Uniwersytet Łódzki

Wydział Matematyki i Informatyki

Instytut informatyki

Andrzej Krupa

nr indeksu: 338689

Język wysokiego poziomu do tworzenia procesów ETL w Hurtowniach danych.

High-level language to create ETL process in data warehouse.

Praca magisterska

przygotowana w Zakładzie Katedry Informatyki Stosowanej

promotor: dr Jan Pustelnik

Łódź 2014

Spis treści

Wstęp

Ojcem koncepcji hurtowni danych jest Bill Inmon, napisał on ponad 40 książek związanych z tą tematyką. Koncepcja ta dotyczy, jak wspomóc osoby zarządzające firmą, korporacją w podejmowaniu działań strategicznych. Hurtownie danych odniosły sukces związany z problemem biznesowym związanym z zarządzaniem relacjami z klientem, w skrócie CRM, *(ang. Customer Relationship Management)*

Projektowanie jaki i tworzenie hurtowni danych jest procesem bardzo złożonym i kosztownych, który trwa od pół roku do dwóch lat. Firmy podejmujące decyzje o inwestycji utworzenia hurtowni danych, są świadome, że nie produkt zakupiony ma generować zyski tylko dostarczać wiarygodnych i rzetelnych informacji, na podstawie których możliwe jest podjęcie decyzji strategicznych. Jeżeli projekt hurtowni danych jest nie ukierunkowany pod danego klienta lub przechowuje niepoprawne dane, to staje się dużą stratą dla firmy.

Celem pracy jest napisanie języka wysokiego poziomu, który pomoże programistą w tworzeniu procesów zasilających hurtownie danych. Zadaniem owego języka, na podstawie podanych poleceń jest wygenerowanie:

- szablonu pobierającego dane (źródło),
- szablonu pgloader lub gotowe polecenia insert,
- kodu umożliwiającego utworzenie tabeli,
- kodu języka SQL zasilającego table.

Pierwszy rozdział pracy opisuje hurtownie danych i powody jej budowanie, jak również została przedstawiona w nim architektura hurtowni danych. Kolejny rozdział

opisuje, czym są procesy zasilania hurtowni danych, oraz omawia przykładowe procesy zasilania, które są realizowane w ramach niniejszej pracy.

(Zastanawiam się jeszcze, czy w tym miejscu napisać o bazie PostgreSQL 8.4.14 ,czy na samym końcu wstępu. Język wysokiego poziomu,który mam napisać, planuje testować na tej bazie.)

(zarys 3,4,5) Trzeci rozdział zawiera definicję i pojęcia, które są niezbędne są do zrozumienia czwartego rozdziału pracy, opisującą tematykę tworzenia języków interpretowanych . Ostatni rozdział niniejszej pracy dotyczy opisu programu wraz z przykładem.

Rozdział 1

Hurtownie danych

Definicję Hurtowni Danych *(ang. Data Warehouse)* przypisuje się Bill'owi Inmon'owi w 1992 roku. Zgodnie z tą definicją Hurtownią danych jest bazą danych mającą następujące cztery cechy.

- Zorientowaną na temat *(ang. Subject-oriented)* – dane są gromadzone w ściśle określonej dziedzinie, aby możliwe było zrobienie sensownego zestawienia danych. Nie są przechowywane działania czy operacje biznesowe. Hurtownia danych ograniczona w firmie do jednego działu lub wybranego obszaru (np. Biznesowego), jest określana jako lokalna hurtownia danych lub tematyczna hurtownia danych *(ang. data mart)* stanowiąca podzbiór hurtowni danych

- Nie ulotność *(ang. Non-volatile)* – dane przechowywane w Hurtowni danych nie są nigdy usuwane i modyfikowane, przeznaczone są wyłącznie do odczytu w celu utworzenia raportu na podstawie zadanego zapytania SQL.

- integracja *(ang. Intergrated)* – W hurtowni danych znajdują się informacje, które pochodzą z całej firmy, przechowywanych w dowolnych technologiach, związku z tym faktem musi wystąpić ujednolicenie typów danych.

- Zmienność w czasie *(ang. Time-Variant)* – Na podstawie historii są podejmowane decyzje, musi zostać określone, co jaki okres czasu chcemy zapamiętać stan obecny w danej firmie.

1.1 Powody budowania hurtowni danych.

Uzasadnieniem budowania hurtowni danych może być:

- **Przeprowadzanie analizy danych bez ingerencji w operacyjną pracę systemów transakcyjnych.** – Analiza danych ze względu na bardzo dużą liczbę danych wymagają złożonych i czasochłonnych obliczeń. Dopuszczalne są zapytania kilku sekundowe, minutowe. Mogą wystąpić zapytania nawet kilku dniowe. Zapytania te nie mogą wpłynąć na pracę systemu operacyjnego, w którym zapytanie nie może trwać dużej niż kilka sekund. (np. Użytkownik płacący kartą nie wie, czy odpowiedź o akceptacji przyjdzie za jedną sekundę, czy za 2 minuty, czy za 5 minut. Jest to sytułacja nie dopuszczalna.)

- **Całościowy wgląd w dane firmy** – Firmy posiadające dane na różnych środowiskach sprzętowych, w różnych aplikacjach zainstalowane. Posiada głębszą wiedzę na temat zdarzeń, które miały miejsce w jej firmie, jeżeli ma możliwość zintegrowania danych. Np. Pan K. ma sklep i warsztat samochodowy i chciałby widzieć. Ile sprzedanych części samochodowych i kto naprawiał samochód w jego warsztacie, a kto nie.

- **Dostęp do danych historycznych** – Dzięki danym historycznym możliwe jest wykonywanie analiz, z których można wyciągnąć wnioski, przekładające się na realne korzyści dla firmy.

- **Ujednolicenie posiadanych informacji** – Eliminuje tzw. problem wielu wersji prawdy firmy. Przedstawiony raport opiera się na podstawie jakiś danych. Jeżeli dane pochodzą z różnych źródeł to są to wnioski osoby soporządzającej raport. Firma w jednym obszarze może prosperować bardzo dobrze, ale inny obszar może generować straty, które mogą być przyczyną upadku firmy.

- **Przetwarzanie analityczne danych** *(ang. On-Line Analytical Processing, OLAP)* – Z danych zgromadzonych w hurtoni danych są tworzone zestawienia statystyczne, wykresy i raporty w różnych okresach czasowych.

- **Wspomaganie decyzji** *(ang. Decision Support, DS)* - wykonywanie analizy symulującej scenariusz biznesowy.

1.1.1 OLAP a OLTP

Przetwarzanie analityczne danych *(ang. On-Line Analytical Processing, OLAP)* i przetwarzanie transakcyjne *(ang. On-Line Transactional Processing, OLTP)* są to systemy optymalizowane pod kątem przetwarzania danych.

System OLTP jest przeznaczony dla pracowników, komunikujących się z Systemem bazodanowym w celu uzyskania informacji np. sprawdzenie dostępnych miejsc na jakimś koncercie.

Podstawowymi cechami systemów OLTP są:

1. wykonywanie przez wielu użytkowników bardzo dużej ilości zapytań, których czas realizacji jest krótki ,
2. system bazodanowy powinien być zoptymalizowany pod kątem odczytu danych,
3. częste usuwanie lub modyfikacja pojedynczych rekordów w bazie danych,
4. dane przechowywane w bazie danych są zawsze aktualne.

System OLAP jest przeznaczony dla pracowników przygotowujących zestawienie danych, raportów dla kadry zarządzającej, jak również dla analityków, którzy na podstawie zadanych zapytań do hurtowni danych, mogą odkryć zależności występujące w firmie, a następnie wyciągnąć odpowiednie wnioski, które w ich opinii mogą dać firmie zysk.

Podstawowymi cechami systemów OLAP są:

1. wykonywanie przez nie wielką liczbę użytkowników małej ilości zapytań na dużym obszarze danych,
2. cyklicznie zasilane w ustalonych przedziałach czasowych,
3. dane w bazie nie muszą być aktualne w czasie rzeczywistym.

1.1.2 Wspomaganie decyzji

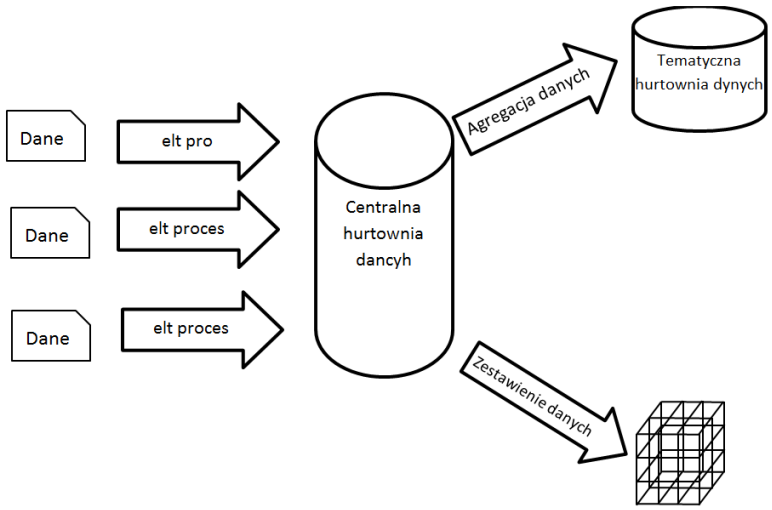
Systemy wspomagania decyzji *(ang. decision support systems)* tworzone są w celu szukania minimalizacji kosztów prowadzonej działalności, lepszego przewidywania ryzyka podejmowanych działań, podniesienia jakości obsługi klienta. System OLAP jest jednym z takich narzędzi, które wspierają podejmowanie decyzji. Przykładowymi pytaniami na które system powinien odpowiedzieć są:

1. Jaki był dochód w rozbiciu na poszczególnych klientów?
2. Jaki był procentowy wzrost lub spadek dochodu w porównaniu z zeszył miesiącem?
3. Jakie są cech najlepszych/najgorszych klientów (cechy klientów muszą być ściśle określone)?
4. Listę klientów, dla których współczynnik odejścia jest wysoki, a przynoszą zysk firmie.

Hurtownie danych odniosły sukces związanym z zarządzanie relacjami z klientem *(ang. Customer Relationship Management, CRM)*, które mają na celu zatrzymanie najlepszych klientów, sprzedawanie im większej liczby produktów, jak również pozyskiwanie nowych klientów.

1.2 Architektura hurtowni danych

W niniejszym podrozdziale zostanie przestawiona podstawowa architektura hurtowni danych oraz proces związany z tworzeniem hurtowni, który jest bardzo drogi, czasochłonny i żeby osiągnął sukces musi on być ukierunkowany na klienta, czyli pod jego wymagania, które opierają się na intuicji. Na rysunku 1.1 przedstawia główne elementy hurtowni danych oraz kierunek przepływu danych. Przy użyciu strzałek został pokazany przepływ danych.



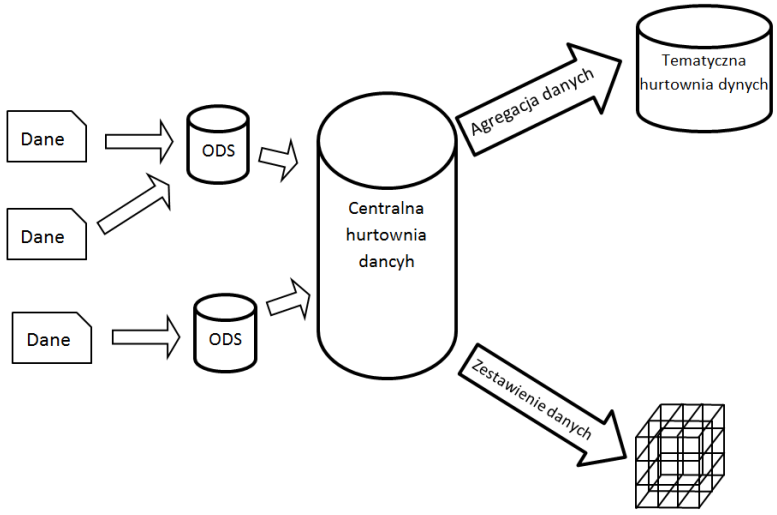
Rysunek 1.1: Architektura hurtowni danych.

Strukturę przepływu danych możemy podzielić na:

- **Źródło danych** ***(ang. source)*** — są to dane, które będą pobierane do hurtowni danych
- **proces ETL** ***(ang. extract, transfer, load)*** — procesem elt nazywamy czynności wykonywane w celu pobrania danych źródłowych przekształcenie w odpowiedni format danych, a następnie umieszczeni ich w centralnej hurtowni danych, proces elt dokładnie będzie omówiony w rozdziale drugim.
- **centralna hurtownia danych** ***(ang. center data warehouse)*** — jest to miejsce docelowe przetowrzonych danyc ze źródeł,
- **hurtownie tematyczne** ***(ang. data marts)*** — zawierają wybrane dane z centralnej hurtowni danych w sposób zagregowany, umożliwiające szybkie operowanie sporządzanie raportów,
- **zestawienie danych** — docelowym produktem hurtowni danych jest tworzenie odpowiednich zestawień danych. Na rysunku 1.1, został przedstawiona jako kostka.

Tworzenie hurtowni danych jest stosunkowo młodą dziedziną, która się dynamicznie rozwija. Dzięki zastosowaniom CRM odniosły one sukces co spowodowało większe

zapotrzebowanie na przechowywanie i analizowanie danych historycznych. Przedstawiona architektura danych na rysunku 1.1, nie spełnia swojej roli dla Hurtowni Danych, w których przyrost danych jest bardzo duży. Poniżej została przedstawione inna architektura danych.



Rysunek 1.2: Architektura hurtowni danych z magazynem danych ODS.

Do architektury z rysunku 1.1 został dodany magazyn danych operacyjnych *(ang. operational data store, ODS*), który pełni role magazynu danych. Ładowane są do niej dane pobrane ze źródeł i przetworzone w celu uzyskania zgodności typów danych. Kolejnym etapem jest załadowanie danych w sposób zagregowany do centralnej hurtowni danych.

1.3 Projektowanie hurtowni danych.

Projektowanie hurtowni danych tak jak relacyjnych baz danych polega na utworzeniu następujących modeli:

- **Model pojęciowy** — przy użyciu języka biznesowego w danej firmie opisuje się cele biznesowe, które będzie można określić przez gromadzenie ściśle określonych danych. Na modelu pojęciowym powinny, być zaznaczone nazwy kolumn, które mają być przechowywane w tabeli znajdującej się w Hurtowni Danych

- **Model logiczny** — jest to opis elementów logicznych hurtowni danych, wykonany np. w języku UML.
- **Model fizyczny** — jest to opis indeksowania, partycjonowania, opis sprzętu komputerowego, sieci rozmieszczenie poszczególnych zasobów fizycznych.

Najpopularniejszymi metodami przyjętymi podczas tworzenia hurtowni danych są:

- **Projektowanie wstępujące** (od szczegółu do ogółu)— polega na tworzeniu wszystkich etapów hurtowni danych jednocześnie, a następnie na integracji poszczególnych etapów ze sobą.
- **Projektowanie zstępujące** — Dopóki jeden etap tworzenia hurtowni danych się nie skończy, to następny się nie zacznie. Jeżeli pojawią się błędy to wraca się do poprzedniego etapu i zaczyna się prace na kolejnym etapie od nowa.

1.4 Wielowymiarowy danych danych

Wielowymiarowym modelem danych *(ang. Multidimensional Data Model*) nazywamy dane zorganizowane w:

- **fakt** ***(ang. facts)*** — są to dane opisujące jakieść zdarzenie, tabelkę przechowującą te dane nazywamy *tablicą faktów* Fakt opisany jest przez wymiary i miary,
- **wymiar** ***(ang. dimension)*** — Jest jakąś cechą opisującą dany fakt, cechy te znajdują się w tablicy wymiarów i są opisane przez atrybuty,
- **atrybut** *(ang. attribute)* — Przechowuje dodatkowe informację na temat wymiaru,
- **miara** ***(ang. measures)*** — Jest wartością mierzalną przypisaną do pojedynczego rekordu w tablicy faktów.

Model ten jest zintegrowaną częścią z systemem OLAP. Podstawowym atutem wielowymiarowego modelu danych jest proste zrozumienie hurtowni danych i poruszania się po niej w sposób efektywny, szybsze wykonywanie zapytań zadawanych do hurtowni danych, Jak również możliwość analizy danych w różnych wymiarach, które jest bardzo istotne ze względów biznesowych:

- Oglądanie informacji rozłożonych w czasie,
- Wyświetlanie informacji w sposób graficzny,
- Możliwość zmiany przekroju danych w dowolny sposób,
- Analizę danych pod kątem informacji istotnych dla danej firmy.

Podstawowymi schematami wielowymiarowego modelu danych są:

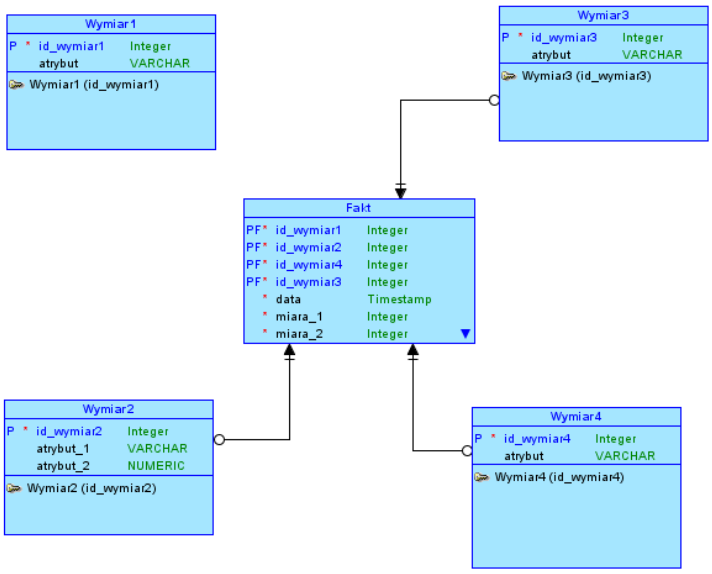
- schemat gwiazdy *(ang. Star schema)*
- schemat płatka śniegu *(ang. Snowflake schema)*

1.4.1 Schemat gwiazdy

Schemat gwiazdy jest podstawowym schematem wielowymiarowego modelu danych, w którym znajduje się jedna tabela faktów połączona z wieloma tabelami wymiarów. Tabela faktów w tym schemacie jest w trzeciej postaci normalnej, a tabela wymiarów jest w drugiej postaci normalnej, dzięki takiej strukturze możliwe jest szybsze przeglądanie danych poprzez:

- poszczególne wymiary,
- sumowanie danych,
- agregację danych,
- filtrowanie danych

Na rysunku 1.3 został przedstawiona przykładowa architektura schematu gwiazdy.



Rysunek 1.3: Przykładowy schemat gwiazdy w postaci abstrakcyjnej.

Poniżej znajduje się listing zapytań do bazy danych w języku postgresql, który realizuje model logiczny gwiazdy zawarty na rysunku 1.3

Listing 1.1: Listing kodu tworzący schemat gwiazdy.

```
1  DROP TABLE IF EXISTS fakt;
2  DROP TABLE IF EXISTS wymiar1;
3  DROP TABLE IF EXISTS wymiar2;
4  DROP TABLE IF EXISTS wymiar3;
5  DROP TABLE IF EXISTS wymiar4;
6  CREATE TABLE wymiar1
7  (
8    id_wymiar1 integer PRIMARY KEY
9  , atrybut varchar
10 );
11
12 CREATE TABLE wymiar2
13 (
14   id_wymiar2 integer PRIMARY KEY
15 , atrybut_1 varchar
16 , atrybut_2 numeric
```

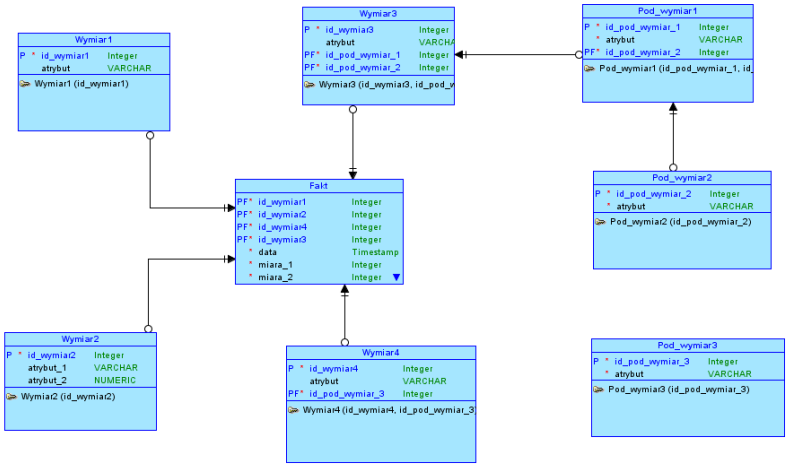
```
17 );
18
19 CREATE TABLE wymiar3
20 (
21   id_wymiar3 integer PRIMARY KEY
22 , atrybut varchar
23 );
24
25 CREATE TABLE wymiar4
26 (
27   id_wymiar4 integer PRIMARY KEY
28 , atrybut varchar
29 );
30
31 CREATE TABLE fakt
32 (
33   id_wymiar1 integer  references wymiar1(id_wymiar1)
34 , id_wymiar2 integer  references wymiar2(id_wymiar2)
35 , id_wymiar3 integer  references wymiar3(id_wymiar3)
36 , id_wymiar4 integer  references wymiar4(id_wymiar4)
37 , data       timestamp not null
38 , miara_1    integer  not null
39 , miara_2    integer  not null
40 );
```

1.4.2 Schemat płatka śniegu

Architektura schematu gwiazdy jest uproszczoną formą architektury płatka śniegu. Podstawową różnicą pomiędzy tymi schematami jest tabela wymiarów, która jest znormalizowana.

Schemat płatka śniegu jest stosowany wtedy, gdy tabela wymiarów osiąga duży rozmiar. Normalizuję się tabele wymiarów, aby zmniejszyć jej liczebność, dzięki czemu czas zapytań, powinien się znacząco skrócić. Wadą tego podejścia jest, że im bardziej znormalizowana jest tabela wymiarów, to tym bardziej skomplikowane łączenia SQL muszę zostać użyte, aby pobrać odpowiednie dane z hurtowni danych. [6]

Przykładowa architektura płatka śniegu został przedstawiona na rysunku 1.4 .



Rysunek 1.4: Przykładowy schemat płatka śniegu w postaci abstrakcyjnej.

Rozdział 2

Procesy zasilania hurtowni danych

2.1 Ogólna koncepcja zasilania hurtowni

Hurtownia danych jako system magazynujący dane i wspierający raportowanie stawia sobie jako jeden z głównych celów gromadzenie danych i takie ich przekształcanie, aby przyszłe raportowanie było jak najłatwiejsze w kontekście pytań biznesowych, jakie stawiają użytkownicy końcowi. Ogół procesów zasilania jest najczęściej określany skrótem ETL, pochodzącym od angielskich słów Extract, Transform, Load (ekstrakcja, transformacja, ładowanie), które oddają charakter procesów, oraz podsumowują cele, jakie są stawiane przed procesami zasilania hurtowni danych.

2.1.1 Ekstrakcja

Dane, które ostatecznie trafiają do hurtowni, pochodzą z różnych źródeł w firmie lub poza nią i różnią się sposobem dostępu. Źródłami mogą być systemy transakcyjne (transakcje bankowe, system płatności online, systemy obsługi klienta, zapisy partii szachowych online itp.), logi systemów (logi stron internetowych, systemów e-commerce, pliki z wykazem połączeń telefonicznych), publicznie dostępne pliki (dane giełdowe, wskaźniki i dane makroekonomiczne GUS czy nawet ręcznie generowane przez użytkowników biznesowych (arkusze kalkulacyjne, plany i cele sprzedaży). Różnorodność źródeł stawia po stronie hurtowni danych konieczność ekstrakcji danych z formatu,

w którym są dostępne, niezależnie od źródła i formatu (pliki tekstowe, bazy danych, arkusze kalkulacyjne, dane nieustrukturyzowane, obrazy itp.). Czasami trudność przedstawia samo znalezienie właściwego źródła danych, bądź znalezienie kilku źródeł, które razem zawierają potrzebne informacje, czasami najtrudniejsze jest wykonanie ekstrakcji (zwłaszcza, jeśli mamy do czynienia z wiekowymi systemami pisanymi kilkadziesiąt lat temu w języku COBOL na komputerach klasy mainframe — wbrew pozorom tego typu systemy są jeszcze w użyciu). Mogą również pojawić się problemy wydajnościowe związane z transportem danych (np. danych jest na tyle dużo, że wąskim gardłem staje się przepustowość sieci i należy uciec się do kompresji danych jako do rozwiązania problemu). Jednym z problemów do rozwiązania przy ekstrakcji danych jest minimalizacja wpływu procesów ekstrakcji na funkcjonowanie źródła danych — zazwyczaj systemy transakcyjne nie potrafią sobie poradzić z pobieraniem dużej ilości danych, gdyż same z siebie są zoptymalizowane do szybkich przetwarzań niewielkich ilości danych. Może okazać się, że próby ekstrakcji danych bezpośrednio z systemu źródłowego są tak wielkim obciążeniem wydajnościowym, że doprowadzają do nieakceptowalnych czasów działania systemu transakcyjnego. Do typowych rozwiązań należy harmonogramowanie procesów ETL w taki sposób, by ekstrakcja odbywała się np. w nocy, kiedy użytkowników systemu transakcyjnego jest mniej lub nie ma ich wcale, oraz korzystanie z kopii systemu źródłowego dla celów ekstrakcji (np. druga baza danych połączona z pierwszą w system „hot standby” lub po prostu zwykła kopia odtwarzania z codziennych backupów systemu transakcyjnego.

2.1.2 Transformacja

Po pobraniu danych, dane muszą zostać przekształcone do wspólnej postaci. Wynika to z faktu, że różne źródła, nawet podobne do siebie, przechowują dane w różnej postaci i zakładają różne zależności pomiędzy poszczególnymi elementami. Na przykład, bank powstały w wyniku fuzji kilku innych banków może korzystać z kilku systemów obsługi klienta. W jednym systemie kluczowi klienci biznesowi mogą mieć przypisanego jednego opiekuna i taka bieżąca informacja jest dostępna w systemie źródłowym, modelowana jako relacja jeden do wielu (jeden opiekun dla wielu klientów), natomiast w drugim systemie może to być relacja wiele do wielu z archiwizacją historii przypisań

opiekunów do klientów. Dane w systemach transakcyjnych bardzo często są zgodne z trzecią postacią normalną, natomiast hurtownia danych często przechowuje i prezentuje dane w postaci zdenormalizowanej, zwłaszcza jeśli do modelowania hurtowni został wybrany schemat gwiazdy. Kontynuując przykład opiekuna klienta, model hurtowni danych może przewidywać opiekuna jako zwykły atrybut klienta, ignorując fakt, że w rzeczywistości relacja jest postaci jeden do wielu bądź wiele do wielu i tak jest zamodelowana w systemach źródłowych. Tego typu transformacje oraz zamiany kluczy z systemów źródłowych na własne klucze używane przez hurtownię (zazwyczaj zwane w środowisku hurtowni danych kluczami sztucznymi) są podstawowymi zadaniami procesów zasilania hurtowni. Do innych typowych przekształceń należy uwspólnianie typów danych (np. ten sam atrybut może być opisywany w różnych systemach przez ciągi znaków różnej długości), ujednolicanie zawartości atrybutów (np. typ klienta może przybierać wartości „biznesowy", „business", 3, „firma" itp. w różnych systemach, w hurtowni chcemy przechowywać jedną wartość, wspólną dla wszystkich rekordów jednego typu), łączenie bądź rozdzielanie atrybutów (np. „małżeństwo z dziećmi" chcemy rozdzielić na dwa atrybuty, jeden opisujący „małżeństwo", drugi niosący informację, czy „ma dzieci"), łączenie atrybutów (np. rodzaj „firma", rozmiar „poniżej 200 pracowników" chcemy oznaczyć jako „SME"). Możliwe są również przekształcenia specyficzne dla danej dziedziny, np. wyliczanie stopy zwrotu czy procentowej zmiany cen z danych giełdowych.

2.1.3 Ładowanie

Przekształcone dane muszą trafić do docelowego modelu danych. Większość pracy została już wykonana na etapie ekstrakcji i transformacji, jednak pozostaje zadbać o spójność danych (czy podczas ładowania hurtownia pozwoli na dostęp do transakcji dla nowych klientów, których jeszcze nie zdążyliśmy załadować?). Często tego typu kwestie są rozwiązywane w zupełnie inny sposób niż w systemach transakcyjnych, w których spójność danych jest zapewniana mechanizmami bazodanowymi typu transakcje czy więzy integralności. Przenoszenie takich rozwiązań do hurtowni danych stwarza potencjał dla problemów wydajnościowych zależnych bądź nie od implementacji konkretnego systemu zarządzania bazami danych – np. więzy integralności spowalniają ładowanie,

gdyż są sprawdzane wiersz po wierszu, a transakcje czasami wręcz nie są możliwe do użycia z uwagi na ograniczenia techniczne przy dużej ilości danych przetwarzanych w hurtowniach. Przykładowo, w s.z.b.d. Oracle, transakcje standardowo generują UNDO i REDO, więc zawarcie całości ładowania w jednej transakcji, nawet gdyby było technicznie możliwe (bazę można skonfigurować, aby udostępniała wystarczająco dużo miejsca na UNDO i REDO), stwarzałoby olbrzymie problemy wydajnościowe z uwagi na kilkukrotne zwiększenie ilości operacji wejścia/wyjścia (należy pamiętać, że UNDO też jest chronione przez REDO, więc ilość operacji dyskowych może wzrosnąć nawet czterokrotnie). Typową „sztuczką" praktyczną bywa np. wyłączenie generowania REDO na poziomie bazy danych. Jeśli chodzi o UNDO, to typowym rozwiązaniem jest podzielenie transakcji na mniejsze części, co redukuje ilość UNDO, które musi być przechowywane (każde zatwierdzenie transakcji pozwala na pozbycie się UNDO, które zostało wygenerowane przez daną transakcję), ale to już oznacza, że zapewnienie spójności musi leżeć po stronie procesów zasilania. Na szczęście zasilanie hurtowni danych jest procesem wsadowym, który w razie czego może zostać powtórzony, więc większość tradycyjnych mechanizmów bazodanowych zapewniających ochronę transakcji nie jest potrzebna. Zapewnienie spójności danych jest realizowane w samych procesach ładowania, chociażby przy ustalaniu kolejności zasileń. Przykładowo, ładując hurtownię zbudowaną w oparciu o schemat gwiazdy, tradycyjnie ładuje się najpierw wymiary, a potem fakty. Pomimo że nie zapewnia to spójności w najściślejszym znaczeniu tego pojęcia (mogą pojawiać się wiersze w tabelach wymiarów, które nie mają swoich odpowiedników w tabelach faktowych), to w praktyce jest to spójność, jakiej oczekują użytkownicy i personel utrzymujący hurtownie danych. Wynika to z faktu, że w modelu gwiazdy wymiary są używane do interpretacji danych faktowych, a więc ich znaczenie jest drugorzędne. Najbardziej istotne jest, aby wszystkie dane faktowe dostępne dla użytkownika były opisane przez wymiary, więc ładowanie wymiarów w pierwszej kolejności zapewnia ten stan rzeczy.

2.2 Analiza przykładowego procesu zasilania

Dla zilustrowania koncepcji języka do budowy hurtowni danych oraz jego praktycznego zastosowania, zostanie zbudowane przykładowa, uproszczona hurtownia danych wraz z procesami zasilania. Tematyką hurtowni będą dane giełdowe, a konkretnie notowania ciągłe z warszawskiej Giełdy Papierów Wartościowych (GPW). Wybór ten jest umotywowany powszechną dostępnością danych — każdy może sobie w dowolnej chwili pobrać publicznie dostępne dane z wielu różnych stron internetowych. Nie bez znaczenia jest również prostota danych, które są intuicyjnie zrozumiałe dla większości osób i nie będą wymagały wyjaśniania.

Do zbudowanie hurtowni danych zostanie użyty schemat gwiazdy, w którym występować będzie jedna tabela faktów, zaprezentowana na listingu 2.1, wraz z towarzyszącym jej wymiarem pokazanym na listingu 2.2.

Listing 2.1: Kod tworzący tabelę faktów.

```
1  drop table if exists public.gpw;
2  create table public.gpw
3  (
4    npw_id integer
5  , data_notowania date
6  , otwarcie decimal(20, 2)
7  , max  decimal(20, 2)
8  , min decimal(20, 2)
9  , zamkniecie decimal(20, 2)
10 , wartosc decimal(20,3)
11 );
```

Listing 2.2: Kod tworzący tabelę wymiaru.

```
1  drop table if exists npw;
2  drop sequence if exists npw_kmap_seq;
3  create sequence  npw_kmap_seq
4    increment by 1
5    no minvalue
6    no maxvalue
7    start with 1
8    cache 1
9    cycle;
10
11 create table npw
```

```
12 (
13   npw_id integer default nextval('npw_kmap_seq')
14 , nazwa varchar(50)
15 );
```

Docelowe rozwiązanie będzie składało się z trzech warstw:

- warstwy interfejsowej,
- warstwy pośredniej,
- warstwy docelowej,

które zostaną omówione w kolejnych podrozdziałach.

2.2.1 Warstwa interfejsowa

Warstwa interfejsowa będzie służyła do komunikacji ze światem zewnętrznym celem pobrania danych do hurtowni. Przed procesami zasilania będą postawione następujące zadania szczegółowe:

1. Wykrywanie nowych danych w systemie źródłowym
2. Pobieranie danych z systemu źródłowego
3. Ładowanie plików do bazy danych

Dane o cenach akcji GPW są publicznie dostępne w internecie i są już wyekstraktowane w postaci gotowych do ściągnięcia plików tekstowych. Zatem nie jest konieczne, aby procesy zasilania wykonywały ekstrakcję ze źródła (jakiegoś systemu operacyjnego). Zamiast tego, przydatne będzie wykrywanie, czy od ostatniego wykonania procesów zasilania pojawiły się nowe pliki z danymi i pobranie wyłącznie nowych danych.

Cel ten zostanie zrealizowany za pomocą skryptów powłoki systemu Linux, przedstawiony na listingu oraz podstawowych narzędzi systemowych dostępnych z poziomu systemu operacyjnego.

Listing 2.3: Skrypt pobierający dane serwera bossa.pl .

```
# skrypt do ściągania danych giełdowych z GPW
# dane pochodzą ze strony bossa.pl/notowania/metastock
#(notowania ciągłe , dane bieżące, plik sesjacgl.prn)

if [ ! -f sesjacgl.prn ]
then
 touch sesjacgl.prn
 md5sum sesjacgl.prn > md5file.md5
fi

if [ ! -f md5file.md5 ]
then
 md5sum sesjacgl.prn > md5file.md5
fi

t=$(date '+%s')
while [[ $(($(date '+%s') - $t)) -le 10800 ]]
do
  # ściągamy plik jeśli jest nowszy niż uprzednio ściągnięty
  wget http://bossa.pl/pub/ciagle/mstock/sesjacgl/sesjacgl.prn -N 2>/dev/null
  sleep 5s
  # Jeżeli suma kontrolna się zmieni, to znaczy, że plik pod w/w adresem się
    zmienił
  # i zostanie wykonany warunek
  # Jeżeli w zmiennej $? będzi 1, to plik został zmienony
  md5sum -c md5file.md5
  wynik=$?
  echo $wynik
  if [  $wynik -eq 1 ]
  then
    echo "------ Plik jest nowy ----------"
    # Tworzony jest plik sesja.txt, który jest wykorzystywany do ładowania danych
    do bazy.
    md5sum sesjacgl.prn > md5file.md5
    cp ./sesjacgl.prn ./copySesjacgl.prn
    tar -zcvf  ./arch_gpw/arch$(date '+%Y_%m_%d_%H%M%S').tar.gz sesjacgl.prn
    cp  sesjacgl.prn sesja.txt
    break
  fi
done
```

Procesy ETL muszą pobrać dane i przygotować do ładowania, są to wszystkie czynności, które programiści uznają za potrzebne w celu przygotowania danych do poprawnego załadowania danych do tabelki interfejsowej. Czynnościami tymi chociażby

może być rozpakowanie danych i rozmieszenie ich w odpowiednich katalogach. W naszym rozważanym przykładzie pobieramy plik o nazwie *sesjacgl.prn*, który jest plikiem tekstowym.

Pliki muszą zostać załadowane do bazy danych w niezmienionej formie celem ich udostępnienia do dalszych przekształceń. Czyli trafiają do tabelki interfejsowej, która ma typ zmiennych zgodny z pobranymi danymi źródłowymi. Tabelkę te nazywać będziemy *intf_pgw*, a ma ona strukturę zaprezentowaną na listingu 2.4 .

Listing 2.4: Kod tworzący tabelkę interfejsową.

```
1  drop table if exists intf_gpw;
2  create table intf_gpw
3  (
4    nazwa varchar(50)
5  , data_notowania date
6  , otwarcie decimal(20, 2)
7  , max decimal(20, 2)
8  , min decimal(20, 2)
9  , zamkniecie decimal(20, 2)
10 , wartosc decimal(20,3)
11 );
```

Dla umożliwienia pełnej audytowalności procesów, konieczne jest przechowywanie ściągniętych plików przynajmniej przez jakiś czas. W razie wystąpienia wątpliwości odnośnie jakości danych, będzie możliwość weryfikacji danych i porównania hurtowni z danymi źródłowymi. Często w hurtowniach danych istnieje dedykowana warstwa służąca tylko temu celowi. W rozwiązaniu zbudowanym na potrzeby niniejszej pracy, archiwizacja będzie odbywać się za pomocą kompresji plików i przeniesienia ich do dedykowanego katalogu, co zostało pokazane na listingu 2.3.

W komercyjnych rozwiązaniach, jeśli archiwizacja danych odbywa się za pomocą plików, najczęściej stosowane są specjalistyczne narzędzia do backupów, a dane ostatecznie nagrywane są na taśmy. Z uwagi na wysoką cenę tego typu urządzeń oraz łatwą publiczną dostępność danych źródłowych użytych na potrzeby niniejszej pracy, taki poziom dbałości o bezpieczeństwo danych nie jest konieczny.

W celu załadowania danych do bazy danych zostanie użyte narzędzie pgloader, które dobrze współpracuje z systemem zarządzania bazą danych Postgres. Na listingu 2.5 został pokazany skrypt pgloader'a , który realizuje to zadanie.

Listing 2.5: Skrypt pgloader'a ładujący dane do tabelki interfejsowej.

```
1  [pgsql]
2  base=dwh
3  host=localhost
4  user=etl
5  port=5432
6  pass=etl
7  log_mis_messages=INFO
8  client_min_messages=WARNING
9  pg_option_client_encoding='win-1250'
10 pg_option_standard_conforming_strings=on
11 pg_option_work_mem=128MB
12 copy_every=15000
13 empty_string="\ "
14 max_parallel_sections=4
15 null=NULL
16 [gpw]
17 table=intf_gpw
18 format=csv
19 datestyle=ymd
20 field_size_limit=512kB
21 field_sep=,
22 quotechar="
23 columns=*
24 skip_header_lines=0
25 truncate=True
26 filename=sesja.txt
27 reject_log=sesja.reject_log
28 reject_data=sesja.reject_data
```

2.2.2 Warstwa pośrednia

Celem warstwy pośredniej jest przekształcenie danych z formatu źródłowego, które znajdują się w tabeli interfejsowej, czyli w tabeli *intf_gpw*, pokazanej na listingu 2.4, na format umożliwiający załadowanie do tabeli docelowej. Szczegółowe cele zależą najczęściej od konkretnego rozwiązania i jego architektury, a także od ładowanych danych. W przykładzie stworzonym na potrzeby naszej pracy będą to:

1. Usuwanie pobranych duplikatów.

2. Zasilanie tabel przejściowych wymiarów. W naszym przykładzie, dla uproszczenia będzie zasilana tabela wymiarów pokazana na listingu 2.2,

3. Zamian klucza naturalnego, którym jest nazwa papieru wartościowego na wartość *integer*, nadawaną przy użyciu sekwencji dla każdej nowej nazwy pojawiającej się w tabeli.

Aby osiągnąć cel wymieniony w podpunkcie 1, musi zostać utworzona tabelka o identycznej strukturze co tabela *intf_gpw*, pokazana na listingu 2.4. Kod realizujące owe zadanie został przedstawiony na listingu 2.6

Listing 2.6: Usuwanie pobranych duplikatów.

```
insert into stg_gpw
(
  nazwa
, data_notowania
, otwarcie
, max
, min
, zamkniecie
, wartosc
)
select
  nazwa
, data_notowania
, otwarcie
, max
, min
, zamkniecie
, wartosc
from intf_gpw i
where not exists
(
  select 1
  from stg_gpw s
  where s.nazwa = i.nazwa
    and s.data_notowania = i.data_notowania
);
```

W podpunkcie 2, zostało wspomnione, że dla uproszczenia przykładu do tabeli wymiarów ładowana jest tylko nazwa tabeli, więc z tego powodu w tym miejscu

zasilamy tabele wymiarów nazw papierów wartościowych, a realizujemy to w sposób zaprezentowany na listingu 2.7. W przypadku gdyby przykład nie został uproszczony to tabelka ta powinna zostać zasilona w warstwie docelowej opisanej na stronie 30 w podrozdziale 2.2.3.

Listing 2.7: Ładowanie danych do tabeli wymiaru – papierów wartościowych.

```
1  insert into npw
2  (
3    nazwa
4  )
5  select distinct
6    nazwa
7  from stg_gpw s
8  where not exists
9  (
10   select 1
11   from npw  k
12   where k.nazwa = s.nazwa
13 );
```

Kolejnym etapem jest nadawanie kluczy sztucznych w hurtowni, poprzez zastąpienie klucza naturlanego (nazwy papierów wartosciowych). W omawianej warstwie dokonuje się również łączenia danych pochodzących z różnych źródeł. Dane, które będą zasilać tabelę faktów w naszej hurtowni danych pochodzą z jednego źródła, z tego powodu tabela przejściowa będzie miała bardzo podobną strukturę do tabel opisanych poprzednio. Tabele tą będziemy nazywać *promo_gpw*, zaprezentowaną na listingu 2.8

Listing 2.8: Struktura tabelki promo_gpw.

```
1  drop table if exists promo_gpw;
2  create table  promo_gpw
3  (
4    npw_id integer
5  , data_notowania date
6  , otwarcie decimal(20, 2)
7  , max decimal(20, 2)
8  , min decimal(20, 2)
9  , zamkniecie decimal(20, 2)
10 , wartosc decimal(20,3)
11 );
```

Wykonanie zadania z podpunktu 3 ze strony 28 zostało zaprezentowane na listingu 2.9.

Listing 2.9: Proces ładowania do tabeli promo_gpw.

```
truncate table promo_gpw;
insert into  promo_gpw
(
  npw_id
, data_notowania
, otwarcie
, max
, min
, zamkniecie
, wartosc
)
select
  n.npw_id
, s.data_notowania
, s.otwarcie
, s.max
, s.min
, s.zamkniecie
, s.wartosc
from stg_gpw s
join npw n  on n.nazwa=s.nazwa
;
```

2.2.3 Warstwa docelowa

Celem warstwy docelowej jest załadowanie danych do tabel faktów i wymiarów, jak również udostępnianie ich dla użytkowników korzystających z hurtowni danych.

Dane, które będą zasilać tabelę faktową w naszej hurtowni są dziennymi danymi podsumowującymi cały dzień notowań ciągłych na giełdzie. Tego typu dane, z uwagi na swój charakter, nie zmieniają się po załadowaniu, dlatego zostanie pominięty UPDATE danych. Wykona jedynie zostanie operacja INSERT z danych przygotowanych w tabel *promo_gpw*, do tabeli faktów *gpw*, co zostało pokazane na listingu 2.10

Listing 2.10: Proces ładowania danych do tabeli gpw.

```
insert into  gpw
(
  npw_id
, data_notowania
, otwarcie
```

```
6   , max
7   , min
8   , zamkniecie
9   , wartosc
10  )
11  select
12    npw_id
13  , data_notowania
14  , otwarcie
15  , max
16  , min
17  , zamkniecie
18  , wartosc
19  from promo_gpw p
20  where not exists
21  (
22    select
23      1
24    from gpw t
25    where
26          t.npw_id=p.npw_id
27    and   t.data_notowania=p.data_notowania
28  );
```

Rozdział 3

Temat rozdziału

3.1 Gramatyka

3.1.1 Języki formalne

Alfabet lub *słownik* oznaczają dowolny niepusty, skończony zbiór symboli. *Słowem* nazywamy ciąg symboli *alfabetu* o skończonej długości. Jeżeli słowo jest długości zero, to nazywamy go *słowem* pustym, które będziemy oznaczać przez małą literę grecką epsilon (ϵ). Synonimami *słowa* są *napis* i *zdanie*. [5]

Przykładami *alfabetu* mogą być:

- zbiór niektórych liter alfabetu polskiego,
- zbiór składający się z symbolu zera i jedynki,
- zbiór liczb całkowitych i zbiór symboli kodowania znaków UTF-8,
- zbiór { AA, BB }, w którym AA i BB są traktowane jako jeden symbol.

Przykładami *słów* dla *alfabetu* liczb całkowitych, który składa się ze znaków {+,-,.(kropka),0,1,2,3,4,5,6,7,8,9} mogą być: ϵ, 0, 1, 01, 10, 090, -1001, +098, -121, 100, +41, +0000010, -000011 itd.

Językiem formalnym (językiem) nazywamy podzbiór zbioru wszystkich słów nad skończonym alfabetem.

Przykładami języków formalnych mogą być:

- zbiór pusty, oznaczany jako Ø,
- zbiór zawierający tylko słowo puste $\{\epsilon\}$
- zbiór programów, które po skompilowaniu i uruchomieniu zawieszą dany komputer,
- zbiór wszystkich poprawnie napisanych nierówności.

W tabeli 3.1 zostały zdefiniowane prawa na językach.

Tabela 3.1: Definicja operacji na językach.

Termin	**Definicja**
suma L i M zapisywana $L \bigcup M$	$L \bigcup M = \{s : \quad s \in L \quad lub \quad s \in M\}$
złączenie L i M zapisywane LM	$LM = \{st : \quad s \in L \quad oraz \quad t \in M\}$
podnoszenie do potęgi	$L^0 = \{\epsilon\}$ $L^i = L^{i-1}L$
domknięcie L zapisywane $L*$	$L* = \bigcup_{i=0}^{+\inf} L^i$ $L*$ oznacza "zero lub więcej złączeń" L

Tabela 3.1 – Kontynuacja tabeli z poprzedniej strony.

Termin	**Definicja**
dodatnie domknięcie L zapisywane $L+$	$L+ = \bigcup_{i=1}^{+\inf} L^i$ $L*$ oznacza "co najmniej jedno złączenie" L

Rozważmy przykład. Niech L będzie zbiorem małych i dużych liter, a M zbiorem cyfr. Ponieważ symbole mogą być traktowane jako słowa o długości jeden, to zbiory L i M są językami skończonymi. Poniżej znajduje się kilka przykładów nowych języków utworzonych za pomocą L i M przy zastosowaniu operatorów zdefiniowanych w tabeli 3.1.

1. $L \bigcup M$ jest zbiorem liter i cyfr.
2. ML jest zbiorem słów składających się z cyfry i występującej po niej litery.
3. $L*$ jest zbiorem wszystkich słów złożonych z liter, włączając w to słowo puste.
4. $L+$ jest zbiorem wszystkich słów złożonych z liter, bez słowa pustego.
5. $L(L \bigcup M)*$ jest zbiorem wszystkich słów złożonych z liter i cyfr, zaczynających się od litery.

3.1.2 Gramatyka formalna

Języki formalne opisywane są przez *gramatyki formalne*, to jest uporządkowane czwórki (T,N,P,S), gdzie[5, 10]:

- T jest skończonym zbiorem symboli terminalnych (inaczej alfabetem),
- N jest skończonym zbiorem symboli nieterminalnych, przy czym, N$\bigcap$T=$\emptyset$,
- P jest skończonym zbiorem reguł produkcji postaci R_1: R_2;, gdzie R_1 i R_2, to symbole które reprezentują ciągi, o skończonej długości, *symboli terminalnych* i *symboli nieterminalnych*, przy czym, symbol R_1 musi zawierać co najmniej jeden symbol nieterminalny.

- S jest symbolem startowym i należy do zbioru symboli nieterminalnych. Od symbolu startowego zaczyna się wyprowadzanie wszystkich słów danego *języka formalnego.*

Rozpatrzmy przykład gramatyki G, która opisuje język akceptujący słowa postaci $\{ ({}^{n})^{n} : n \in \mathbb{N} \}$, Gramatyka G ma postać:

G = ({ (,) } , {S} , {S:(S), S: ϵ } ,S)

Słowo ((())) możemy wyprowadzić: S: (S) : ((S)) : (((S))) : ((()))

Do tak opisanego języka należy każde słowo, dla którego możliwe jest wyprowadzenie (utworzenie), przy użyciu reguł produkcji. Jeżeli nie jest możliwe wyprowadzenie słowa to nie należy do języka.

3.1.3 Klasyfikacja języków

Avram Noam Chomsky badał języki formalne, czyli podzbiór wszystkich słów nad skończonym alfabetem wyniku tych badań w 1956r. podał klasyfikację języków formalnych, która powszechnie uznawana jest za standard.

Hierarchia ta składa się z czterech klas [5, 8, 9]:

- języki typu 3 - regularne, są to języki opisywane za pomocą gramatyki regularnej, w której reguły produkcji mogą mieć postać:
  - N:TN
  - N:T
  - N:N
  - N:ϵ

  gdzie N jest symbolem terminalnym, a T symbolem nieterminalnym.
- języka typu 2 - bezkontekstowe, są to języki opisane za pomocą gramatyk bezkontekstowych, w której reguły produkcji mogą mieć postać:

– N:C

gdzie N jest symbolem nieterminalnym, a C to symbole które reprezentują ciągi, o skończonej długości, symboli terminalnych i symboli nieterminalnych,

- języka typu 1 - kontekstowe, opisywany jest przez gramatykę kontekstową, w której lewa strona produkcji nie może zawierać mniej symboli terminalnych i nieterminalnych niż prawa strona,
- języka typu 0 - rekurencyjnie przeliczalne, opisywany przez gramatykę rekurencyjnie przeliczalną, w której reguły produkcji nie zostały ograniczone.

Mówimy, że język należy do danej klasy wtedy, gdy jest możliwe zbudowanie gramatyki, która generuje dany język, a reguły produkcji nie wykraczają poza ograniczenia dla danej klasy.

3.1.4 Wyrażenia regularne

Wyrażenie regularne nad alfabetem Σ nazywamy ciąg znaków ϵ,), (, *, + oraz symboli z alfabetu Σ następującej postaci:

1. ϵ (słowo puste) jest wyrażeniem regularnym,
2. wszystkie symbole należące do alfabetu są wyrażeniami regularnymi,
3. niech r i s będą wyrażeniami regularnymi, to są nimi również:
   - $r|s$ (suma),
   - rs (łączność) ,
   - $r*$ (domknięcie),
   - $r+$ (dodatnie domknięcie)
   - (r) (grupowanie).
4. wszystkie wyrażenia regularne są postaci opisanej w punkcie 1 – 3.

Wyrażenie regularne r służy do opisywania języka regularnego, który będziemy oznaczać $L(r)$. Język opisywany przez wyrażenie regularne ma następującą postać:

- $\mathrm{L}(\epsilon) = \{\epsilon\}$
- $\mathrm{L}(a) = \{a\}$, gdzie a jest dowolnym symbolem z alfabetu Σ

Załóżmy, że r i s jest wyrażeniami regularnymi oznaczającymi języki $L(r) = M$ i $L(s) = L$, wtedy

- $r|s = M \bigcup L$
- $rs = ML$
- $r* = M*$
- $r* = M+$

Operatory na językach zostały opisane w tabeli 3.1, na stronie 34. Rozważmy przykłady. Niech alfabet Σ będzie zbiór liter języka polskiego oraz cyfr i znaków matematycznych:

1. wyrażenie regularne a|b oznacza zbiór {a,b },
2. wyrażenie regularne (a|b)* oznacza zbiór { ϵ, a, b, aa, ab, bb, ba, aaa, ... },
3. wyrażenie regularne (a|b)|(a|b) oznacza zbiór {aa,ab,ba,bb},
4. wyrażenie regularne $(-|+)((1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*)|0$ oznacza zbiór wszystkich liczb całkowitych.

3.2 Język wysokiego poziomu

Językiem wysokiego poziomu *(ang. High-level language)* nazywamy język, którego składnia i słowa kluczowe języka ułatwiają użytkownikom napisanie programu, bądź skrypty, jak również powinien być on wolny od zależności sprzętowych i systemowych.

Celem niniejszej pracy jest napisanie języka wysokiego poziomu, który będzie językiem interpretowany. Na rysunku ... został przedstawione etapy budowania, są to:

1. Analizator leksykalny — czynności związane z analizą leksykalną, które została przedstawiona w podrozdziale .....
2. Analiza składniowa — została przedstawiona w podrozdziale .....
3. Analiza semantyczna — została przedstawiona w podrozdziale .....,
4. Wykonywanie instrukcji — Po przekazaniu polecenia jest ono wykonywane, zgodnie z założeniem programisty lub zwracany jest błąd.

3.2.1 Analiza leksykalna

Zadaniem analizatora leksykalnego jest tworzenie symboli leksykalnych z przesłanego przez użytkownika strumienia znaków na wejście. Symbole te są następnie przesyłane do analizatora składniowego. Symbole leksykalne tworzone są na podstawie tablicy wyrażeń regularnych i odpowiadającym i zadaniom, czyli jeżeli przesłany strumień znaków należy do n-tego wyrażenie regularne to zostanie wykonana powierzone mu zadanie, którym na przykład może być [5]:

- Nie wykonanie żadnego działania (strumień znaków zostanie pominięty)
- przesłanie symbolu leksykalnego do analizatora
- przesłanie symbolu leksykalnego wraz z ciągiem znaków, które należą do wyrażeniami
- przesłanie symbolu leksykalnego wraz z modyfikowanym ciągiem znaków

Podczas analiz leksykalne możemy zaprojektować analizator leksykalny, aby mógł wykrywać następujące błędy:

- znaki pojawiające się na wejściu nie stanowią żadnego symbolu leksykalnego
- jeśli na wejściu pojawi się znak nie nieobsługiwany
- gdy istniej możliwość przewidzenia błędnych ciągów znaków odpowiadające jakiemuś symbolowi leksykalnemu. Np. operator mniejszy równy “>=”, często jest pisany niepoprawnie w następujący sposób “=<”

3.2.2 Analiza leksykalna

Analizator składniowy otrzymuje od analizatora ciąg symboli leksykalnych, które traktowane są jak symbole terminalne w gramatyce. Zadaniem analizatora składniowego, jest grupowanie symboli leksykalnych i tworzenie drzewa składniowego zgodnie z regułami gramatyki. Wynikiem działania analizatora składniowego jest drzewo składniowe lub wyświetlenie komunikatów o błędach.

Drzewem składniowy *(ang. parse tree)* nazywamy hierarchiczną strukturą danych, w której węzły odpowiadają:

- symbolu leksykalnemu, czyli skończonemu ciągowi symboli terminalnych,
- symbolu nieterminalnemu.

Najpopularniejszymi metodami wykorzystywanymi w analizatorach składniowych dla gramatyk są to metody zstępujące i wstępujące. W metodzie zstępującej drzewo jest tworzone od korzenia do liści, a wstępującej odwrotnie od liści do korzenia. W obu metodach wejście jest przeglądane od lewej strony.

Analizator składniowy może wykryć błędy w przypadku, gdy strumień symboli leksykalnych nie jest akceptowany przez gramatykę języka. Jeżeli zostanie wykryty błąd to analizator składniowy powinien próbować odzyskać kontrolę w celu wykrycia kolejnych błędów i poinformować o nich użytkownika. Poniżej przedstawiono następujące strategie[5]:

Tryb paniki — Po natrafieniu na błąd, analizator usuwa symbole leksykalne z wejścia, aż natrafi na symbol z ustalonego zbioru od których może rozpocząć się dalsze poszukiwanie błędów.

Poziom frazy — Pozwala zmienić lub dodać symbole leksykalne, które umożliwią dalsze dopasowania,

Produkcja dla błędu — Jeżeli wiemy, gdzie pojawiają się najczęściej błędy, to możemy dopisać odpowiednią regułę produkcji w gramatyce.

3.2.3 Analiza semantyczna

Kolejnym etapem z rysunku ... na stronie jest analiza semantyczna, której zadaniem jest sprawdzenie, czy każdy identyfikator jakiegoś działania nazywany operatorem ma odpowiednią ilość składników nazywane argumentami. Zadaniem analizatora semantycznego jest również sprawdzenie, czy każdy argument ma odpowiedni typ danych. Wejściem do analizy semantycznej jest poprawnie zbudowane drzewo składniowe według gramatyk. [5]

Błędy semantyczne możemy sklasyfikować w następujący sposób [7]:

Błędy krytyczne — błędy uniemożliwiające dalszą analizę

Błędy tworzące nieoczekiwany — istnieją błędy, które nie zostaną przechwycone i zostanie zwrócony nie oczekiwany wynik. Bardzo, częstym przykładem tego typu jest program napisany w języku C, który czyta nie ze swojej pamięci,

Niepotrzebnie rozbudowane polecenia — są to polecenia, które zostały podane nadmiarowe dane.

Listingi kodu

Bibliografia

[1] Chris Todman, *Projektowanie Hurtowni Danych. Wspomaganie zarządzania relacjami z klientem*, Wydawnictwa HELION 2011.

[2] W.H. Inmon, *Building the Data Warehouse, Fourth Edition*, Wydawnictwo Wiley Publishing, Inc. 2005

[3] Kimball R., Ross M., *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, Wydawnictwo John Wiley and Sons, Inc. 2004

[4] Rainardi V., *Building a data warehouse with exmaples in SQL server*, Wydawnictwo Springer-Verlag New York, Inc. 2008

[5] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman: *Kompilatory*, Wydawnictwa Naukowo-Techniczne, Warszawa 2002

[6] `http://etl-tools.info/pl/bi/hurtownia_danych_schemat-gwiazdy.htm`

[7] `http://dbs.informatik.uni-halle.de/sqllint/semerr_techrep.pdf`

[8] `http://en.wikipedia.org/wiki/Chomsky\_hierarchy`

[9] `http://pl.wikipedia.org/wiki/Hierarchia\_Chomsky'ego`

[10] `http://en.wikipedia.org/wiki/Formal_grammar`