykorzystywane jedynie do połączenia już istniejących części grafu ze sobą. W przypadku neuronów reprezentujących wartości liczbowe tworzone jest pomiędzy nimi połączenie zgodnie ze zdefiniowaną regułą sortującą, dzięki czemu nie ma potrzeby sortowania danych po zakończeniu budowy grafu [1][2].

Agregacja, która jest ogromną zaletą omawianych struktur może w pewnych przypadkach okazać się wadą, kiedy wymagamy, aby rekordy o dokładnie tym samym zestawie wartości były pomiędzy sobą rozróżnialne. Rozwiązaniem wykorzystanym w grafie DASNG jest wyodrębnienie w neuronach klucza będącego unikalnym identyfikatorem rekordu. Wartość taka może zostać przedstawiona jako funkcja analityczna, której nie można zredukować podczas agregacji. Taka strategia wymagana jest w przypadku odpytywania grafu na podstawie jego unikalnego identyfikatora. Jako przykład może posłużyć potrzeba wyszukania w grafie informacji o fakturze z konkretnym numerem. Grafy DASNG oprócz agregacji danych również wykorzystują AVB-Drzewa, które zostały omówione w poprzednim podrozdziale jako specjalna dedykowana struktura danych, przy czym zamiast węzłów przechowują one sensory sieci DASNG. Pozwala to na pominięcie sortowania w procesie wstawiania nowych sensorów i neuronów [2][5].

Dzięki obustronnym połączeniom użytkownik posiada szybki i bezpośredni dostęp do cech reprezentowanego obiektu. Możliwe jest dzięki temu zdefiniowanie klastrów, które będą wyrażać cechy podobne dla dowolnych kryteriów. Poprzez stymulację podzbioru cech podobnych, ich zakresu lub dowolnego podzbioru obiektów następuje stopniowa indukcja oraz aktywowanie połączonych neuronów związanych z obiektami powiązanymi. Każda stymulacja grafu DASNG ma stały czas, co jest dużą przewagą w porównaniu do podobnych tego typu metod [1][2].

4. Implementacja

Niniejszy rozdział opisuje implementację algorytmu przekształcającego relacyjną bazę danych do postaci struktury grafowej AGDS. Aplikacja wykonana została na platformie .NET przy użyciu języka C#. Część implementacyjna została podzielona na pięć wynikających z siebie części:

- implementacja warstwy danych,
- implementacja struktury grafowej AGDS,
- implementacja mechanizmu serializacji zapytań SQL.

Całość wykonana została zgodnie z najnowszymi standardami z wykorzystaniem elementów programowania obiektowego oraz wzorców projektowych.

4.1. Warstwa danych

Warstwa danych aplikacji służy do unifikacji danych wejściowych na postać odpowiednią do dalszego przetwarzania przez algorytm. Zaprojektowany mechanizm oparty został o jeden z najpopularniejszych wzorców projektowych wykorzystywany do połączenia z bazą danych – *fasada*. W tym celu zaprojektowano dedykowany interfejs udostępniający niezbędne metody odczytu. W przedstawionej aplikacji uwzględniono tylko bazę danych SQL Server. Do odczytu surowych danych i parametrów bazy wykorzystano standardowy mechanizm z przestrzeni nazw *System.Data* oraz dodatkowo wykorzystano zewnętrzną bibliotekę *SQL Server Management Object (SMO)*, która jest oficjalną biblioteką wspieraną przez Microsoft. Motywacją do jej zastosowania była lepsza obsługa informacji o kluczach obcych w RDBMS. Projekt warstwy danych nie przewidywał modyfikowania danych, a jedynie ich odczyt. Zaproponowana hierarchia klas została przedstawiona poniżej:

DatabaseGateway

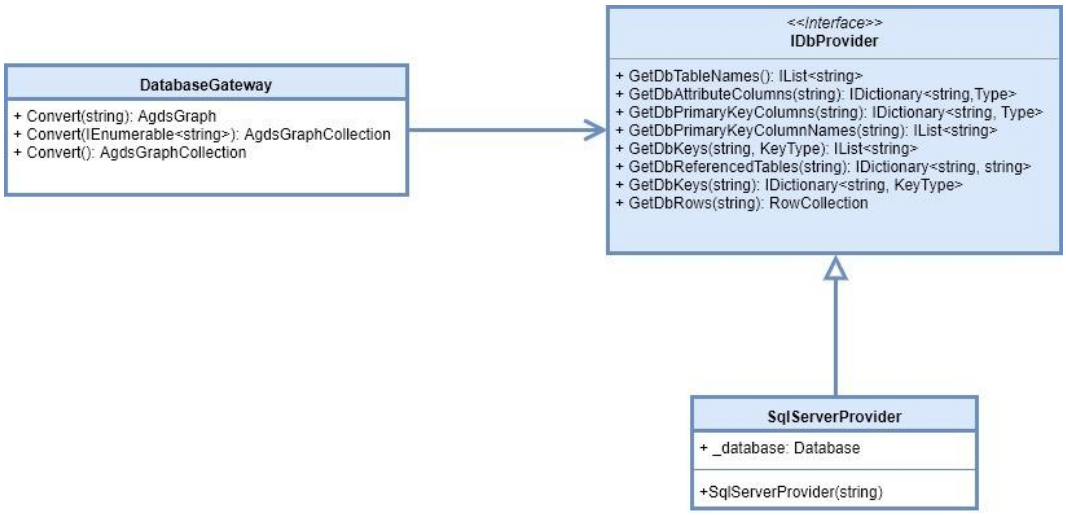
Klasa DatabaseGataway jest najważniejszą istotną częścią aplikacji. Odpowiada ona za połączenie z bazą danych oraz za konwersję do struktury *AgdsGraph* oraz *AgdsGraphCollection*. Zdefiniowane są w niej metody *Convert*, które w zależności od przyjętych parametrów zwracają odpowiednie struktury. Motywacją stworzenia powyższej klasy było ukrycie implementacji bazy danych, jak również uniemożliwienie użytkownikowi odczytywania wartości z tabel. Zapobiega to wprowadzaniu zmian z zewnątrz oraz gwarantuje spójność strukturom AGDS.

Rola klasy *DatabaseGateway* ogranicza się do tworzenia grafów AGDS na żądanie, jak również wywoływania zapytań SQL zwracających całkiem nowe

niezależne instancje obiektów, co uniemożliwia zmiany w strukturze. Przy projektowaniu klasy wykorzystano wzorzec projektowy Fasada, który uniemożliwia korzystanie z bazy danych z pominięciem wyżej wymienionej klasy [10].

IDbProvider

Kluczową częścią powyższej struktury jest interfejs *IDbProvider*(*rys 4.1)*, który udostępnia kluczowe funkcje odczytu danych i parametrów z bazy. W tabeli 4.1 zostało omówione ich zastosowanie. Klasa *SqlServerProvider* implementuje ten interfejs przekształcając dane zwracane przez bazę danych SQL Server.



Rys. 4.1 Architektura warstwy danych

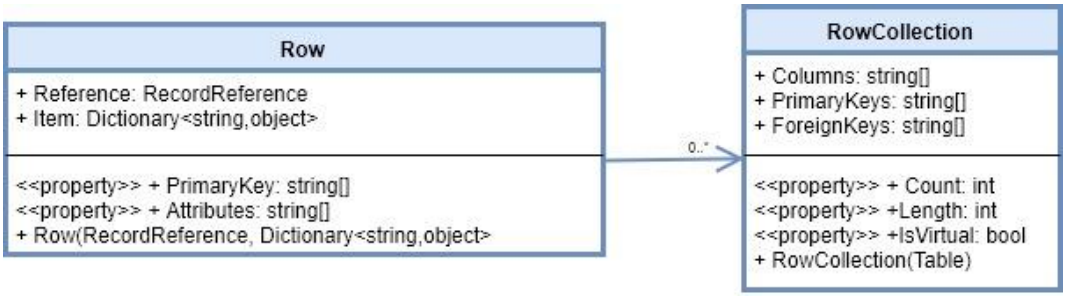
Tabela 4.1. Metody *IDbProvider*

IDbProvider	
GetDbTableNames()	Funkcja zwraca listę tabel dostępnych w rozważanej bazie danych
GetDbAttributeColumns(string table)	Funkcja zwraca listę kolumn (bez kluczy głównych) na podstawie *table* w postaci <*nazwa_kolumny; typ_kolumny*>
GetDbPrimaryKeyColumns(string table)	Funkcja zwraca listę kluczy głównych tabeli w postaci <*nazwa_kolumny; typ_kolumny*>
GetDbPrimaryKeyColumnNames(string table)	Funkcja zwraca listę nazw kluczy
GetDbKeys(string table, KeyType keyType)	Zwracana jest lista kluczy w zależności od parametru *keyType* (PrimaryKey/ForeignKey)
GetDbReferencedTables(string table);	Funkcja zwraca listę tabel powiązanych z *table* w postaci słownika o budowie <*tabela_powiązana;powiązanie*>, gdzie *powiązanie* jest typem string o budowie – „*kluczObcyTable_columnaTabelaPowiązana*”
GetDbKeys(string table)	Zwraca listę kluczy w postaci słownika z uwzględniem typu jakiego jest klucz <*nazwa klucza;typ klucza*>
GetDbRows(string table)	Zwraca rekordy w postaci *RowCollection*

RowCollection i Row

Kolejnym krokiem, który należało zrealizować, była konwersja standardowej kolekcji *System.Data.DataRowCollection* na strukturę posiadającą wbudowane informacje o kolumnach. Została zaprojektowana klasa *RowCollection* (rys 4.2), która była obiektem typu DTO (Data Table Object), który przetwarzał dostarczoną tabelę i udostępniał dane tylko do odczytu. Należy zwrócić uwagę na właściwość *IsVirtual*, której zadaniem było określenie, czy wszystkie atrybuty tabeli są kluczami obcymi,

własność ta będzie miała kluczowe znaczenie przy budowaniu złożonej struktury AGDS łączącej wiele tabel. Jako pojedynczy rekord wykorzystano strukturę *Row*, która poza przechowywaniem danych o rekordzie tworzyła również jego „zdegenerowaną postać", która następnie wykorzystywana była w zbudowanej strukturze AGDS jako referencja.

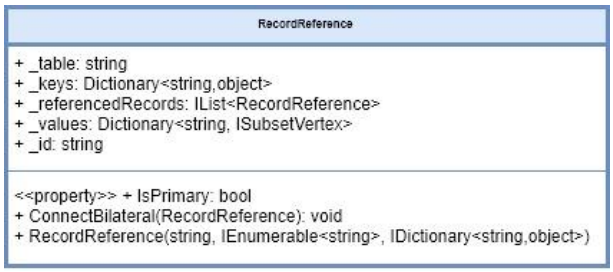


Rys. 4.2 Projekt klas typu DTO Row i RowCollection

Powyższe klasy nie zawierają logiki biznesowej, a służą jedynie do odczytu wartości i istnieją tylko w czasie tworzenia grafu AGDS. Po przetworzeniu obiekty są usuwane, a w dalszym wyszukiwaniu i przetwarzaniu wykorzystywane są już tylko obiekty typu *RecordReference*.

RecordReference

Projekt klasy *RecordReference* był kluczowy w dalszej implementacji, ponieważ od jego postaci zależał proces przebiegu algorytmu (Rys 4.3).



Rys. 4.3 Referencja do rekordu *RecordReference*

W klasie *RecordReference* wykorzystano strukturę typu HashTable, jaką jest *Dictionary*. Są to struktury niezwykle efektywne przy odczycie danych na podstawie klucza, co będzie miało ogromne znaczenie w późniejszym działaniu struktury AGDS przy przeszukiwaniu. Każdy „zdegenerowany rekord" składa się z dwóch słowników, z których jeden zawiera pary klucz-wartość, natomiast drugi składa się z nazwy kolumny oraz wskaźnika do obiektu *ISubsetVertex* (jest to węzeł zawierający wartość dla danego parametru, szersze omówienie znajduje się w sekcji „Implementacja Struktury Grafowej AGDS). Na listingu 4.1 i 4.2 przedstawione zostały kluczowe sekcje kodu (konstruktor wraz z indekserem).

Na listingu 4.1 znajduje się implementacja indeksera, który zwraca wartość w zależności od podanego atrybutu. Przy odczycie sprawdzany jest warunek klucza, co implikuje obiekt, z którego ma zostać odczytana wartość (słownik

atrybutów/słownik kluczy). W przypadku podania nieistniejącej wartości atrybutu zwracana jest wartość NULL.

Listing 4.1 ***RecordReference Indexer***

```
public object this[string attr]
{
  get {
    try {
      if (IsPrimary(attr))
        return _keys[attr];
      else
        return _values[attr];
    }
    catch { return null; }
  }
}
```

W konstruktorze (Listing 4.2) oprócz tworzenia czystej referencji (bez wartości atrybutów, tylko klucze) tworzony jest również identyfikator, który pozwoli na identyfikację referencji do rekordu na podstawie tekstu i znajdzie zastosowanie przy wzajemnym łączeniu rekordów na podstawie kluczy obcych.

Listing 4.2 ***RecordReference Constructor***

```
public RecordReference(string table, IEnumerable<string> primaryKey,
                       IDictionary<string,object> rowVal) {
  _keys = new Dictionary<string,
                         object>(StringComparer.InvariantCultureIgnoreCase);
  _referencedRecords = new List<RecordReference>();
  _values = new Dictionary<string, ISubsetVertex>();
  _table = table;

  string id = String.Format("[{0}]", table);
  foreach (var key in primaryKey) {
    _keys.Add(key, rowVal[key]);
    id += String.Format("[{0}_{1}]", key, rowVal[key]);
  }
  _id = id;
//Przykład:
//table: "Tabela1" ; primaryKey: {id1, id2} ;
//rowVal: {id1=1;id2=5;obj=object}
//=> ID: [Table1][id1_1][id2_2]
}
```

Dzięki powyższej implementacji warstwy danych udało się uzyskać spójny kod o dużej rozszerzalności. W przyszłej implementacji możliwe będzie bezstratne dodanie nowej bazy danych, jak np. Oracle lub PostgreSQL. Ograniczeniem, które zastosowano w poniższej implementacji jest możliwość przetwarzania jednocześnie pojedynczej bazy danych. Wykorzystanie interfejsów w powyższych implementacjach znacznie ułatwia rozszerzalność, a napisany kod zgodny jest z obowiązującymi praktykami.

4.2. Struktura Grafowa AGDS

Kolejnym etapem implementacji było utworzenie Asocjacyjnej Grafowej Struktury Danych AGDS. Implementacja została podzielona na dwie części, które są od siebie zależne:

- implementacja grafu *AgdsGraph*,
- implementacja kolekcji grafowej *AgdsGraphCollection*.

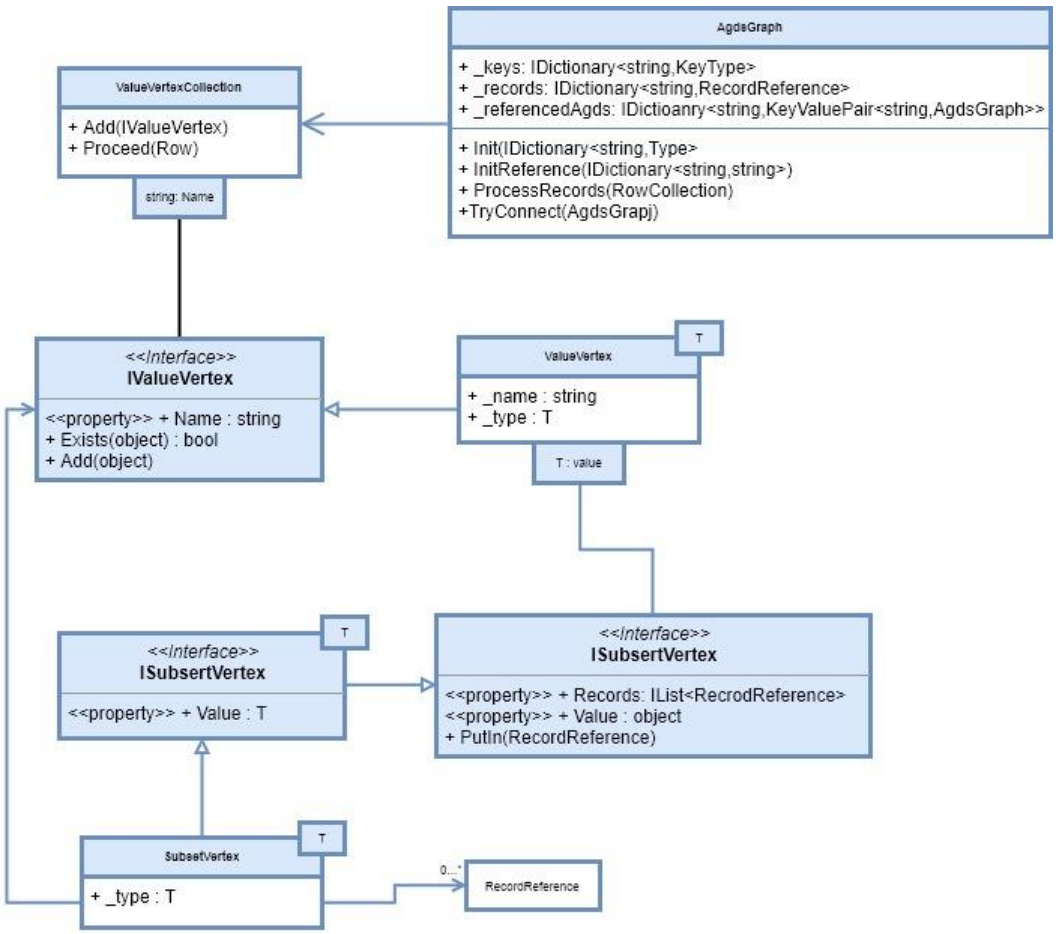
Struktura *AgdsGraph* wyraża graf utworzony na podstawie pojedynczej tabeli. W przypadku wystąpienia w tabeli kluczy informacje o nich zapisywane są wewnątrz grafu, co w późniejszym czasie pozwoli na połączeniu grafu z innym. Pierwszym krokiem jest wydzielenie rodzajów tabel mogących wystąpić w powyższej realizacji:

- tabela prosta – tabela nie posiadająca kluczy obcych,
- tabela złożona – tabela zawierająca klucze obce,
- tabela wirtualna (tabela złączeniowa) – składa się wyłącznie z kluczy obcych.

Dla każdej z wymienionych wyżej tabel otrzymamy inną postać struktury AGDS. W pierwszym etapie omówiona została klasa *AgdsGraph*, która jest główną składową klasy *AgdsGraphCollection*.

AgdsGraph

Klasa *AgdsGraph*, jak zostało wcześniej podkreślone, odnosi się do pojedynczej tabeli. Przy implementacji zwracano uwagę na zasadę „pojedynczej odpowiedzialności", co doprowadziło do uzależnienia jedynie od klasy *RowCollection*. W późniejszych przypadkach możliwe będzie rozszerzenie danych wejściowych, np. o plik XML. Tak jak zostało to zaznaczone w poprzednich sekcjach, duże znaczenie przywiązano do podstawowych struktur danych, które miały największy wpływ na działanie algorytmu. Wykorzystano pola typu *IDictionary*, które są niezwykle szybkie do odczytu [9]. Kolejnymi obiektami składowymi klasy *AgdsGraph* jest obiekt typu *ValueVertexCollection*, który jest odpowiednikiem zbiorem wierzchołków *ValueVertex* omówionych w sekcji dotyczącej Asocjacyjnych Struktur. W kolekcji tej zagnieżdżone są obiekty *SubsetVertex*, które są konkretnymi wartościami przypisanymi do parametru. W poniżej hierarchii (Rys. 4.3) uwzględnione zostały w postaci skrótów klasy z poprzedniej sekcji.



Rys. 4.3 Hierarchia dla kasy AgdsGraph

Kolejnym krokiem jest omówienie dwóch najważniejszych składowych *ValueVertex* oraz *SubsetVertex*.

ValueVertexCollection i ValueVertex<T>

Motywacją utworzenia klasy *ValueVertexCollection* była potrzeba dodania dodatkowych funkcjonalności obejmujących cały zbiór ValueVertex. Umożliwiło to dodatkowo odwoływanie się do konkretnych parametrów na podstawie polecenia *ValueVertexCollection[name]*, gdzie *ValueVertexCollection* jest w tym przypadku konkretną instancją obiektu.

Klasa ta zastosowaniem pokrywa się z definicją węzła ValueVertex omówioną w rozdziale teoretycznym dotyczącym struktur AGDS. Przechowuje ona nazwę parametru wraz z referencjami do wszystkich wartości, które w ramach niego wystąpiły. Klasa *ValueVertex* jest klasą generyczną, ponieważ każdy z przetwarzanych typów bazodanowych może być dowolnego typu. Do konwersji typów bazodanowych utworzono statyczną klasę *SqlConverter*, która pobierała typ kolumny z bazy danych i zwracała typ z przestrzeni nazwy *System*. Konwersja odbywała się na poziomie *IDbProvider*, aby nie przenosić typów SQL w głąb aplikacji. Problemem, który wystąpił podczas tworzenia obiektów była statyczność typów. Język C# posiada statyczną kontrolę typów, co oznacza, że dokładny typ każdego obiektu musi być znany na etapie kompilacji [9]. Komplikowało to etap tworzenia parametrów, ponieważ z góry nie były znane typy parametrów, z których będzie składać się

przetwarzana tabela. Listing 4.3 przedstawia proces tworzenia zbioru parametrów *ValuVertex<T>*, gdzie *T* jest typem kolumny SQL przekonwertowanej na *System.Type* poprzez użycie metody *MakeGeneric* z przestrzeni nazw *System* [dokumentacja Microsoft]. Umożliwiała ona tworzenie obiektów danego typu w czasie działania aplikacji. Można zauważyć, że najpierw tworzony jest właściwy typ, np. *ValueVertex<int>*, a następnie na tym obiekcie wywoływany jest konstruktor [9].

Listing 4.3 Inicjalizacja *AgdsGraph*

```
public void Init(IDictionary<string,Type> columns) {
  if (_valueVertices != null)
    throw new InvalidOperationException("VertexCollection can't be assigned twice");

  _valueVertices = new ValueVertexCollection();
  foreach (var column in columns) {
    Type type = typeof(ValueVertex<>).MakeGenericType(new[] { column.Value });
    _valueVertices.Add((IValueVertex)Activator.CreateInstance(type, column.Key));
  }
}
```

Po zainicjalizowaniu pustych węzłów ValueVertex (posiadały jedynie nazwę i typ) można było przystąpić do przetwarzania rekordów i wstawiania ich w konkretne miejsca. Można przyjąć jedną z dwóch ścieżek przetwarzania – pierwsza z nich przypisuje wartości na podstawie kolumn, druga natomiast na podstawie wierszy. Ze względu na specyfikę danych otrzymywanych z serwera SQL (wartości rekordu miały postać object[]) konieczne byłoby dodatkowe przejście przez dwuwymiarową tablicę. Funkcja, która realizowała przypisanie realizowana była poprzez wywołanie metody *ValueVertexCollection.Proceed(Row)* (Listing 4.4).

Dzięki zastosowaniu indekserów oraz słowników można szybko odwoływać się do konkretnych wartości (przykład w listingu). Zastosowanie ww. funkcji *MakeGeneric* umożliwiło wstawianie wartości o zdefiniowanym typie *T* zamiast *object* (Rekordy zawierają tylko wartości typu object). Metoda przechodzi po wszystkich wartościach atrybutów (bez kluczy podstawowych) i sprawdza, czy w węźle ValueVertex istnieje już dana wartość rekordu, jeśli tak, to dla tej wartości dodawana jest referencja do rekordu, w przeciwnym wypadku nowa wartość jest dodawana w zbiorze *SubsertVertex<T>*, a następnie do tej wartości przypisywana jest referencja do rekordu.

Listing 4.4 *ValueVertexCollection Proceed*

```
public void Proceed(Row row) {
  foreach (string c in row.Attributes) {
    object val = row[c];
    if (Vertices[c].Exists(val))
      Vertices[c][val].PutIn(row.Reference);
    else {
      Vertices[c].Add(row[c]);
      Vertices[c][val].PutIn(row.Reference);
    }
  }

//Przykład:
```

```
//Row : {Name=Jan ; City=Kraków ; Age=19}
//Vertices[Name][Jan] => Row
}
```

SubsetVertex<T>

Funkcjonalność klasy *SubsetVertex<T>* pokrywa się z definicją przedstawioną w teoretycznym omówieniu AGDS. Przechowuje ona listę rekordów powiązanych wartością danego parametru. Dodatkowo dzięki zdefiniowanemu polu *Parent* umożliwia wsteczne poruszanie się po grafie rozpoczynając od konkretnego rekordu kończąc na wierzchołku grafu. Klasa ta implementuje interfejs *IComparable<T>* dzięki czemu można w prosty sposób porównywać typy na podstawie typu *T*, a nie poprzez *object [9]*.

AgdsGraphCollection

Jest to kolekcja, która jest zbiorem elementów *AgdsGraph*. Poprzez wykorzystanie *AgdsGraphCollection* możliwe jest tworzenie złożonych grafów AGDS składających się z wielu zależnych (niezależnych) od siebie tabel. Proces tworzenia można podzielić ze względu na istnieje odniesień pomiędzy grafami:

- Istnieją referencje pomiędzy obiektami *AgdsGraph.*
- Obiekty są całkowicie niezależne.

W przypadku utworzenia kolekcji obiektów niezależnych nie będzie możliwe wykonywanie zapytań SQL typu JOIN. Hierarchia klasy została przedstawiona poniżej:

Rys. 4.4 Hierarchia dla kasy AgdsGraphCollection

Struktura *AgdsGraphCollection* może zostać stworzona na dwa sposoby:

- W postaci pustej struktury, do której dodawane są kolejne rekordy (obiekty są całkowicie niezależne).
- Na podstawie podanej bazy danych (automatycznie definiowane są zależności między rekordami).

Poniżej omówiona została metoda *TryConnect*, która odpowiada za utworzenie połączeń pomiędzy niezależnymi strukturami grafowymi. Przyjęte zostały dwie strategie połączeń – pierwsza dotyczyła tabel złożonych, gdzie połączenia wykonywane były w pętli 1 do 1, druga natomiast odnosiła się do tabel wirtualnych, gdzie połączenie odbywało się na podstawie trzech tabel (listing 4.5).

Listing 4.5 *TryConnect* (Dla całej bazy)

```
public void TryConnect() {
  foreach(var agds in AgdsCollection) {
    if (agds.IsSimple)
      continue;
    else if (!agds.IsVirtual) {
      foreach (var refAgds in agds.ReferencedAgdsNames)
        agds.TryConnect(this[refAgds]);
      }
    else
      agds.TryConnectVirtual(this);
  }
}
```

Listing 4.6 TryConnect Virtual

```
public void TryConnectVirtual(AgdsGraph agdsGraphSource, AgdsGraph agdsGraphDest) {
  string refKey, vrtAttr1, sourceAttr, vrtAttr2, destAttr;
  string[] keys;
  KeyValuePair<string, AgdsGraph> agdsKeyGraphSource, agdsKeyGraphDest;

  if (_referencedAgds.TryGetValue(agdsGraphSource.Name, out agdsKeyGraphSource)
      && _referencedAgds.TryGetValue(agdsGraphDest.Name, out agdsKeyGraphDest))
  {
    refKey = agdsKeyGraphSource.Key;
    keys = refKey.Split('_');
    vrtAttr1 = keys[0]; sourceAttr = keys[1];

    refKey = agdsKeyGraphDest.Key;
    keys = refKey.Split('_');
    vrtAttr2 = keys[0]; destAttr = keys[1];

    foreach (var rec in Records) {
      object sourceVal = rec[vrtAttr1];
      object destVal = rec[vrtAttr2];
      IList<RecordReference> sourceRecords = agdsGraphSource[sourceAttr, sourceVal];
      foreach(RecordReference sourceRec in sourceRecords)
        sourceRec.ConnectBilateral(agdsGraphDest[destAttr, destVal]);
    }

    agdsKeyGraphSource = new KeyValuePair<string, AgdsGraph>(agdsKeyGraphSource.Key,
                                                             agdsGraphSource);
    ReplaceReferencedAgds(agdsKeyGraphSource);

    agdsKeyGraphDest = new KeyValuePair<string, AgdsGraph>(agdsKeyGraphDest.Key,
                                                           agdsGraphDest);
    ReplaceReferencedAgds(agdsKeyGraphDest);
  }
}
```

Obie metody łączenia działają na podobnej zasadzie i sprowadzają się do odczytania Rekordów z obu grafów na podstawie odpowiednich kolumn, a następnie połączenia ich względem wartości określonych w tabeli łączenia. Proces ten został omówiony w części teoretycznej dotyczącej tworzenia grafów AGDS. Implementacja struktury grafowej oraz warstwy danych umożliwia już w pełni funkcjonalne korzystanie z grafu. Kolejne części dotyczyć będą parsowania języka zapytań SQL, co umożliwi efektywne porównanie otrzymywanych wyników poprzez definiowanie pojedynczego zapytania.

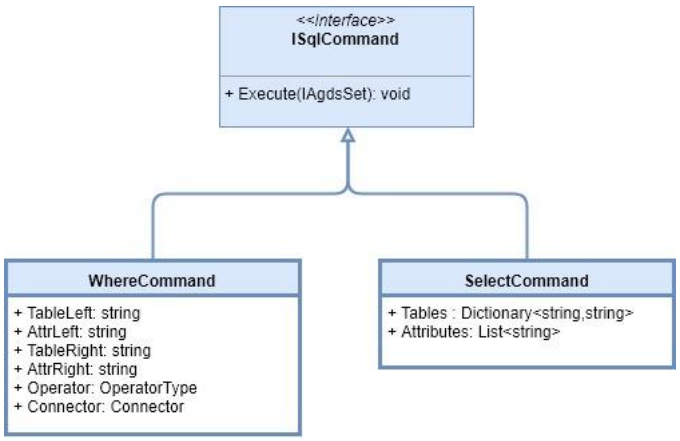
4.3. Parser SQL

Implementację parsera zapytań z języka SQL na strukturę grafową AGDS można podzielić na dwie główne części:

- Parsowanie zapytania SQL na postać funkcji dla grafu AGDS.
- Implementacja funkcji dla grafu AGDS.

Parsowanie języka zapytań SQL jest zagadnieniem skomplikowanym i wiąże się z elementami konstrukcji kompilatorów, co w znacznym stopniu wykracza po za zakres niniejszej pracy. W związku z tym podczas implementacji skorzystano z gotowej biblioteki znalezionej na *GitHubie*. Pozwalała ona na parsowanie zapytania SQL i pobierania kluczowych wartości, jak aliasy oraz nazwy wymaganych kolumn i używanych tabel. Wadą rozwiązania była jego prędkość jednak w przypadku samego parsera skupiono się na funkcjonalności. Biblioteka nazywała się *TSql Parser* i została napisana przez użytkownika *Bruce Dunwiddie***.**

Druga część implementacji odnosiła się do implementacji funkcji przetwarzających graf AGDS i na podstawie warunku *WHERE* oraz klauzuli *SELECT* zwracanie gotowych danych. W tym celu wykorzystano wzorzec projektowy *Polecenie* [9]. Wzorzec ten jest uznawany za jeden z najprostszych wzorców projektowych wyróżnianych w programowaniu obiektowym. Składa się na niego pojedynczy interfejs *ISqlCommand*, który zawiera jedną metodę *Execute (Rys. 4.5)*. Następnie stworzono implementację tego interfejsu w postaci klas *SelectCommand*, *JoinCommand* oraz *WhereCommand*. Główną ideą zastosowanego rozwiązania było utworzenie listy obiektów *ISqlCommand*, które przetwarzałyby części zapytania w kolejności dodawania.



Rys. 4.5 Hierarchia dla klas ISqlCommand

Proces przetwarzania opierał się na utworzeniu listy funkcji, które rozpoczynały się od klauzuli WHERE, gdzie następowała filtracja rekordów, a następnie były one wyświetlane na podstawie atrybutów określonych w funkcji SELECT.

Jak można zauważyć dodatkowo w klasie *WhereCommand* znajdują się pola Operator oraz Connector. Oznaczają one odpowiednio operator porównania wykorzystywany przy filtracji (zaimplementowano operatory <, >, =), natomiast connector oznacza operator logiczny, jaki łączy obecne zapytanie z kolejnym (None, AND, OR). Zaimplementowana klasa SELECT umożliwiała przetwarzanie danych do postaci przyjaznej dla użytkownika jednak w znacznym stopniu spowalniała przetwarzanie zapytania, niż w przypadku gdy wykorzystywane były tylko referencje do rekordów.

5. Opracowanie wyników

Po wykonaniu implementacji i przeprowadzeniu testów funkcjonalnych w celu stwierdzenia poprawności działania przystąpiono do realizacji badań porównawczych. Pierwsza część rozdziału dotyczyć będzie opisu instancji testowej oraz przebiegu prowadzonych badań z uwzględnieniem *„wąskich gardeł”*, czyli części systemu, które mają główny wpływ na czas przetwarzania:

- czas budowania grafu AGDS,
- porównanie czasu przeszukiwania grafu AGDS w stosunku do SQL Serwer oraz frameworka `Microsoft.SqlServer.Server`.

Na koniec omówione zostaną otrzymane rezultaty oraz potencjalne czynniki, które miały wpływ na przebieg badań.

5.1. Instancja testowa

Badania zostały przeprowadzone przy wykorzystaniu przykładowej bazy danych dostarczonej przez *Microsoft* jaką jest *AdventureWorks2012*. Powyższa instancja pozwoli zbudować w pełni funkcjonalny graf AGDS.

Opis:

Adventure Works Cycle jest fikcyjną firmą, na której oparta została powyższa baza. Jest to duża międzynarodowa firma, która zajmuje się produkcją i sprzedażą rowerów metalowych i kompozytowych na rynkach komercyjnych w Ameryce Północnej, Europie oraz Azji. Główna siedziba firmy znajduje się w Stanach Zjednoczonych w Bothell w stanie Waszyngton i zatrudnia 290 pracowników. Posiada również kilka zespołów sprzedażowych, które rozmieszczone są w całej bazie rynkowej.

Powyższa baza danych pozwala na przeprowadzenie kilku odrębnych scenariuszy:

- scenariusz sprzedaży i marketingu – opisuje środowisko sprzedaży, klientów oraz promocje;
- scenariusz produktu – opis produktów wytwarzanych przez firmę;
- scenariusz zakupu i dostawcy – opisuje zakup poszczególnych części oraz relacje pomiędzy firmą i dostawcami;
- scenariusz produkcji – Opisuje środowisko produkcyjne i proces technologiczny.

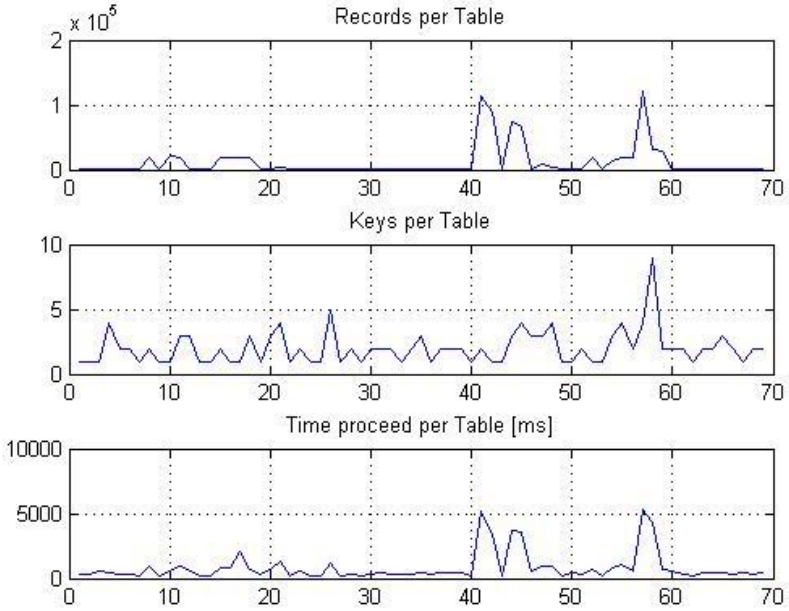
5.2. Czas budowania grafu AGDS

Czas budowy grafu AGDS na podstawie relacyjnej bazy danych, mimo że nie jest przedmiotem badań w niniejszej pracy, odgrywa niezwykle istotną rolę pod względem potencjalnego zastosowania przedstawionych implementacji w ramach projektów komercyjnych. Ze względu na istnienie grafu w czasie działania aplikacji (*runtime)* lub ewentualne odczytywanie jego budowy z pliku czas jego tworzenia jest niezwykle istotnym czynnikiem. Poniżej przedstawiono rezultaty badań dla kilku przypadków testowych. W rozważaniach brano pod uwagę liczbę kluczy obcych tabeli (mają one wpływ na ilość tworzonych referencji pomiędzy grafami AGDS), liczbę rekordów oraz liczbę atrybutów.

Scenariusz testowy 1:

W pierwszej części testu zbadany zostanie czas budowy odrębnych grafów AGDS ze wszystkich tabel występujących w bazie *AdventureWorks2012*. Wyznaczone zostaną przedziały czasowe dla każdej tabel z osobna z uwzględnieniem powyższych parametrów.

Do zobrazowania wyników wykorzystano środowisko *Matlab 2013b***.** Zgodnie z przewidywaniami, główny czynnikiem oddziałującym na czas przetwarzania była liczba rekordów w tabeli. Na uwagę zasługuje fakt, iż liczba kluczy (obcych i głównych) nie miała większego wpływu na otrzymywane czasy, co zostało zaprezentowane przy tabelach 48 i 57, gdzie liczba rekordów była porównywalna, a liczba kluczy była dwukrotnie większa (rys 5.1). Czas przetworzenia wszystkich tabel nie przekraczał jednej minuty.



Rys. 5.1 Czasy przetwarzania tabel

Scenariusz testowy 2:
Kolejnym krokiem było sprawdzenie, w jakim czasie zostaną połączone poszczególne grafy AGDS pomiędzy sobą. W tym celu uruchomiono proces wzajemnego łączenia wszystkich grafów. Podczas przetwarzania tabeli *Person.BusinessEntity* napotkano na pierwszy problem wydajnościowy. Tabela składała się z 20 777 rekordów i posiadała klucz obcy do samej siebie. Klucz obcy odpowiadał kluczowi głównemu tabeli, co implikowało unikalność wartości w grafie AGDS. W celu przetworzenia powyższej tabeli wymagane było 20 777 powiązań. Dla pozostałych tabel czasy utrzymywały się na akceptowalnym poziomie.

5.3. Czas budowania grafu AGDS

Kolejnym, a zarazem kluczowym badaniem w odniesieniu do niniejszej pracy było porównanie czasu przetwarzania zapytań SQL dla trzech różnych instancji:

- SQL Server 2012
- Grafy AGDS
- Microsoft.SQL.Server Framework

Pierwszym krokiem w trakcie badanie było określenie zbioru testowego. W tym celu przeanalizowano wszystkie tabele zawarte w bazie *AdventureWorks2012*, a następnie wybrano spośród nich optymalny zbiór tabel wraz z relacjami, aby możliwe było wykonywanie skomplikowanych zapytań z użyciem łączenia tabel JOIN.
Przed rozpoczęciem testów należało określić mierzony zakres wykonywanych działań na strukturach grafowych. Przeprowadzona została analiza typu zwracanego przez Microsoft.SQL.Server Framework, jakim jest *DataSet*. Obiekt ten obejmuje wszystkie informacje na temat bazy danych, jednak jego kluczową częścią jest dwuwymiarowa tablica *ItemArray*. Podjęto decyzję projektową o stworzeniu prostego obiektu, który imitowałby strukturę *DataSet*, lecz odnosiłby się do grafów AGDS. Pozwoliłoby to na zrównoważenie otrzymywanych wyników.

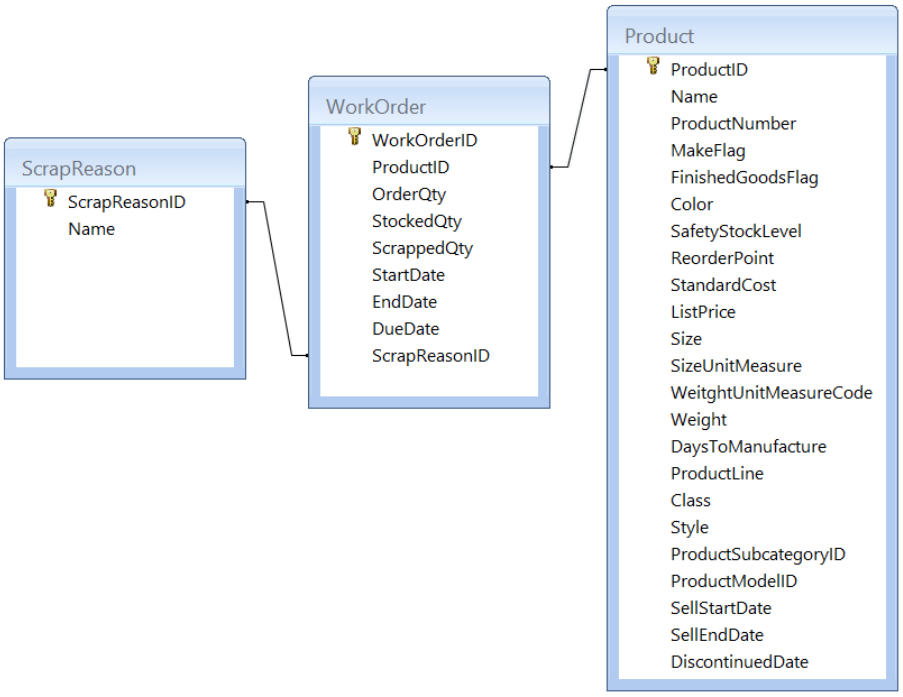
Porównanie czasów przeszukiwania wymaga ustalenia pewnych niezmienników (oprócz tych omówionych wcześniej), które zwiększyłyby wiarygodność wyników. Testy przeprowadzone zostaną w 10 iteracjach, z których po odrzuceniu wyniku najgorszego i najlepszego wyciągnięta zostanie średnia. Spowoduje to zwiększenie konkurencyjności relacyjnych baz danych, które posiadają mechanizm *cache'u* i podobne (te same) zapytania przetwarzają szybciej z każdą iteracją.

Ze względu na ograniczoną liczbę zaimplementowanych mechanizmów SQL w grafach AGDS testy ograniczą się do JOIN'ów standardowych (na podstawie kluczy obcych), a zapytania nie będą obejmować przeszukiwania złączonych tabel oraz prostych jednocześnie.

*Select * From AdventureWorks2012.Production.WorkOrder as wo*
JOIN AdventureWorks2012.Production.ScrapReason as sr
ON wo.ScrapReasonID = sr.ScrapReasonID,
~~*AdventureWorks2012.Production.Product*~~

Przypadek testowy:

Relacja, która została wybrana do testów odnosiła się do rzeczywistego scenariusza opartego na produkcji. W tym celu wybrano następującą relację:



Rys.5.2 Przypadek testowy – relacje

Poniżej przedstawione zostały informacje na temat powyższych tabel odczytane ze zbudowanego grafu AGDS

Tabela 5.1. Informacje – AGDS ScrapReason

ScrapReason			
Liczba rekordów		16	
Czas budowania grafu		232 ms	
Czas łączenia		0 ms	
Kolumna	Typ (System)	Liczba Unikalnych wartości	Rodzaj kolumny
ScrapReasonID	System.Int32	16	Klucz podstawowy
Name	System.String	16	Atrybut

Kluczem w powyższej tabeli jest *ScrapReasonID*, dlatego liczba unikalnych wartości wynosi 16. Czas łączenia grafu wyniósł *0ms*, ponieważ tabela nie posiadała żadnych kluczy obcych.

Tabela 5.2. Informacje – AGDS Product

Product			
Liczba rekordów		504	
Czas budowania grafu		2258 ms	
Czas łączenia		95 ms	
Kolumna	Typ (System)	Liczba Unikalnych wartości	Rodzaj kolumny
ProductID	System.Int32	504	Klucz podstawowy
Name	System.Object	504	Atrybut
MakeFlag	System.Object	504	Atrybut
FinishedGoodsFlag	System.Object	504	Atrybut
Color	System.String	9	Atrybut
SafetyStockLevel	System.Int16	6	Atrybut
ReorderPoint	System.Int16	6	Atrybut
StandardCost	System.Decimal	114	Atrybut
ListPrice	System.Decimal	103	Atrybut
Size	System.String	18	Atrybut
SizeUnitMeasureCode	System.String	1	Klucz obcy
WeightUnitMeasureCode	System.String	2	Klucz obcy
Weight	System.Decimal	127	Atrybut
DaysToManufacture	System.Int32	4	Atrybut
ProductLine	System.String	4	Atrybut
Class	System.String	3	Atrybut
ProductSubCategoryID	System.Int32	37	Klucz obcy
ProductModelID	System.Int32	119	Klucz obcy
SellStartDate	System.Date	4	Atrybut
SellEndDate	System.Date	2	Atrybut
DiscountedDate	System.Date	0	Atrybut

Czas przetwarzania i próby łączenia wynika z istnienia kluczy obcych w tabeli. Algorytm sprawdza, czy istnieją niezależne grafy o identyfikatorze określonym przez klucz obcy. Proces łączenia grafów zostanie przedstawiony przy tabeli *WorkOrder*. Różnica między czasami tworzenia jest ściśle powiązana z liczbą kolumn oraz rekordów w tabeli.

Tabela 5.3. Informacje – AGDS Product

WorkOrder			
Liczba rekordów		72591	
Czas budowania grafu		3313 ms	
Czas łączenia		45510 ms	
Kolumna	Typ (System)	Liczba Unikalnych wartości	Rodzaj kolumny
WorkOrderID	System.Int32	72591	Klucz podstawowy
ProductID	System.Int32	232	Klucz obcy

OrderQty	System.Int32	903	Atrybut
StockedQty	System.Int32	1013	Atrybut
ScrappedQty	System.Int16	87	Atrybut
StartDate	System.DateTime	1093	Atrybut
EndDate	System.DateTime	1093	Atrybut
DueDate	System.DateTime	1093	Atrybut
ScrapReasonID	System.Int16	16	Klucz obcy

Graf *WorkOrder* idealnie oddaje korzyści płynące z wykorzystania struktur AGDS. Dla około 70000 rekordów udało się zredukować liczbę trzymanych w pamięci unikalnych wartości dla kolumny *OrderQty* do 232 w czasie 3,13 sekundy. Dla powyższego grafu istnieją dwa klucze obce, których referencje do grafów istnieją, dlatego algorytm przeprowadził proces łączenia dla dwóch grafów (*Product, ScrapReason)* i trwał on około 45 sekund. Czas łączenia i przetwarzania grafu był ściśle związany z ilością unikalnych wartości dla grafu referencyjnego (w przypadku grafu *Product* wykonane zostało łączenie dla 232 wartości na podstawie kolumny *ProductID*).

Przeszukiwanie:

Wykonywane na zbiorze zapytania uszeregowano pod względem skomplikowania. Zwrócono również uwagę, aby każde kolejne było rozszerzeniem poprzedniego. Pozwoliłoby to na analizę wzrostu czasu odpowiedzi na podstawie dodatkowych ograniczeń oraz dodatkowych tabel. Pierwszym typem zapytań było pobieranie wartości z jednej tabeli i obserwacja czasów przetwarzania poprzez zmianę warunku *WHERE*.

Zapytanie 1:

```
SELECT wo.WorkOrderID, wo.OrderQty, wo.ScrappedQty
FROM Production.WorkOrder AS wo
WHERE wo.OrderQty > 100
```

Tabela 5.4. Rezultat – Zapytanie 1

AGDS	Microsoft.SqlServer	Microsoft SQL Server	
00:00:00.0303747	00:00:00.0351704	00:00:00.0943492	
00:00:00.0181001	00:00:00.0304608	00:00:00.1041748	
00:00:00.0164440	00:00:00.0410319	00:00:00.0882853	
00:00:00.0213440	00:00:00.0320031	00:00:00.0898707	
00:00:00.0176794	00:00:00.0501788	00:00:00.0814772	
00:00:00.0175265	00:00:00.0308298	00:00:00.0877583	
00:00:00.0188606	00:00:00.0386351	00:00:00.0820236	
00:00:00.0233754	00:00:00.0304664	00:00:00.0855246	
00:00:00.0176118	00:00:00.0395820	00:00:00.1067833	
00:00:00.0215462	00:00:00.0346686	00:00:00.0895176	
19 [ms]	39 [ms]	90 [ms]	**średnia [ms]**

Zapytanie 2:

```
SELECT wo.WorkOrderID, wo.OrderQty, wo.ScrappedQty
FROM Production.WorkOrder AS wo
WHERE wo.OrderQty > 100
AND wo.ScrappedQty = 4
```

Tabela 5.5. Rezultat – Zapytanie 2

AGDS	Microsoft.SqlServer	Microsoft SQL Server	
00:00:00.0129584	00:00:00.0376435	00:00:00.0758665	
00:00:00.0035974	00:00:00.0271162	00:00:00.0606357	
00:00:00.0055099	00:00:00.0365906	00:00:00.0621808	
00:00:00.0037858	00:00:00.0248158	00:00:00.0591566	
00:00:00.0036981	00:00:00.0334056	00:00:00.0581520	
00:00:00.0036578	00:00:00.0256948	00:00:00.0629286	
00:00:00.0050006	00:00:00.0373069	00:00:00.0544324	
00:00:00.0044776	00:00:00.0277250	00:00:00.0597879	
00:00:00.0037163	00:00:00.0325780	00:00:00.0592206	
00:00:00.0038893	00:00:00.0267164	00:00:00.0554446	
4 [ms]	30 [ms]	60 [ms]	**średnia [ms]**

Zapytanie 3:

```
SELECT wo.WorkOrderID, wo.OrderQty, wo.ScrappedQty
FROM Production.WorkOrder AS wo
WHERE wo.OrderQty > 100
AND wo.ScrappedQty = 4
AND wo.StockedQty > 170
```

Tabela 5.6. Rezultat – Zapytanie 3

AGDS	Microsoft.SqlServer	Microsoft SQL Server	
00:00:00.0182384	00:00:00.0454605	00:00:00.0504586	
00:00:00.0079367	00:00:00.0398929	00:00:00.0455028	
00:00:00.0069048	00:00:00.0271992	00:00:00.0409194	
00:00:00.0076341	00:00:00.0377809	00:00:00.0490150	
00:00:00.0075765	00:00:00.0259583	00:00:00.0450438	
00:00:00.0073382	00:00:00.0330734	00:00:00.0438709	
00:00:00.0076938	00:00:00.0281734	00:00:00.0457209	
00:00:00.0064007	00:00:00.0334139	00:00:00.0405046	
00:00:00.0090251	00:00:00.0259982	00:00:00.0445535	
00:00:00.0068302	00:00:00.0387753	00:00:00.0489624	
8 [ms]	33 [ms]	45 [ms]	**średnia [ms]**

Można zauważyć, że wraz ze wzrostem liczby warunków malał czas przetwarzania grafu AGDS. Przyczyną jest zmniejszająca się z każdym warunkiem liczba rekordów do dalszej analizy. Wpływ na czasy ma również liczba „unikalnych wartości" dla rozważanego węzła, ponieważ znacznie zmniejsza zakres do przeszukania. Otrzymane wyniki obrazują różnicę w szybkości, jednak nie wpływają na czas wykonywania zapytania, ponieważ dla każdego z przypadków jest on

znikomy. Prawdopodobnie graf AGDS działałby jeszcze z większą prędkością, gdyby zwracane dane nie były zapisywane w tablicy (analogicznej jak *System.Data.DataSet*). W następnym kroku przeprowadzono rzeczywiste testy ze złączeniami.

Zapytanie 4:

```
SELECT wo.WorkOrderID, wo.OrderQty, wo.ScrappedQty
From Production.WorkOrder as wo
JOIN Production.Product as pr
ON wo.ProductID = pr.ProductID
```

Tabela 5.7. Rezultat – Zapytanie 4

AGDS	Microsoft.SqlServer	Microsoft SQL Server	
00:00:00.6120516	00:00:01.0707785	00:00:01.4886003	
00:00:00.5714685	00:00:00.9752805	00:00:01.4011122	
00:00:00.6518790	00:00:01.0331417	00:00:01.3774021	
00:00:00.6108080	00:00:00.9531641	00:00:01.3884086	
00:00:00.6205313	00:00:00.9247947	00:00:01.4119029	
00:00:00.5330834	00:00:00.9896845	00:00:01.4067441	
00:00:00.5584042	00:00:00.9803306	00:00:01.4248305	
00:00:00.5156233	00:00:01.0121307	00:00:01.4256969	
00:00:00.5241120	00:00:00.9572716	00:00:01.4440681	
00:00:00.5891874	00:00:00.9733372	00:00:01.4022432	
578 [ms]	986 [ms]	1417 [ms]	średnia [ms]

Zapytanie 5:

```
SELECT wo.WorkOrderID, wo.OrderQty, wo.ScrappedQty, pr.Color
From Production.WorkOrder as wo
JOIN Production.Product as pr
ON wo.ProductID = pr.ProductID
WHERE pr.SafetyStockLevel > 600
AND pr.ProductNumber > 'FR'
```

Tabela 5.7. Rezultat – Zapytanie 5

AGDS	Microsoft.SqlServer	Microsoft SQL Server	
00:00:00.0362518	00:00:00.2123119	00:00:00.0667718	
00:00:00.0385564	00:00:00.2381249	00:00:00.0854891	
00:00:00.0272509	00:00:00.2758490	00:00:00.0685824	
00:00:00.0248592	00:00:00.3009283	00:00:00.0822295	
00:00:00.0232663	00:00:00.1886931	00:00:00.0700978	
00:00:00.0246420	00:00:00.1894184	00:00:00.0710346	
00:00:00.0248924	00:00:00.1519045	00:00:00.0677856	
00:00:00.0241019	00:00:00.1148362	00:00:00.0698150	
00:00:00.0256833	00:00:00.1587679	00:00:00.0678926	
00:00:00.0259725	00:00:00.1186616	00:00:00.0787719	
31 [ms]	194 [ms]	66 [ms]	**średnia [ms]**

W przypadku zapytania typu JOIN z ograniczeniami można zauważyć znaczący spadek wydajności Microsoft.SQL.Server Framework. Na szczególną uwagę zasługuje fakt, że Microsoft SQL Server przy JOINACH zaczął działać zdecydowanie szybciej, prawdopodobnie wynika to z wykorzystania zaimplementowanych wewnątrz mechanizmów optymalizujących. Mimo to przeszukiwanie przy pomocy grafu AGDS wciąż jest 2x szybsze.

Przeprowadzono testy poszczególnych funkcji i okazało się, że głównym hamulcem przy przeszukiwaniu jest wykorzystanie prostej struktury DTO, do której zapisywane są wyniki. W przypadku operowania na czystych referencjach osiągane rezultaty były znacznie powyżej oczekiwań i zmniejszyły czas przetwarzania o cały rząd wielkości (średnio 2[ms]).

Dalsze testy powinny zostać przeprowadzone w oparciu o rozszerzenie funkcji standardowych SQL jak np. SUM, COUNT, AVG, MEDIAN. Można przypuszczać, że na tym polu grafy AGDS również wykazałyby się nieporównywalnie lepszymi rezultatami ze względu na szybki dostęp do wartości granicznych. Obsługa pełnego zestawu operatorów SQL, jak również zaimplementowanie kopii mechanizmu *System.Data.DataSet* dla grafów AGDS jest osobnym tematem.

6. Podsumowanie

Niniejsza praca miała na celu wprowadzenie do zagadnień związanych ze sztucznymi systemami asocjacyjnymi oraz sztuczną inteligencją. Główną jej częścią było zapoznanie czytelnika z zagadnieniem relacyjnych baz danych oraz możliwości ich zastąpienia poprzez wykorzystanie grafowej asocjacyjnej struktury danych AGDS.
Struktura AGDS to jedna z najprostszych struktur należących do rodziny struktur asocjacyjnych. Główną zaletą, a zarazem ograniczeniem jest jej pasywna charakterystyka, która uniemożliwia dostosowywanie się do zmiennych danych w czasie rzeczywistym. W przypadku danych niemodyfikowalnych pozwala na stworzenie w pełni funkcjonalnej mapy danych, której przeszukiwanie było niezwykle szybkie.

W trakcie lektury czytelnik przeprowadzony został przez pełny proces myślowy od momentu odczytu danych z relacyjnej bazy danych do momentu przekształcenia jej w strukturę AGDS z wykorzystaniem dodatkowych algorytmów optymalizacyjnych jak B-Drzewa. Implementacja realizowana w środowisku .Net, z którym wiązało się zalety jak również pewne ograniczenia. Dzięki zastosowaniu podejścia obiektowego możliwe było zaimplementowanie w pełni generycznego interfejsu, którzy przyjmował dowolne tabele (relacje tabel) jako wejście i przeprowadzał ich transformacje.

Wyniki otrzymane w trakcie procesu testowania wykazały przewagę struktury AGDS nad relacyjną bazą danych. Wyniki mogły być znacznie lepsze, gdyby nie symulacja prostego obiektu typu DTO, który przekształcał wyniki do formatu zrozumiałego dla użytkownika. Na uwagę zasługuje fakt, że zarówno Microsoft.SQLServer Framework jak również oprogramowanie Microsoft SQL Server zostały w pełni zoptymalizowane, co sugeruje, iż w przyszłości implementacja struktury AGDS może osiągać jeszcze lepsze rezultaty czasowe.

W niniejszej pracy zostały zaimplementowane ograniczone funkcje języka SQL, które ograniczały się do prostych operatorów logicznych AND oraz OR, jak również operatorów porównania =, <, >. Niniejsza implementacja może zostać w przyszłości rozszerzona o dodatkowe funkcjonalności, jak np. AVG, SUM, MEDIAN, oraz o dodatkowe algorytmy przeszukiwania i sortowania, które mają główny wpływ na osiągane rezultaty czasowe. Wraz ze strukturą AGDS można zaimplementować dodatkowo w pełni funkcjonalny DataSet przeznaczony do powyższej struktury grafowej.

Kolejnym celem stawianym przed przedstawioną implementacją jest wykonanie interfejsu graficznego, które obrazowałby omówiony proces myślowy oraz zrezygnowanie z elementów języka zapytań LINQ dostępnego, dla języka C#, który charakteryzuje się prostotą oraz niekiedy sporym nakładem czasowym.

Omówiona struktura AGDS dzięki wykorzystaniu AVB-Drzew omówionych w rozdziale 3 może zostać przekształcona do grafu DASNG, który posiada cechy grafu aktywnego poprzez wykorzystanie modelu neuronu. Pozwoliłoby to na znaczne rozszerzenie dostępnych funkcji poprzez umożliwienie wnioskowania i wyszukiwania korelacji pomiędzy danymi. Niniejsza praca ujawniła, iż systemy asocjacyjne oraz sztuczna inteligencja mogą być kluczem do efektywności w zagadnieniach Big Data.

7. Bibliografia

[1] A. Horzyk, *Sztuczne systemy skojarzeniowe i asocjacyjna sztuczna inteligencja*, Akademicka Oficyna Wydawnicza EXIT, 2013.

[2] A. Horzyk, *Deep Associative Semantic Neural Graphs for Knowledge Representation and Fast Data Exploration*, Proc. of KEOD 2017, SCITEPRESS Digital Library, 2017.

[3] A. Horzyk, *Neurons Can Sort Data Efficiently*, In: Rutkowski L., Korytkowski M., Scherer R., Tadeusiewicz R., Zadeh L., Zurada J. (eds), Artificial Intelligence and Soft Computing, Proc. of ICAISC 2017, Springer-Verlag, LNCS, Vol. 10245, pp. 64-74, 2017, DOI: 10.1007/978-3-319-59063-9_6.

[4] A. Horzyk, *Human-Like Knowledge Engineering, Generalization and Creativity in Artificial Neural Associative Systems*, Springer-Verlag, AISC 11156, ISSN 2194-5357, ISBN 978-3-319-19089-1, ISBN 978-3-319-19090-7 (eBook), DOI 10.1007/978-3-319-19090-7, Springer, Switzerland, 2016, pp. 39-51, DOI: 10.1007%2F978-3-319-19090-7_4.

[5] A. Horzyk, *Innovative Types and Abilities of Neural Networks Based on Associative Mechanisms and a New Associative Model of Neurons* - referat na zaproszenie na międzynarodowej konferencji ICAISC 2015, Springer-Verlag, LNAI 9119, 2015, pp. 26-38, DOI 10.1007/978-3-319-19324-3_3.

[6] A. Horzyk, *How Does Generalization and Creativity Come into Being in Neural Associative Systems and How Does It Form Human-Like Knowledge?*, Elsevier, Neurocomputing, 2014, pp. 238-257, DOI: 10.1016/j.neucom.2014.04.046.

[7] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Algorytmy i struktury danych*, Wydawnictwo Helion, 2003.

[8] T. H. Cormen, Ch. E. Leiserson, *Wprowadzenie do algorytmów*, Wydawnictwo Naukowo-Techniczne Warszawa, 1994.

[9] J. Albahari, B. Albahari, C# 6.0 w Pigułce, Wydawnictwo Helion, 2016.

[10] R. C. Martin, *Zwinne wytwarzanie oprogramowania*, Wydawnictwo Helion, 2015

[11] K. Czapla, *Bazy danych. Podstawy projektowania i języka SQL*, Wydawnictwo Helion, 2015

[12] *SQL Podstawy*, Dostęp 2017-08-12, URL: *http://www.sqlpedia.pl/*

[13] Jason Gillikin, *What Are the Limitations of Relational Databases in Business Applications?*, Dostęp 2017-08-17, URL: *http://smallbusiness.chron.com/limitations-relational-databases-business-applications-24159.html*

[14] *Oracle Docs*, Dostęp 2017-08-19, URL: www.docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL190

[15] H. Garcia-Molina, J.D. Ullman, J. Widom *Systemy baz danych. Kompletny podręcznik,* Wydawnictwo Helion, 2011

[16] R. Elmasri, S. B. Navathee, *Wprowadzenie do systemów baz danych*, Wydawnictwo Helion, 2005

[17] C. M. Costa, A. L. Sousa, *Adaptive Query Processing in Cloud Database Systems*, IEEE Third International Conference on Cloud and Green Computing, 2013

[18] *Rule-Based vs Cost-Based Optimization*, Dostęp: 2017-07-06, URL: *www.searchoracle.techtarget.com/answer/Rule-based-vs-cost-based-optimization*

[19] S. Deepak, S. U. Kumar, Sh. Deepak, *The performance ehancement approach for parameterized queries*, Software Engineering (CONSEG), 2012

[20] *TSqlParser* Dostęp: 2017-07-31 URL: *https://github.com/bruce-dunwiddie/tsql-parser*

[21] What is selectivity in SQL Dostęp: 2017-07-08, URL: *www.programmerinterview.com/index.php/database-sql/selectivity-in-sql-databases/*