

AKADEMIA PEDAGOGICZNA w KRAKOWIE

Im. Komisji Edukacji Narodowej

WYDZIAŁ MATEMATYCZNO-FIZYCZNO-

TECHNICZNY

INSTYTUT TECHNIKI

TOMASZ RUTKOWSKI

TECHNIKI INTEGRACJI BAZ DANYCH I SYNCHRONIZACJI

PRZESYŁANIA DANYCH MIĘDZY NIMI

**Praca magisterska napisana pod kierunkiem
dr hab. inż. Kazimierza Jaracza, prof. nadzw. AP**

Kraków 2007

Spis treści

WSTĘP	4
1. SYSTEMY BAZ DANYCH.....	7
1.1. Budowa baz danych	8
1.2. Krótka historia baz danych	10
1.3. Rodzaje baz danych	10
1.4. Podstawowe modele danych	13
1.4.1. Model związków encji	14
1.4.2. Relacyjny model danych	15
1.4.3. Obiektowy model danych	16
1.4.4. Semistrukturalny model danych	18
1.4.5. Hierarchiczny model danych	19
1.4.6. Sieciowy model danych	21
2. INTEGRACJA BAZ DANYCH.....	23
2.1. Zapewnienie integralności danych.....	23
2.1.1. Integralność danych	23
2.1.2. Transakcje	30
2.2. Integracja bazy danych z programami	34
2.2.1. Osadzony SQL	35
2.2.2. Dynamiczny SQL	35
2.2.3. Interfejs programów użytkowych (API)	36
2.3. Wrapper (osłona, translator)	37
3. SYNCHRONIZACJI DANYCH POMIĘDZY BAZAMI DANYCH ...	38
3.1. Systemy rozproszone	38
3.2. Rozproszone bazy danych	40
3.2.1. Transakcje rozproszone	42
3.2.2. Replikacja danych	44
3.3. System typu klient-serwer	47
3.4. Jednorodna rozproszona baza danych.....	48
3.5. Niejednorodna rozproszona baza danych	48
3.6. Mediatory	49
3.7. Federacyjne bazy danych	50
3.8. Hurtownie danych	52
4. ALGORYTM SYNCHRONIZACJI PRODUKTÓW	54
4.1. Założenia	55
4.2. Możliwe sposoby synchronizacji	58
4.3. Schematy baz danych	59
4.3.1. Schemat bazy danych programu Subiekt GT	59
4.3.2. Schemat bazy danych sklepu osCommerce	61
4.4. Algorytm synchronizacji produktów	63
4.4.1. Nawiązanie połączeń z bazami danych	65
4.4.2. Synchronizacja stawek podatkowych	65
4.4.3. Synchronizacja grup/kategorii	66
4.4.4. Synchronizacja produktów	68
4.4.5. Zakończenie połączeń z bazami danych	71

5. IMPLEMENTACJA ALGORYTMU	72
5.1. Uzasadnienie wyboru środowiska	72
5.2. Implementacja	72
5.3. Testy wydajności	74
PODSUMOWANIE	81
BIBLIOGRAFIA.....	82
ANEKS	84
GNU FREE DOCUMENTATION LICENSE.....	85
SPIS RYSUNKÓW	93
SPIS TABEL.....	95
OŚWIADCZENIE	96

Wstęp

W dobie społeczeństwa informacyjnego dostęp do informacji ma kluczowe znaczenie dla społeczeństwa i organizacji. Wraz z rosnącą ilością informacji nasiliły się problemy z brakiem odpowiednio wydajnych metod ich przechowywania i udostępniania. Rozwiązaniem problemu okazało się stworzenie systemów baz danych. Jednak łatwość przechowywania informacji w bazach danych i mnogość rozwiązań stworzyła kolejny problem jakim jest brak integralności danych z różnych źródeł. Ponieważ nie istnieje jeden uniwersalny model danych, a w konsekwencji baza danych problem integralności źródeł danych jeszcze się nasilił. W obecnej chwili, gdy bazy danych stały się nieodłącznym elementem systemów informatycznych wiele organizacji boryka się z problemem zaszłości baz danych. Oznacza to że na około bazy danych narosła taka ilość systemów informatycznych, iż niemożliwe jest wyłączenie lub zmiana tej bazy danych. Mogłoby to sparaliżować działalność organizacji i wygenerować koszty, które mogłyby przyczynić się do bankructwa organizacji. Dlatego integracja baz danych jest kluczowym czynnikiem do poprawnego działania rozwijających się organizacji, które muszą wdrażać nowe rozwiązania by dobrze prosperować na rynku i w społeczeństwie.

Należy jednak pamiętać, że wraz z cyfrowymi metodami gromadzenia i przetwarzania danych zaistniała potrzeba ochrony prawnej tych danych. W Polsce kwestię tę reguluje ustawa o ochronie danych osobowych [13] oraz ustawa o ochronie baz danych [14] i powinno się ją uwzględnić w czasie ich tworzenia.

Cel pracy i zakres pracy

Praca ta ma na celu przedstawienie technik integracji baz danych oraz programów użytkujących bazy danych. Ponadto przedstawienie sposobów synchronizowania danych w rozproszonych bazach danych. Z uwagi na największą popularność baz relacyjnych i wiążącą się z tym zaszłością skupię się głównie na tego rodzaju bazach danych.

Pierwszy rozdział jest rozdziałem wprowadzającym. Przedstawione zostały w nim systemy baz danych na poziomie koncepcyjnym i różne modele oraz podejścia do ich budowy. W kolejnych podrozdziałach została opisana budowa baz danych, ich krótka historia, rodzaje baz danych i ich podstawowe modele.

Rozdział drugi traktuje o integracji baz danych. W pierwszym podrozdziale przedstawione zostały metody zapewnienia integralności danych na poziomie pojedynczej bazy danych. W drugim podrozdziale integracja baz danych z programami ukierunkowane

jest na sposoby łączenia języka bazy z językiem programu. Trzeci podrozdział mówi wrapperach jako istotnym elemencie dostosowującym język zapytań do języka wewnętrznego źródła danych.

Treścią trzeciego rozdziału są sposoby synchronizacji danych pomiędzy bazami danych. W początkowych podrozdziałach zostały opisane architektury systemów rozproszonych, rozproszone bazy danych z uwzględnieniem transakcji i replikacji. Pozostałe podrozdziały opisują typy rozproszonych baz danych i relacje występujące między nimi.

Wkład własny autora

Wkładem własnym autora jest opracowanie i implementacja algorytmu synchronizującego produkty z programu „Subiekt GT” firmy Insert [15] służącego do obsługi firm oraz z darmowego sklepu internetowego osCommerce [16]. Algorytm ten i cała niniejsza praca jest udostępniana na licencji GNU Free Documentation License [17] w celu dydaktycznej pomocy wszystkim zainteresowanym. Skrypt będący implementacją algorytmu udostępniony został na wolnej licencji GNU General Public License [18].

Czwarty rozdział opisuje algorytm synchronizujący produkty z Subiekta GT do sklepu internetowego osCommerce. W pierwszym podrozdziale znajduje się opis założeń jakie musi on spełniać. W drugim możliwe sposoby synchronizacji produktów przy pominięciu części założeń. W trzecim przedstawiam schemat replikowanych baz danych koncentrując się na istotnych relacjach i atrybutach tych relacji. W podrozdziale czwartym prezentuję opracowany przeze mnie algorytm.

Piąty rozdział zawiera omówienie implementacji stworzonego przeze mnie algorytmu, którego źródła wraz z dokumentacją znajdują się na dołączonej do pracy płycie CD. W kolejnych podrozdziałach znajduje się uzasadnienie wyboru środowiska, implementacja oraz testy wydajności.

Copyright © Tomasz Rutkowski

Udziela się zezwolenia do kopiowania, rozpowszechniania i/lub modyfikacji tego dokumentu zgodnie z zasadami Licencji GNU Wolnej Dokumentacji w wersji 1.2 lub dowolnej późniejszej opublikowanej przez Free Software Foundation [19]; nie zawiera Sekcji Niezmiennych, bez Tekstu na Przedniej Okładce, bez Tekstu na Tylnej Okładce. Kopia licencji załączona jest w rozdziale zatytułowanym GNU Free Documentation License [17].

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Wszystkie znaki firmowe bądź towarowe występujące w tekście należą do ich właścicieli i użyte zostały tylko z myślą o tej publikacji, bez zamiaru naruszenia tych znaków.

Kontakt: majorserwis@gmail.com

1. Systemy baz danych

Zanim jednak przystąpię do prezentowania rozwiązań z zakresu integracji baz danych należy zdefiniować samo pojęcie bazy danych. W literaturze fachowej znajdują się ich różne definicje.

Jedną z nich jest definicja zaprezentowana przez Paula Benon-Davies.

„Bazę danych możemy uważać za zbiór danych, których zadaniem jest reprezentowanie pewnego obszaru analizy. Dane to fakty. Dana, jednostka danych, jest jednym symbolem lub zbiorem symboli, którego używamy, aby reprezentować jakąś rzecz. Fakty same w sobie nie mają znaczenia. Aby były użyteczne, muszą być zinterpretowane. Zinterpretowane dane to informacje. Informacja to dane z przypisaną im semantyką – znaczeniem.” [1]

Encyklopedia PWN z kolei definiuje bazę danych jako:

„Baza danych jest to zbiór wzajemnie powiązanych danych, przechowywanych w pamięci komputerów i wykorzystywanych przez programy użytkowe instytucji lub organizacji wraz z oprogramowaniem umożliwiającym definiowanie, wykorzystywanie i modyfikowanie tych danych.” [20]

Definicja b.d. przedstawiona w encyklopedii Wikipedia prezentuje się następująco:

„Baza danych to zbiór informacji zapisanych w ściśle określony sposób w strukturach odpowiadających założonemu modelowi danych. W potocznym ujęciu obejmuje dane oraz program komputerowy wyspecjalizowany do gromadzenia i przetwarzania tych danych. Program taki (często zestaw programów) nazywany jest systemem zarządzania bazą danych. W ścisłej nomenklaturze baza danych oznacza zbiór danych, który zarządzany jest przez system DBMS.” [21]

Natomiast Molina, Ullman, Widom określają bazę danych jako:

„Baza danych jest zbiorem danych zarządzanym przez system zarządzania bazą danych. DBMS jest to narzędzie przeznaczone do tworzenia i efektywnego zarządzania dużymi zbiorami danych oraz bezpiecznego ich przechowywania przez długi czas.” [2]

Po przeanalizowaniu definicji baz danych do niniejszego opracowania przyjmuję za wyjściową następującą definicję: „*Baza danych jest zbiorem danych przechowywanym w sposób zorganizowany w pamięci komputera. Jest ona nadzorowana przez system zarządzania bazą danych (ang. DataBase Management System, DBMS).*”

1.1. Budowa baz danych

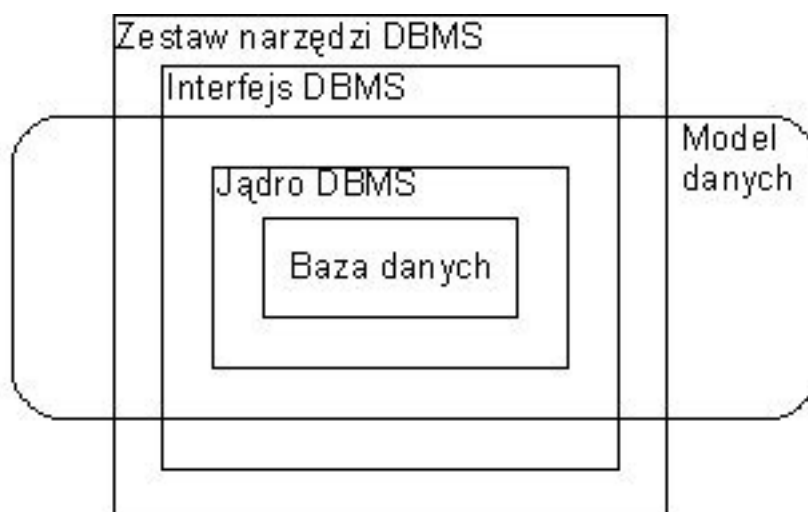
Każdą bazę danych można podzielić na kilka części odpowiedzialnych za różne rodzaje działań. Pomocny tu będzie rysunek 1.1 przedstawiający budowę baz danych.

Jądro - zajmuje się operacjami związanymi z dostępem do fizycznego zapisu danych na urządzeniach pamięci masowej.

Model danych – jest zbiorem zasad regulujących sposób przechowywania danych, dozwolone operacje i spójność bazy danych.

Interfejs – udostępnienia środki programistyczne służące do aktualizacji/przetwarzania danych.

Zestaw narzędzi – są to dodatkowe programy ułatwiające użytkowanie i zarządzanie bazą danych.



Rys. 1.1. Budowa bazy danych

Należy tutaj wspomnieć o tak ważnym elemencie jakim jest System Zarządzania Bazą Danych (ang. *DataBase Management System*, DBMS). Jest to obecnie jeden z najbardziej skomplikowanych programów komputerowych, który musi posiadać następujące cechy [21]:

- gromadzenie, utrzymywanie i administrowanie trwałymi i masowymi zbiorami danych,
- zapewnienie spójności i bezpieczeństwa danych,
- zapewnienie dostępu do danych,

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

- udostępnienie środków programistycznych służących do aktualizacji/przetwarzania danych (API dla popularnych języków programowania),
- umożliwienie jednoczesnego dostępu do danych dla wielu użytkowników,
- zapewnienie środków pozwalających na regulację dostępu do danych (autoryzację),
- możliwość odtworzenia zawartości bazy danych po awarii,
- zarządzanie katalogami, schematami i innymi metadanymi,
- optymalizacje zajętości pamięci oraz czasu dostępu,
- możliwość pracy lub współdziałania w środowiskach rozproszonych.

I. Możliwości dostarczone przez DBMS użytkownikom [2]:

1. Pamięć trwała – DBMS udostępnia pamięć do przechowywania wielkiej liczby danych istniejących niezależnie od procesów, które z nich korzystają.
2. Interfejs programisty – DBMS udostępnia użytkownikowi lub aplikacji język zapytań, dzięki któremu uzyskuje się dostęp do danych oraz modyfikuje je.
3. Zarządzanie transakcjami – DBMS zapewnia jednoczesny dostęp do danych, tzn. wiele różnych procesów (zwanymi *transakcjami*) jednocześnie uzyskuje dostęp.

II. Oczekiwania względem DBMS [2]:

1. Umożliwienie użytkownikowi utworzenia nowej bazy danych i określenia jej schematu (logicznej struktury danych) za pomocą specjalizowanego języka *definiowania danych* (ang. *data definition language*).
2. Udostępnienie użytkownikowi możliwości tworzenia zapytań (ang. *query*) o dane (czasem określanych po polsku również mianem kwerend) oraz aktualizowania danych, za pomocą odpowiedniego języka nazywanego *językiem zapytań* (ang. *query language*) lub *językiem operowania danymi* (ang. *data manipulation language*).
3. Zapewnienie możliwości przechowywania ogromnej ilości danych mierzonej w gigabajtach lub nawet większych jednostkach, przez długi czas, chroniąc je przed przypadkowym lub niepowołanym dostępem, a także umożliwiając efektywny dostęp do danych z poziomu języka zapytań i operacji na danych.
4. Sterowania jednoczesnym dostępem do danych przez wielu użytkowników, z zapewnieniem bezkolizyjności oraz ochrony danych przed przypadkowym uszkodzeniem.

1.2. Krótka historia baz danych

Aby zrozumieć ewolucję baz danych warto prześledzić ich historię. Pierwsze systemy zarządzania bazami danych pojawiły się pod koniec lat 60-tych. Były one oparte o przetwarzanie plików. Posiadały one pewne właściwości wymienione w punkcie (II.3). Jednak braki sterowania jednoczesnym dostępem do danych, awarie i inne czynniki (II.4, II.2, II.1) nie czyniły z niego dobrego narzędzia. Największym brakiem wspomnianych systemów był brak języka zapytań wysokiego poziomu. Pierwsze systemy baz danych powstałe w oparciu o system plików w zdecydowanej większości posiadały hierarchiczny model danych, inaczej zwany drzewiastym oraz sieciowy nazywany również grafowym. Drugi z wymienionych modeli został uznany za standard w raporcie Komitetu ds. Systemów i Języków Danych (ang. Committee on Data Systems and Languages, CODASYL) [8] z roku 1971. Największe zmiany w systemach baz danych zaszły po ogłoszeniu w 1970 roku artykułu Teda Codd'a [22]. Pierwsze systemy baz danych oparte na modelu relacyjnym pojawiły się w latach 80-tych i pierwszej połowie lat 90-tych. Kolejnym przełomowym modelem danych był obiektowy model danych. W 1989 roku został opracowany ideologiczny manifest, który zaowocował w drugiej połowie lat 80-tych powstaniem obiektowych baz danych. OBD po mimo swoich licznych zalet do dzisiaj nie uzyskały mocnej pozycji na rynku. W celu połączenia zalet obiektowego i relacyjnego modelu danych został stworzony system hybrydowy zwany obiektowo-relacyjnym modelem danych. Obecnie prawie wszyscy producenci baz relacyjnych oferują także ORDBMS (ang. *Object-Relational DataBase Management System*). Obiektowo relacyjny system baz danych to nie jedyna hybryda funkcjonująca na rynku. Wraz z upowszechnieniem się XML-a (ang. eXtensible Markup Language – rozszerzalny język znaczników) wzrosło zapotrzebowanie na przechowywanie plików w tym formacie. Producenci baz danych stosowali różne metody w celu lepszej obsługi XML-a jednak przełomowa okazała się hierarchiczno-relacyjna baza danych pozwalająca na rozszerzenie zapytań o język XQuery i XPath.

1.3. Rodzaje baz danych

Można wyróżnić wiele rodzajów (podziałów) baz danych. Dla niniejszego opracowania przyjąłem podział baz danych ze względu na używany model danych.

Tak więc bazy danych można podzielić na:

I. Bazy proste

1. Bazy kartotekowe

2. Hierarchiczne bazy danych
 3. Sieciowe bazy danych
- II. Bazy złożone
1. Relacyjne bazy danych
 2. Obiektowe bazy danych
 3. Strumieniowe bazy danych
 4. Temporalne bazy danych
 5. Dedukcyjne bazy danych
 6. Wielowartościowe bazy danych
 7. Rozproszone bazy danych
- III. Bazy hybrydowe
1. Relacyjno-obiektowe bazy danych
 2. Relacyjno-hierarchiczne bazy danych

Bazy kartotekowe

W bazach kartotekowych każda tablica (zbiór) danych jest samodzielnym zbiorem informacji i nie może współpracować z innymi zbiorami. Takie bazy są uściślone do jednego konkretnego zastosowania. Najczęściej są stosowane w programach typu: spis książek, książka kucharska, książka telefoniczna.

Hierarchiczne bazy danych

Hierarchiczne bazy danych są to bazy używające hierarchicznego modelu danych. Przechowywane dane są organizowane w drzewiastą strukturę, w której dane najczęściej powiązane są w relacji jeden do wielu. Najbardziej znaną hierarchiczną bazą danych jest IMS (Information Management System) firmy IBM.

Sieciowe bazy danych

Sieciowe bazy danych są uważane za następcę hierarchicznych baz danych. Zostały stworzone w celu rozwiązania problemu o definicji relacji wiele-do-wiele w bazie hierarchicznej bez powtarzania poszczególnych wartości w ramach relacji. Dodatkowo związek nadrzędny-podrzędny został zastąpiony typem kolekcji (*set*).

Relacyjne bazy danych

Obecnie są najczęściej używanymi bazami danych. Swoją popularność zawdzięczają zdyscyplinowanemu traktowaniu danych poprzez zastosowanie ścisłych metod i teorii zbiorów. Było to nowością w stosunku do wcześniej używanych baz danych. Nie zaprzeczalną zaletą tych baz danych jest także język zapytań strukturalny (SQL ang. *Structured Query Language*), który został zaimplementowany (z niewielkimi różnicami i rozszerzeniami) w każdej relacyjnej bazie danych. Najbardziej znanymi bazami tego typu są: Oracle, MS SQL, MySQL, Postgresql. W dalszej części pracy w przykładach dotyczących tego typu bazy danych będę się posługiwał językiem SQL.

Obiektowe bazy danych

Obiektowe bazy danych powstały jako odpowiedź na obiektowe języki programowania. Przed ich powstaniem istniało niedopasowanie modelu używanego przez programistę aplikacji obiektowych i modelu bazy danych. Dzięki dopasowaniu modeli dane nie muszą być konwertowane przed zapisem do bazy danych i po odczycie danych co pozwala ograniczyć koszty i błędy programistyczne. Bazy typu obiektowego są obecnie największym konkurentem baz relacyjnych. Najbardziej znanymi bazami tego typu są: O2, ObjectStore, Objectivity oraz ONTOS.

Strumieniowe bazy danych

Strumieniowe bazy danych są to bazy przechowujące zbiory strumieni danych. Mają one na celu przetwarzanie ciągłych, nieskończonych, zmiennych w czasie strumieni danych. Mogą one przetwarzać ciągłe zapytania często przetwarzane przez cały okres działania aplikacji współpracujących z bazą. Potrafią one dostarczać aktualne wyniki analiz czy innych rezultatów przetwarzania danych [23]. Obecnie są to bazy prototypowe.

Temporalne bazy danych

Temporalne bazy danych są odmianą bazy relacyjnej, w której każdy rekord posiada informacje o czasie wprowadzenia lub czasie ważności danych. Zazwyczaj operacje usuwania starych danych lub przenoszenia ich do archiwum są zautomatyzowane.

Dedukcyjne bazy danych

Dedukcyjne bazy danych są to bazy oparte na logice formalnej. Poszerzają one model relacyjny o idee logicznego programowania. Mają one większe znaczenie akademickie niż praktyczne.

Wielowartościowe bazy danych

Wielowartościowe bazy danych są niezależne koncepcyjnie od relacyjnych baz danych chociaż powstały mniej więcej w tym samym okresie czasu. Najprościej można je określić jako relacyjną bazę danych, która zamiast wartości atomowych może przechowywać w atrybucie wiele wartości nie koniecznie tego samego typu.

Rozproszone bazy danych

Rozproszone bazy danych są to bazy danych, które fizycznie istnieją na co najmniej dwóch komputerach i są traktowane jako jedna logiczna całość. Zmiana wykonana na jednym fizycznym komputerze jest natychmiast uwzględniana na pozostałych. Stosuje się je ze względu na zwiększoną wydajność przetwarzania danych.

Relacyjno-obiektowe bazy danych

Relacyjno-obiektowe bazy danych powstały jako odpowiedź producentów baz relacyjnych na potrzebę przechowywania danych bardziej złożonych niż typy podstawowe. Wykorzystują one sprawdzony model relacyjny rozszerzony o pewne elementy modelu obiektowego. W zależności od producenta są to różne implementacje.

Relacyjno-hierarchiczne bazy danych

Relacyjno-hierarchiczne bazy danych łączą model relacyjny z hierarchicznym w celu lepszej obsługi dokumentów XML przy jednoczesnym zachowaniu zalet baz hierarchicznych. Chociaż istnieją bazy natywne dla XML-a to nie przyjęły się one ze względu na zbyt indywidualny charakter.

1.4. Podstawowe modele danych

W bazach danych stosuje się różne modele danych, które wymuszają inne podejście do projektowania bazy danych oraz jej użytkowania. Integralność baz danych nie da się rozpatrywać w oderwaniu od modeli danych ponieważ już sam model zaimplementowany w bazie danych integruje dane na poziomie bazy danych.

Uściślając model danych jest to zbiór reguł określających strukturę danych, zapewniających poprawność bazy danych (integralność danych) oraz sposób operowania danymi.

Wyróżniamy następujące modele danych:

1. Model związków encji
2. Relacyjny model danych
3. Obiektowy model danych
4. Model semistrukturalny
5. Klasyczne modele danych
 - a. Hierarchiczny model danych
 - b. Grafowy model danych

1.4.1. Model związków encji

Model związków encji (E/R) jest tradycyjną i popularną notacją projektową baz danych. W modelu tym istotnie elementy danych (encje) oraz połączenia między nimi (związki) są przedstawione graficznie jako diagramy związków encji. Model ten jest stosowany do projektowania baz danych ułatwiając projektantowi myślenie o strukturze bazy danych. Jednak on sam nie jest zaimplementowany w żadnej bazie danych. W celu utworzenia bazy danych ze związków E/R należy model ten przetłumaczyć na inny model danych. Ze względu na jego uniwersalność i przejrzystość jest często używany w fazie projektowej.

Model E/R składa się z trzech zasadniczych elementów:

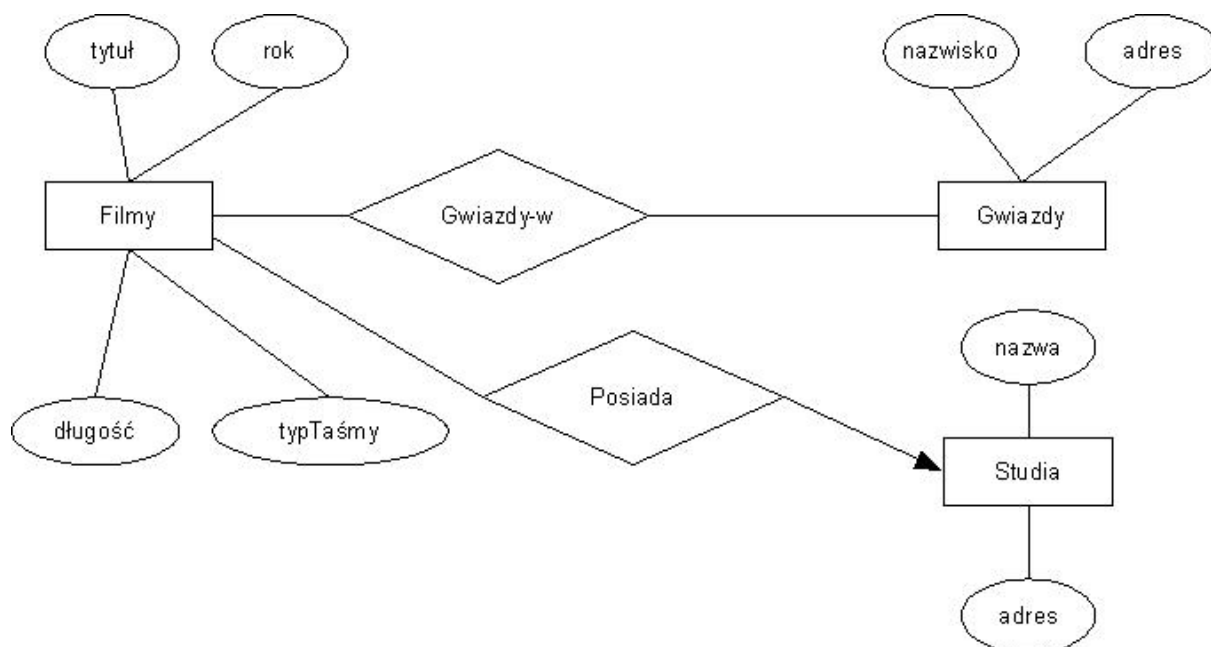
1. zbiory encji (entity sets)
2. atrybutów (attributes)
3. związków (relationships)

Encja jest to obiekt abstrakcyjny, którego zbiór tworzy *zbiór encji*. W odróżnieniu od modelu obiektowego model ten koncentruje się na strukturze (obiekt statyczny), a nie na metodach. Właściwości encji opisują *atrybuty*, które w zależności od wersji modelu E/R mogą być typami atomowymi (np. liczbami) lub prostymi strukturami. Między dwoma lub więcej encjami występują *związki*.

Ze zbiorów encji, atrybutów i związków tworzy się *diagramy związków encji*. Są one grafami w których wierzchołkami są wymienione powyżej typy. W diagramach związków encji stosuje się następujące oznaczenia:

- prostokąty – zbiory encji

- owale – atrybuty
- romby – związki
- strzałki jednostronne – związek wiele do jeden (grot strzałki wskazuje jeden)
- strzałki obustronne – związek jeden do jeden
- linie – związek wiele do wiele



Rys. 1.2. Diagram związków encji dla baz danych filmów [2]

Z diagramu przedstawionego na rysunku 1.2 wynika że związek Posiada jest w relacji wiele do jeden co należy czytać. Jedno studio może posiadać wiele filmów, ale film może być tylko jednego studia. Związek encji studia posiada dwa atrybuty nazwa oraz adres.

1.4.2. Relacyjny model danych

Relacyjny model danych jest obecnie najpopularniejszym modelem baz danych. Model ten koncentruje się na relacjach występujących między zbiorami danych. Relacyjny model danych dostarcza tylko jednego sposobu reprezentacji danych, którym jest dwuwymiarowa tabela nazywana relacją. Charakteryzuje się on względnie prostym (dla ludzi) sposobem przechowywania danych. Przykładowa relacja studenci został pokazana na rysunku 1.3.

nr_indeksu	nr_legitymacji	imie	nazwisko	adres	kod	pesel
847328430	873742347	Alicja	Kowalska	Podchorążych 4/5	30-400	83020101573
847328472	873742365	Andrzej	Jagiello	Miodowa 7	28-122	84062312399
847328411	873742323	Henryk	Popecki	Sławkowska 1/56	78-753	82121000111

Rys. 1.3. Relacja studenci

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Nagłówki relacji nazywamy *atrybutami*. W Relacji studenci atrybutami są:

```
(nr_indeksu, nr_legitymacji, imie, nazwisko, adres, kod, pesel)
```

Każdy atrybut relacji musi mieć określony typ atomowy (podstawowy). Przykładowo `pesel` jest typem liczbowym, `kod` typem znakowym o długości 6 znaków.

Schematem relacji nazywamy jej zbiór atrybutów i zapisujemy go następująco.

```
Relacja(atrybut1, atrybu2, atrybut3, ...)
```

Dla schematu relacji Studenci (przedstawionej na rysunku 1.3) schemat relacji zapisali byśmy następująco:

```
studenci(nr_indeksu, nr_legitymacji, imie, nazwisko, adres, kod,  
pesel)
```

W modelu relacyjnym projekt bazy danych składa się z jednego lub kilku schematów relacji i nazywa się *schematem relacyjnym bazy danych* lub *schematem bazy danych*.

Wszystkie wiersze relacji poza wierszami nagłówkowymi są nazywane *krotkami*. W każdej krotce atrybut ma swój odpowiednik w postaci *składowej* krotki. W pierwszej krotce relacji studenci z rysunku 1.3 jest 7 składowych: 847328430, 873742347, Alicja, Kowalska, Podchorążych 4/5, 30-400, 83020101573, które są kolejnymi wartościami atrybutów: `nr_indeksu`, `nr_legitymacji`, `imie`, `nazwisko`, `adres`, `kod`, `pesel`.

1.4.3. Obiektowy model danych

Obiektowy model danych został stworzony przez grupę ODMG (ang. *Object Data Management Group*) jako uzupełnienie obiektowych języków programowania np. Java, o mechanizm trwałości. Organizacja ODMG stworzyła standard *języka definiowania języków* ODL (ang. *Object Definition Language*). Ostatnia jego wersja (3.0) została udostępniona w roku 2003.

W obiektowych językach programowania programista ma możliwość korzystania z dużej liczby kolekcji typów. Począwszy od typów *atomowych*, takich jak liczby całkowite i rzeczywiste, typ logiczny lub łańcuchy znaków, można tworzyć nowe typy przez stosowanie

konstruktorów typów. Na ogół konstruktory typów umożliwiają tworzenie: struktury rekordów, typów kolekcji, typy referencji (odniesień). W obiektowym modelu danych operujemy na całych obiektach lub klasach, które mogą posiadać funkcje i procedury (zwane *metodami*). Zakłada się że obiekty mają *własny identyfikator* (*object identity* – OID). Żaden obiekt nie może posiadać dwóch lub więcej identyfikatorów oraz żadnego identyfikatora. W odróżnieniu od modelu relacyjnego dwa lub więcej obiektów może mieć dokładnie tą samą zawartość ponieważ są identyfikowane przez różne OID. W obiektowych bazach danych stała się możliwa hermetyzacja (ang. *encapsulate*) ponieważ na obiektach możemy operować metodami. Podobnie jak w obiektowych językach programowania w obiektowym modelu danych możliwa jest *hierarchizacja klas* (*dziedziczenie*).

Składniki obiektowego modelu danych:

- Obiekty
- Klasy obiektów
- Mechanizmy abstrakcji
- Dziedziczenie
- Definiowanie schematu
- Metody
- Komunikaty

Klasa jest złożona z typu i jednej lub wielu funkcji lub procedur, które można wykonać na obiektach danej klasy. Obiekty klasy są wartościami określonego typu (obiekty niezmiennialne) lub zmiennymi, których wartości są tego typu (obiekty zmiennialne) [2]. Na rysunku 1.4 przedstawiona jest przykładowa klasa student.

```
CLASS student = { nr_indeksu: intiger;  
                  pesel: intiger;  
                  nr_legitymacji: intiger;  
                  imie: string;  
                  nazwisko: string;  
                  adres: string;  
                  kod: string;  
                  }
```

Rys. 1.4. Przykładowa klasa student

Metody są to funkcje stowarzyszone z klasami. Metoda ma przynajmniej jeden argument, którym jest obiekt klasy wywołującej. Innymi parametrami funkcji mogą być inne klasy lub typy podstawowe. Przykładowa metoda przedstawiona jest na rysunku 1.5.

ZmienAdres(a: student, b: adres, c: kod)
--

Rys. 1.5. Przykładowa metoda klasy student

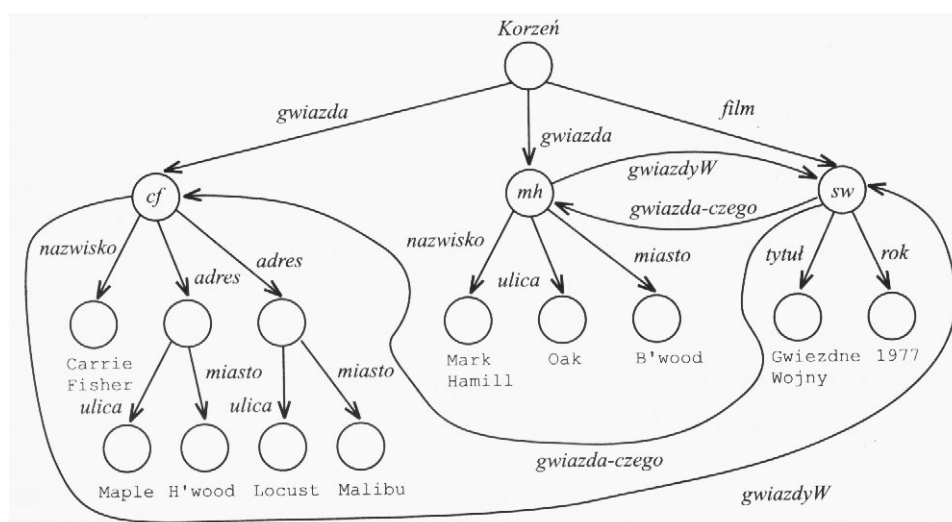
Podobnie jak w językach obiektowych klasy mogą posiadać potomków, którzy dziedziczą wszystkie własności klasy nadrzędnej i jej metody. Klasa podrzędna może posiadać własne typy i metody nieobecne w klasie nadrzędnej.

1.4.4. Semistrukturalny model danych

W systemach baz danych model danych semistrukturalnych odgrywa specjalną rolę [2]:

1. Służy on jako model do integrowania baz danych, to znaczy do opisu danych przechowywanych w dwóch lub większej liczbie baz danych, które zawierają dane podobne, ale w różnych schematach,
2. Służą jako model dokumentów w notacji takiej jak XML.

Baza danych semistrukturalnych jest kolekcją wierzchołków. Każdy wierzchołek jest albo liściem, albo jest wewnętrznym. Z liśćmi są związane dane: ich typ jest atomowy, na przykład są to liczby lub napisy. Z wierzchołków wewnętrznych wychodzi jeden lub większa liczba łuków. Każdy łuk ma etykietę, która określa, jaki związek ma wierzchołek z początku łuku z wierzchołkiem końcowym łuku. Do jednego wierzchołka wewnętrznego, nazywanego korzeniem, nie dochodzi żaden łuk; ten wierzchołek reprezentuje całą bazę danych. Każdy wierzchołek musi być dostępny z korzenia, ale struktura nie musi być drzewem. Na rysunku 1.6 przedstawiono przykładową bazę danych semistrukturalną opisującą filmy i gwiazdy. Na górze znajduje się wierzchołek oznaczony jako Korzeń. Jest to punkt początkowy do przeglądania danych i można go traktować jako reprezentanta całej bazy danych. Główne obiekty lub encje, w tym przypadku filmy i gwiazdy, są reprezentowane przez wierzchołki potomne korzenia [2].



Rys. 1.6. Reprezentacja danych semistrukturalnych filmów i gwiazd [2]

W przedstawionej strukturze jest wiele liści. Etykiety nie są częścią modelu i są umieszczone w wierzchołkach tylko po to, by można się było odwołać do nich, inaczej byłyby nieoznaczone. Można na przykład wierzchołek *sw* traktować jako reprezentujący „Gwiezdne Wojny”: tytuł i rok tego filmu, inne dane nieujawnione i gwiazdy, z których dwie są na rysunku [2].

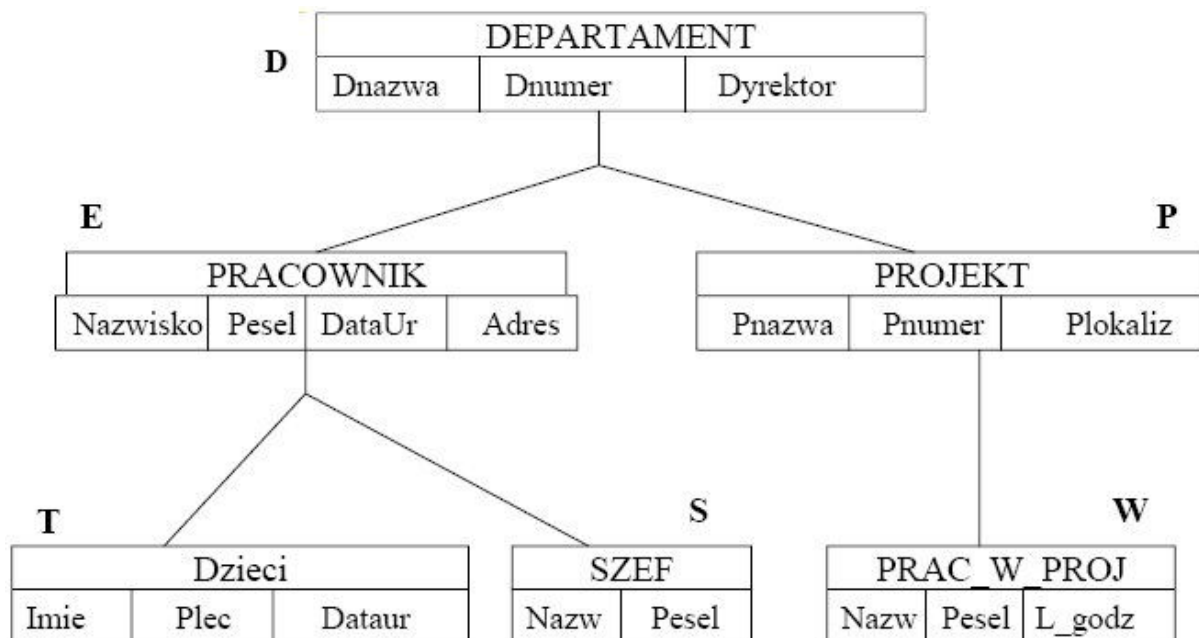
Etykiety łuków odgrywają dwójaką rolę i łączą dane zawarte w definicji klasy i w związkach. Załóżmy, że łuk *L* łączy wierzchołek *M* z wierzchołkiem *N*.

1. Możliwe, że *N* reprezentuje obiekt lub strukturę, a *M* reprezentuje jeden z atrybutów obiektu lub pole struktury. Wówczas odpowiednio *L* reprezentuje nazwę tego atrybutu lub pola.
2. Można traktować *M* i *N* jak obiekty, wówczas *L* jest nazwą związku od *M* do *N*.

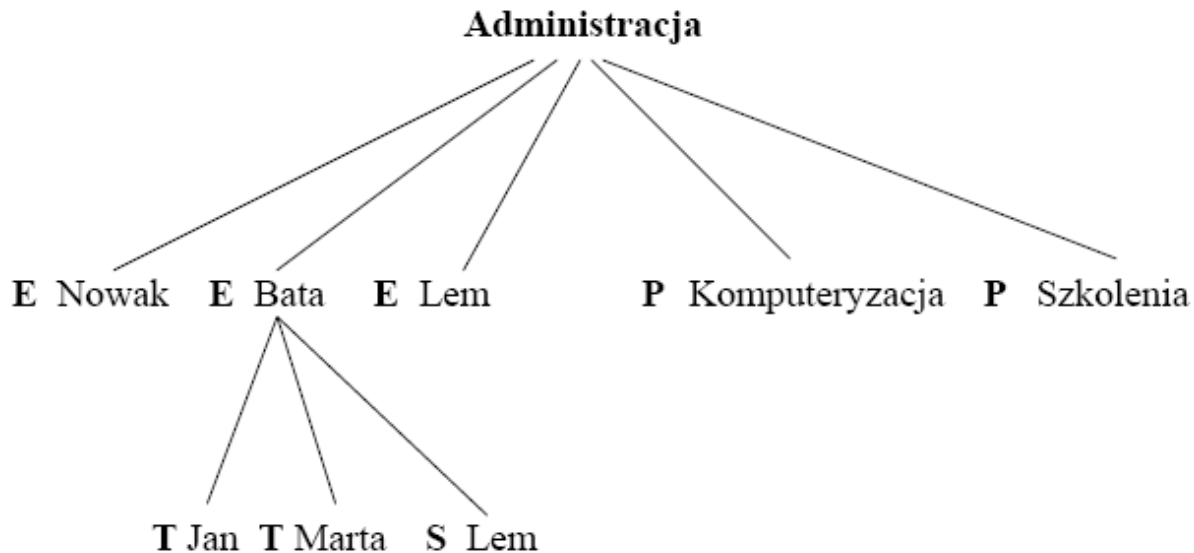
1.4.5. Hierarchiczny model danych

Hierarchiczny model danych został opracowany w wyniku analizy istniejących implementacji programów pełniących rolę baz danych. Używa on dwóch rodzajów struktur danych: typy rekordów i związki nadrzędny-podrzędny. Typ rekordów jest nazwaną strukturą danych, złożoną ze zbioru nazwanych pól np. *pesel* i *nr_legitymacji*. Każde pole jest używane do przechowywania prostego atrybutu i jest mu przyporządkowany typ danych, np. Integer, Character itd. Powiązanie nadrzędny-podrzędny jest związkiem jeden-do-wiele między dwoma typami rekordów. Mówimy, że typ rekordu po stronie jeden związku jest nadrzędnym typem rekordu, takim jak *pracownik*. Rekord po stronie wiele jest podrzędnym typem rekordu, takim jak *dzieci*. A więc, schemat hierarchiczny jest złożony z wielu typów rekordów, powiązanych ze sobą za pomocą związków nadrzędny-podrzędny [1].

Przykładowy schemat hierarchicznego modelu danych dla działu administracji został pokazany na rysunku 1.7. Natomiast na rysunku 1.8 zostały pokazane przykładowe dane umieszczone w schemacie administracji z rysunku 1.7.



Rys. 1.7. Schemat hierarchicznego modelu danych dla działu administracji [22]



Rys. 1.8. Przykładowe drzewo hierarchicznego modelu danych dla schematu bazy zamieszczonego na rysunku 1.7 [22]

Integralność danych w modelu hierarchicznym bazuje na następujących zasadach:

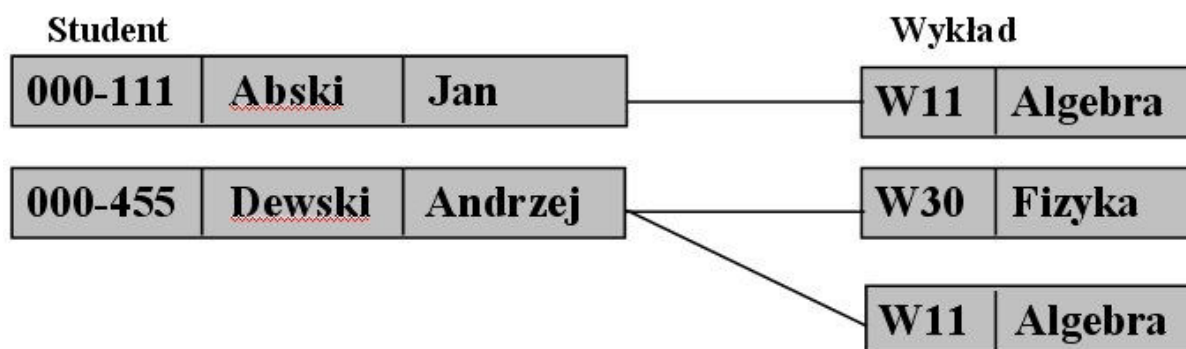
1. Każdy rekord za wyjątkiem rekordu korzenia musi posiadać rekord nadrzędny (tzw. rodzica):

- a) Nie można wstawić rekordu podrzędnego dopóki nie istnieje powiązanie z rekordem nadrzędnym.
 - b) Usunięcie rekordu nadrzędnego powoduje automatyczne usunięcie wszystkich jego rekordów podrzędnych.
2. Jeżeli rekord podrzędny ma związane ze sobą dwa lub więcej rekordów nadrzędnych to rekord podrzędny musi zostać powielony dla każdego rekordu nadrzędnego.

1.4.6. Sieciowy model danych

Sieciowy model danych, podobnie jak hierarchiczny model danych, ma dwie struktury danych: typ rekordów i typ kolekcji. Typ rekordów jest pojęciowo podobny do typu rekordów w modelu hierarchicznym, z tą różnicą, że pola mogą być używane do przechowywania wielu wartości lub do reprezentowania złożonych wartości, które się powtarzają. Na przykład typ rekordów *Studenci* może mieć następujące pola: *pesel*, *nr_legitymacji* i *profil_studenta*. Profil studenta możemy uważać za pole złożone, którego wartością jest powtarzająca się grupa składająca się z wartości *nazwa_kursu*, *rok* i *stopień*. Typ kolekcji jest opisem związku jeden-do-wiele między dwoma typami rekordów.

Konieczne jest rozróżnienie między typami rekordów, a wystąpieniami rekordów i między typami kolekcji, a wystąpieniami kolekcji. Typ rekordów jest częścią definicji schematu. Wystąpienie rekordu jest częścią zawartości bazy danych. To samo dotyczy typów kolekcji i wystąpień kolekcji. Każdy typ kolekcji składa się z nazwy, typu rekordów właściciela (owner) i typu rekordów członka (member) kolekcji. Wystąpienie kolekcji jest grupą złożoną z jednego rekordu właściciela i jednego lub więcej wystąpień członka [1].



Rys. 1.9. Przykładowy wycinek sieciowej bazy danych [24]

Integralność danych w sieciowym modelu danych dotyczy głównie określenie członkostwa w kolekcji i typu wstawiania. Członkostwo w kolekcji może być zdefiniowane

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

jako wymagane lub opcjonalne. Jeśli status jest ustawiony na wymagane, to system spowoduje, że każde wystąpienie rekordu członka będzie należało do pewnego wystąpienia kolekcji. Dla członków kolekcji jest określany tryb wstawiania. Jeśli tryb wstawiania jest ustawiony na ręczny, to programy użytkowe muszą wstawiać rekordy członków do określonego wcześniej wystąpienia kolekcji. Jeśli tryb wstawiania jest automatyczny, to gdy jest tworzony rekord członka, jest on automatycznie wstawiany do bieżącego wystąpienia kolekcji. Opisane więzy definiujemy za pomocą dodatkowej klauzuli w definicji typu kolekcji.

2. Integracja baz danych

Zanim przejdziemy do rozważań na temat integracji baz danych należy zastanowić się na integralnością danych na poziomie bazy danych. Prawidłowa implementacja reguł integralności danych potrafi zdecydowanie lepiej pilnować spójności danych niż jakakolwiek aplikacja łącząca się z tą bazą. Jest to szczególnie ważne przy integracji bazy danych z aplikacjami jak i przy synchronizacji danych pomiędzy bazami danych.

2.1. Zapewnienie integralności danych

Słowo integralność pochodzi od angielskiego słowa integrity. Oznacza ono, że można zaufać temu co dana osoba mówi i robi. Analogicznie jest w przypadku bazy danych. Baza danych posiadająca właściwość integralności musi posiadać bieżące informacje o przechowywanej rzeczywistości (modelu, relacjach, itp.). Przykładowo jeśli chcemy zachować integralność to musimy zadbać aby w przypadku przeniesienia przedmiotu z jednego magazynu do drugiego w obu nastąpiła zmiana. W wydającym magazynie należy usunąć wpis o przedmiocie w tabeli stanu, a w przyjmującym magazynie należy utworzyć wpis w tabeli stanu. [3]

Nie zależnie od używanego modelu danych każdy model ma swoje własne zasady integralności. W dalszej części pracy skupię się głównie na bazach relacyjnych i obiektowo-relacyjnych. Mój wybór motywuję tym iż udział w rynku baz relacyjnych jest dominujący. Co skutkuje dużą zaszłością baz i w konsekwencji coraz większą potrzebą integracji posiadanych przez firmy baz danych.

2.1.1. Integralność danych

Integralność danych w relacyjnej bazie danych jest kontrolowana za pomocą więzów, które można podzielić na dwie grupy [9]:

- Więzy deklaratywne – statyczne, stosunkowo łatwe do tworzenia i zarządzania. Do grupy tej należą: ograniczenia (ang. *constraints*), reguły (ang. *rules*) i wartości domyślne (ang. *default values*).
- Więzy proceduralne – dynamiczne, pozwalające na oprogramowanie skomplikowanych zależności i reguł biznesowych. Do tej kategorii należą: procedury przechowywane, wyzwalacze, kod innych warstw aplikacji (poza warstwą bazy danych).

Jak już wspominałem w rozdziale 1.4.2 model relacyjny posiada dwie reguły integralności: integralność encji i integralność referencyjną. Należą one do grupy więzów deklaratywnych.

Zasada integralności encji mówi, że każda tabela musi posiadać klucz główny. Kolumna lub kolumny wybrane na klucz główny muszą być jednoznaczne (wartość lub zbiór wartości muszą być unikatowe) i nie mogą zawierać wartości `null`. Konsekwencją reguły integralności encji jest fakt, że w tabeli są zabronione powtórzenia wierszy. Jeżeli każdy klucz główny jest unikatowy to takie same wiersze są po prostu niemożliwe. Integralność encji jest definiowana poprzez dodanie do definicji schematu klauzuli klucza głównego. Niektóre komercyjne systemy baz danych zezwalają na tworzenie relacji bez klucza głównego. Na rysunku 1.3 zamieściłem schemat przykładowej relacji studenci. Poniżej przedstawiam definicję tej samej relacji z kluczem głównym założonym na atrybucie `nr_indeksu`.

```
CREATE TABLE studenci (  
    nr_indeksu INT NOT NULL ,  
    nr_legitymacji INT NOT NULL ,  
    imie VARCHAR( 15 ) NOT NULL ,  
    nazwisko VARCHAR( 25 ) NOT NULL ,  
    adres VARCHAR( 40 ) NOT NULL ,  
    kod CHAR( 6 ) NOT NULL ,  
    pesel INT NOT NULL ,  
    PRIMARY KEY ( nr_indeksu )  
);
```

Rys. 2.1. Definicja relacji studenci w języku SQL z kluczem głównym utworzonym na atrybucie `Nr_indeksu`

Oprócz deklaracji klucza głównego można określić jeszcze klucz `UNIQUE`. Za wyjątkiem dwóch różnic ma on takie same ograniczenia jak `PRIMARY KEY`:

1. Dla jednej tabeli może być wiele deklaracji `UNIQUE`, ale może być tylko jedna deklaracja klucza głównego.
2. W przeciwieństwie do „`PRIMARY KEY`” klucz „`UNIQUE`” dopuszcza wartości `NULL`. `UNIQUE` dopuszcza także wystąpienie więcej niż jednej wartości `NULL` co jest odstępstwem od zasady nie powtarzających się krotek w kluczu.

W naszej relacji studenci (rys. 2.1) wartości unikatowe powinny posiadać atrybuty `nr_indeksu`, `nr_legitymacji`, `pesel`. Ponieważ mogliśmy użyć tylko jednego klucza głównego i chcieliśmy by był on wartością unikatową to poprawiając schemat relacji

studenci rozszerzymy ją o nowe więzy integralności nadając atrybutom `nrLegitymacji` oraz `pesel` wartości unikatowe (rys. 2.2).

```
CREATE UNIQUE INDEX idx_nr_legitymacji ON studenci(nr_legitymacji) ;  
CREATE UNIQUE INDEX idx_pesel ON studenci(pesel) ;
```

Rys. 2.2. Instrukcje SQL tworzące więzy UNIQUE w relacji studenci

Gdybyśmy teraz chcieli wstawić do tabeli dane studenta, którego numer `pesel` byłby taki sam jak jednego ze studentów już wpisanego do tabeli wtedy system zarządzania bazą danych anuluje operację wstawiania informując nas o istnieniu krotki z takim numerem `pesel`. W rzeczywistości opisywanej przez tą relację nigdy nie powinna zajść taka sytuacja. Ale w przypadku gdy zajdzie oznacza to, że albo student źle podał swój `pesel`, albo osoba wpisująca dane do bazy pomyliła się wpisując dane tego lub istniejącego już w bazie studenta.

Integralność referencyjna mówi, że każda wartość klucza obcego może znajdować się tylko w jednym z dwóch stanów. W pierwszym stanie wartość klucza obcego odwołuje się do wartości klucza głównego istniejącej tabeli w bazie danych. Jest to stwierdzenie faktu istnienia związku między krotkami powiązanymi kluczem obcym. W drugim stanie wartość klucza obcego jest równa `null`. Co oznacza nieznaną związek pomiędzy krotkami lub brak takiego związku.

Rozważmy związek pomiędzy studentami, a kursami. Ponieważ każdy student może uczestniczyć w wielu kursach, a kurs może mieć wielu uczestników więc w naszej relacji nazwijmy ją `zapisy` istnieje związek wiele-do-wielu. W relacji `zapisy` nie może wystąpić sytuacja, w której student usunięty z listy widnieje nadal na liście uczestników kursu. Drugą zabronioną sytuacją jest istnienie zapisów na kurs, który nie istnieje. W celu uniknięcia opisanych sytuacji można definiować następujące związki kluczy obcych (rys. 2.3).

```
CREATE TABLE kursy (  
    symbol character varying(5) NOT NULL,  
    nazwa character varying(150) NOT NULL,  
    PRIMARY KEY ( symbol )  
);  
  
CREATE TABLE zapisy (  
    symbol character varying(150) NOT NULL,  
    id_studenta integer NOT NULL,  
    CONSTRAINT id_studenta FOREIGN KEY (id_studenta) REFERENCES studenci(nr_indeksu),  
    CONSTRAINT symbol FOREIGN KEY (symbol) REFERENCES kursy(symbol)  
);
```

Rys. 2.3 . Instrukcje SQL definiujące relację kursy oraz związek zapisy pomiędzy studentami i kursami

Od tej pory każda operacja naruszająca integralność referencyjną relacji zapisy zostanie odrzucona przez systemu zarządzania bazą danych. Zatem wykonanie następujących akcji nie powiedzie się:

1. Próbuje się wstawić nową krotkę do relacji zapisy, która w polu symbol i id_studenta ma wartość różną od NULL i która nie występuje w relacji kursy lub studenci. Próba wstawienia nie powiedzie się i krotka nie zostanie wstawiona.
2. Próbuje się zmienić krotkę relacji zapisy, tak aby wartość w polu symbol i id_studenta była różna od NULL. Wstawiana wartość nie występuje w polu symbol relacji kursy lub id_studenta relacji studenci żadnej istniejącej krotki. Modyfikacja zostanie odrzucona i krotka pozostanie taka jak była.
3. Próbuje się usunąć krotkę z relacji kursy, ale jej składowa w polu symbol występuje również w polu symbol jednej lub więcej krotek relacji zapisy. Ta próba również nie powiedzie się i krotka pozostanie w relacji kursy.
4. Próbuje się zmienić wartość atrybutu symbol w relacji kursy, ale stara wartość symbol występuje w polu symbol relacji zapisy. Modyfikacja zostanie odrzucona i krotka pozostanie bez zmian.

Czasami takie zachowanie jest nie pożądane. Na szczęście istnieje jeszcze inny sposób wprowadzenia zmian (typu 3 i 4 wymienionego powyżej) powiązanych ze sobą relacji. Jest nim *kaskadowa procedura*. W przypadku tej procedury zmiany atrybutów referencyjnych są powielane na kluczu obcym. W przypadku usunięcia krotki występującej w relacji kursy i relacji zapisy usunięcie kursu powoduje usunięcie powiązanej krotki z relacji zapisy. Analogiczna sytuacja jest w przypadku zmiany wartości symbolu. Jeśli zmienimy symbol kursu występującego w relacji zapisy to zmianie zostaną automatycznie poddane krotki powiązane w relacji zapisy.

Drugim sposobem zmiany domyślnego zachowania DBMS jest stworzenie krotki wiszącej. Czyli takiej która nie ma wartości (ma wartość NULL) klucza obcego z macierzystej relacji (krotki). W tej sytuacji usunięcie krotki wykładowcy powiązanej z krotką kursy ustawi w atrybucie wykładowca tej krotki wartość NULL. Takie zachowanie jest zasadne ponieważ może się zdarzyć, że przez pewien czas kurs nie będzie miał wykładowcy.

Można wnioskować, że integralność referencyjna w trybach RESTRICT, CASCADE i SET NULL jest rozwiązaniem wielu problemów projektantów aplikacji bazodanowych

jednak dopiero w połączeniu z możliwością przypisywania różnych trybów dla operacji usuwania (DELETE) i modyfikowania (UPDATE) wydaje się kompletna.

W naszej bazie danych daje nam ona następujące możliwości. Stwórzmy relację wykładowcy oraz zmodyfikujmy relację kursy dodając klucz obcy wykładowca z relacji wykładowcy (rys. 2.4). Ustawiamy sprawdzanie klucza obcego w relacji kursy na tryb kaskadowy podczas operacji aktualizacji danych, a tryb wstaw-null podczas usuwania danych.

```
CREATE TABLE wykładowcy (  
    id INT NOT NULL,  
    imie VARCHAR( 15 ) NOT NULL ,  
    nazwisko VARCHAR( 25 ) NOT NULL ,  
    adres VARCHAR( 40 ) NOT NULL ,  
    kod CHAR( 6 ) NOT NULL ,  
    pesel INT NOT NULL ,  
    PRIMARY KEY ( id )  
);  
CREATE UNIQUE INDEX idx_wykladowcy_pesel ON wykładowcy(pesel);  
ALTER TABLE kursy  
    ADD COLUMN wykładowca integer,  
    ADD CONSTRAINT wykładowca FOREIGN KEY (wykładowca) REFERENCES  
        wykładowcy(id) ON UPDATE ON CASCADE DELETE SET NULL;
```

Rys. 2.4. Instrukcje SQL definiujące relację wykładowcy i rozszerzające relację kursy o klucz obcy z relacji wykładowcy

Uzyskaliśmy w tej chwili bardzo naturalną sytuację:

1. W przypadku odejścia wykładowcy jego kursy nie są usuwane (jak również informacja o zapisanych studentach na te kursy) tylko do czasu przypisania nowego wykładowcy kurs pozostaje bez wykładowcy (ma ustawioną wartość NULL).
2. W przypadku zmiany identyfikatora wykładowcy, wszystkie powiązane z nim wartości kluczy obcych są automatycznie zmieniane na nową wartość.

Jeśli więzy integralności wymagają złożonych kontroli danych to może wystąpić sytuacja, w której porządkowana przez programistę kolejności aktualizacji stanie się niewystarczające i żadna zmiana danych nie będzie możliwa. Taką sytuację nazywamy *więzami zapętlonymi*. Przykładowo jeśli mamy relację A i relację B. Relacja A ma klucz obcy z relacji B w trybie RESTRICT. Relacja B ma klucz obcy także w trybie RESTRICT z relacji A. To dodanie nowej krotki do relacji A i B będzie nie możliwe. Na szczęście nie jest to sytuacja bez wyjścia. Istnieje możliwość odroczenia sprawdzania więzów integralności dla *transakcji*. Szczegółowo transakcjami zajmiemy się w rozdziale 2.1.2. W skrócie transakcja

jest to grupa instrukcji traktowana jako całość. Każde więzy można zadeklarować jako możliwe do odroczenia (`DEFERRABLE`) i niemożliwe do odroczenia (`NOT DEFERRABLE`). Druga możliwość jest deklarowana domyślnie jeśli nie określono sposobów obsługi odraczania. Jeżeli więzy zostały zadeklarowane jako możliwe do odroczenia to istnieje możliwość określenia by więzy zostały odroczone do czasu zakończenia całej transakcji. Istnieją dwa rodzaje więzów odraczanych. Odraczane na początku (`INITIALLY DEFERRED`) i na początku natychmiastowe (`INITIALLY IMMEDIATE`). W pierwszym przypadku sprawdzanie zostanie odroczone do końca bieżącej transakcji lub do chwili gdy system otrzyma jawne polecenie zatrzymania odraczania więzów (`SET CONSTRAINT ... IMMEDIATE`). W drugim przypadku komendy zostaną wykonane natychmiast, chyba że odroczymy sprawdzanie więzów (`SET CONSTRAINT ... DEFERRED`). Przy deklaracji natychmiastowego sprawdzanie będzie wykonywane przed modyfikacją krotki.

Do tej pory zajmowaliśmy się więzami dotyczące kluczy. Teraz zajmiemy się więzami działającymi na wartościach atrybutów. Takie więzy można określić [2]:

1. w definicji schematu relacji jako warunek nałożony na atrybut,
2. więzy nałożone na krotki jako na całość (takie więzy są częścią schematu bazy danych i nie są związane z atrybutami).

W przedstawionym przykładzie na rysunku 2.4 można odnaleźć więzy określone w punkcie 1. Więzy określające wystąpienie wartości `NULL` lub ich nie wystąpienie `NOT NULL` są więzami określonymi na atrybucie. Dają nam one możliwość kontroli wymaganej wartości w krotce lub też jej braku. Więzy `NOT NULL` są więzami stosowanymi domyślnie. Przykładowo jeśli uważamy atrybut `imie` za konieczny możemy w deklaracji relacji użyć następującego fragmentu instrukcji.

```
imie VARCHAR( 15 ) NOT NULL ,
```

W celu określania bardziej skomplikowanych ograniczeń dla wartości atrybutów należy zastosować więzy typu `CHECK`. Więzy te są sprawdzane podczas określania nowej wartości ograniczonego atrybutu. Może on być nadany podczas modyfikacji krotki lub w wyniku dołączania nowej krotki. Jeśli nowa wartość nie spełnia narzuconych warunków to operacja zostanie odrzucona przez DBMS. W naszej bazie danych moglibyśmy użyć więzów `CHECK` do sprawdzania numeru `pesel`. Ponieważ wiemy że `pesel` musi się składać z 11 cyfr. Dlatego sprawdzając przedział `pesel < 100 000 000 000, 100 000 000 000`) odrzucimy cyfry składające się z innej liczby cyfr niż 11.

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

```
ALTER TABLE wykadowcy  
ADD CHECK (pesel >= 10000000000 AND pesel < 100000000000)
```

Najsilniejszą formą sprawdzania więzów są *asercje*. Asercja (ang. *assertions*) to ograniczenia występujące w schemacie jako niezależne od tabel obiekty. Asercje służą do kontroli wartości wprowadzanych do tabel [25]. Są one wyrażeniami logicznymi, które zawsze muszą mieć wartość *True*. Największą ich przewagą nad więzami typu CHECK jest sprawdzanie więzów przy modyfikacji danych. W naszej bazie danych mogli byśmy użyć asercji do ograniczenia ilości prowadzonych wykładów do 5 przez jednego wykładowcę.

```
CREATE ASSERTION LimitWykladow CHECK(5) <= ALL (SELECT COUNT(*) FROM  
kursy GROUP BY wykadowca));
```

Niestety asercje są trudne do implementacji, dlatego nie każda baza danych je posiada. Zamiast tego większość baz danych zawiera wyzwalacze.

Wyzwalacze, nazywane także procedurami wyzwalanymi (ang. *triggers*) są wygodnym sposobem realizowania zaawansowanych więzów integralności na poziomie bazy danych, bez konieczności implementacji dodatkowego kodu w innych warstwach aplikacji. Ich niewątpliwą zaletą jest skupianie reguł integralności w jednym miejscu, niezależnie od tego, skąd wywoływana jest modyfikacja danych. Wyzwalacze posiadają jednak także wady, do których należy potencjalny spadek wydajności przetwarzania bazy przy częstych operacjach dodawania, edytowania lub usuwania danych. Są one specjalnym rodzajem procedury przechowywanej na serwerze, która jest przypisana do określonej tabeli i automatycznie uruchamiana podczas modyfikacji danych w tej tabeli. Wyzwalacz jest rzeczywiście wyzwalany po uruchomieniu określonego polecenia (INSERT, UPDATE lub DELETE) modyfikacji danych, dla którego jest zdefiniowany. Za pomocą wyzwalaczy można wykonywać następujące czynności [9]:

- porównywać dane przed i po wykonaniu danej operacji,
- wykrywać i wycofywać błędne modyfikacje danych,
- automatycznie odczytywać i modyfikować dane z innych tabel, należących do tej samej lub innej bazy,
- wywoływać procedury przechowywane (w bazie lokalnej lub zdalnej).

Zastosowanie wyzwalaczy:

- Zarządzanie danymi powielonymi albo obliczonymi na podstawie innych danych.
- Kaskadowa integralność referencyjna – jeśli modyfikujemy albo kasujemy wiersze z jednej tabeli, odpowiednio są zmieniane lub usuwane kaskadowo wiersze z tabeli

powiązanej. W systemach bazodanowych, w których mechanizm ten nie jest realizowany deklaratywnie, wyzwalacze stanowią jeden ze sposobów rozwiązania tego problemu.

- Skomplikowane wartości domyślne pól w tabelach – np. domyślna wartość pola [NumerFaktury] to ciąg znakowy, złożony z kolejnego numeru faktury w danym miesiącu, numeru miesiąca i roku wystawienia.
- Integralność referencyjna pomiędzy różnymi bazami danych – zmiany w jednej bazie danych automatycznie powodują zmiany w innej bazie (lokalnej lub zdalnej).

Olbrzymią zaletą wyzwalaczy jest to, iż są one uruchamiane przy każdej modyfikacji danych w określonej tabeli (z wyłączeniem przypadków szczególnych), niezależnie od tego, z którego miejsca aplikacji zmiana ta jest wywoływana. Od strony programistycznej znacznie ułatwia to zarządzanie regułami logiki biznesowej na poziomie bazy danych, ponieważ kod zapewniający integralność jest skupiony w jednym miejscu, a nie rozproszony np. jest przechowywanych w wielu procedurach programu [9].

2.1.2. Transakcje

Transakcja jest to sekwencja instrukcji dla systemu zarządzania bazą danych, po wykonaniu której, baza danych powinna przejść z jednego stanu spójnego w inny stan spójny. Jeśli tak się nie stanie należy odwołać transakcję. Musi być wykonana niepodzielnie i w pozornej izolacji od innych transakcji. Do sterowania transakcjami służy język TCL (ang. Transaction Control Language). Transakcja składa się z trzech etapów:

1. rozpoczęcia,
2. wykonania,
3. zamknięcia.

Przetwarzanie transakcyjne charakteryzują cechy o akronimie: ACID (ang. *Atomicity, Consistency, Isolation, Durability*) - Niepodzielność (atomowość), Spójność, Izolacja (odseparowanie, wyłączość), Trwałość (wytrzymałość). Integralność bazy danych bez zachowania tych cech była by niemożliwa. Oznaczają one [4]:

- Niepodzielność (ang. *Atomicity*) - wszystkie operacje (podtransakcje) wchodzące w skład transakcji muszą być wykonane wspólnie lub zostać wycofane. Działa tu zasada: "wszystko albo nic". Przykładowo podczas wypłaty z bankomatu nie można zapisać informacji tylko o wypłacie nie zapisując równocześnie zmiany na koncie klienta banku.

- Spójność (ang. *Consistency*) - wymaganie to zapewnia skoordynowane przejście bazy danych z jednego stanu do drugiego. W przypadku gdy wystąpi stan, którego nie da się przyjąć w modelowanej rzeczywistości, mówimy że baza danych nie jest w stanie spójnym. Nie może wystąpić taka sytuacja, że istnieje odwołanie do obiektu który nie istnieje. Przykładowo nie może być przydzielony jeden bilet dwóm pasażerom. Taka sytuacja ma prawo wystąpić tylko przez chwilę w trakcie przetwarzania transakcji, gdy przydziela się miejsca dla pasażerów. Ale po zakończeniu transakcji baza danych musi spełniać wszystkie warunki niesprzeczności. Moduł zarządzania transakcjami zwany monitorem transakcji musi taką sytuację zagwarantować[2].
- Izolacja (ang. *Isolation*) - transakcja musi przebiegać w izolacji od innych transakcji. Dla dowolnych dwóch transakcji tylko jedna z nich może zaakceptować poprawne zakończenie drugiej [4]. Przykładem mogą tu być dwa biura sprzedaży równocześnie sprzedające ten sam produkt, podczas gdy została tylko jedna sztuka. Tylko jedno zapytanie może być obsłużone (wykonane poprawnie).
- Trwałość (ang. *Durability*)- w przypadku poprawnego zakończenia transakcji, zmiany wprowadzone są na stałe i transakcja nie może być już wycofana. W przypadku awarii systemu zaraz po zakończeniu transakcji wszystkie zmiany jakie wykonała transakcja zostaną odzyskane.

Uwzględniając obsługę więcej niż jednej transakcji w tej samej chwili napotykamy problem jednoczesnego korzystania z danych. Brak zabezpieczeń przed dostępem do danych ze strony innych programów do czasu wykonania operacji może spowodować, że wynik operacji będzie nieadekwatny do stanu bazy danych w momencie zakończenia obliczeń [5]. Rozważmy przykładowy system bazy danych obsługujący zbiór terminali linii lotniczej, które są przechowywane w plikach. Jeden lot jest reprezentowany przez jeden plik. Załóżmy że użytkownik A otrzymuje dostęp do rekordu lotu 709 w czasie t_1 , a użytkownik B otrzymuje dostęp do rekordu lotu 709 w czasie t_2 w przybliżeniu równym t_1 . Użytkownik A zwiększa ilość zarezerwowanych miejsc o 2 w locie 709 w czasie t_3 . Użytkownik B zwiększa o 10 ilość zarezerwowanych miejsc w locie 709 w czasie t_4 . Problemem jest utrata ilości zarezerwowanych miejsc przez użytkownika A ponieważ użytkownik B zapisał dane po użytkowniku A. Zmiany w czasie ilości wolnych miejsc pokazuje tabela 2.1.

Tabela 2.1 Zmiana ilość wolnych miejsc w czasie trwania transakcji w przykładowej linii lotniczej

	t1	t2	t3	t4	T5
Ilość wolnych miejsc na lot 709	30	30	27	20	20
Ilość wolnych miejsc widzianych przez użytkownika A	30	30	27	27	20
Ilość wolnych miejsc widzianych przez użytkownika B	30	30	30	20	20

Przykład ten ukazuje problem kontroli współbieżności operacji. Rozwiązaniem tego problemu może być zastosowanie wykonywania operacji szeregowo (sekwencyjnie). Jednak ze względu na ilość operacji (jest ich zbyt dużo) część z nich musi być przetwarzana równoległe w celu zwiększenia szybkości bazy danych. Jednym ze sposobów zapewniającym poprawny wynik przetwarzania jest blokowanie elementów bazy danych i stosowanie brudnopisu. Stosuje się je w celu uniknięcia jednoczesnego dostępu do tych samych danych. W komercyjnych systemach baz danych są różne implementacje blokad i brudnopisów, dla nas istotny jest ogólny mechanizm i jego konsekwencje.

Blokady umożliwiają zablokowanie określonego obszaru bazy danych dla danej transakcji. Umożliwia to zachowanie pozornej izolacji transakcji. Istnieją dwa rodzaje transakcji: do odczytu i zapisu (ang. *READ WRITE*) oraz tylko do odczytu (ang. *READ ONLY*). Transakcje tylko do odczytu oznaczają, że dane nie będą modyfikowane co pozwala przeprowadzać odczyt równoległy tych samych danych wielu transakcjom jednocześnie bez zagrożenia kolizji.

SET TRANSACTION READ ONLY

Transakcje do odczytu i zapisu są domyślnie stosowane i informują on bazę danych, że transakcja może odczytywać i zapisywać dane. Oznacza to że baza danych wstrzyma ewentualne równoległe transakcje zapisujące dane w tym samym miejscu do czasu zakończenia transakcji, która założyła blokadę.

SET TRANSACTION READ WRITE

Brudnopis jest to mechanizm mający na celu przyspieszenie operacji modyfikacji w bazie danych. Można sobie go wyobrazić jako zwykły brudnopis (w potocznym znaczeniu). Wszystkie nie zatwierdzone transakcje zapisują dane w brudnopisie. Po zatwierdzeniu transakcji dane są zapisywane bazie danych. W przypadku odwołania transakcji dane są usuwane z brudnopisu. Negatywnym skutkiem używania brudnopisu jest zmniejszenie wydajności bazy danych. Dlatego w celu zwiększenia szybkości umożliwia się domyślne czytanie danych z brudnopisu dla innych transakcji. Przy projektowaniu transakcji należy przewidzieć ten efekt ponieważ czasami odczyt danych z brudnopisu może być niepożądany.

Przykładowy niepożądany odczyt z brudnopisu występuje przy operacjach bankowych. Załóżmy że mamy trzy konta K1, K2, K3 z saldami odpowiednio: 10zł, 20zł, 20zł.

Algorytm transakcji przelewu jest następujący:

- Dodać pieniądze do konta na które jest przelew (docelowego)
- Sprawdzić czy na koncie dokonującym przelew (źródłowym) jest wystarczająca ilość środków.
 - Jeśli nie to usuwa dodanie pieniędzy do konta docelowego i kończy transakcję
 - Jeśli jest wystarczająca kwota to odejmij kwotę z konta źródłowego i zakończ transakcję.

Założmy, że transakcja T1 ma powodować przelew 15zł z konta K1 na K2. W tym samym czasie transakcja T2 ma dokonać przelewu 25zł z konta K2 na K3.

Poniżej przedstawiam potencjalną możliwość zdarzeń:

1. T2 dodaje 25zł na konto K3, gdzie obecne saldo wynosi 55zł.
2. T1 dodaje 15zł do konta K2, gdzie obecne saldo wynosi 35zł.
3. T2 wykonuje sprawdzenie salda konta źródłowego, na koncie K2 jest dostateczna suma pieniędzy (35zł), co umożliwia przelew 25zł z K2 na K3.
4. T1 wykonuje sprawdzenie salda konta źródłowego, na koncie K1 nie ma wystarczających środków (10zł), aby dokonać przelewu z K1 do K2.
5. T2 odejmuje 25zł od konta K2, gdzie teraz pozostaje 10zł, a transakcja T2 zostaje zatwierdzona.
6. T1 odejmuje kwotę 15zł z konta K2 i zostaje tam teraz -5zł. Transakcja T1 zostaje cofnięta.

Ogólna kwota na wszystkich kontach się nie zmieniła. Jednak w 3 punkcie transakcja T2 wykonała odczyt z brudnopisu co spowodowało błędny odczyt salda i zapisanie wartości ujemnej na koncie.

Kontrolę nad zachowaniem transakcji przy współpracy z brudnopisem określa poziom izolacji. Istnieją takie cztery poziomy:

1. Szeregowalny – jest to omówiony powyżej tryb do odczytu i zapis *READ WRITE*
`SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`
2. Odczyt niezatwierdzony – umożliwia czytanie z brudnopisu nie zatwierdzonych transakcji. Przykład z przelewami obrazuje jego zachowanie.
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`
3. Odczyt zatwierdzony – zabrania on odczytu danych z brudnopisu. Czyli jeśli transakcja T1 odczytuje relację X przy poziomie izolacji odczytu zatwierdzonego, a

transakcja T2 dokonuje zmian krotek relacji X ale a następnie cofa transakcję to transakcja T1 nie zobaczy żadnych zmian wykonanych przez transakcję T2.

SET TRANSACTION ISOLATION LEVEL READ COMMITED

4. Odczyt powtarzalny – nazwa trochę myląca. Ponieważ zapytanie nie gwarantuje otrzymania tej samej odpowiedzi. Jeśli krotka wystąpiła przy pierwszym wykonaniu zapytania to w kolejnych odpowiedziach też wystąpi, ale mogą się także pojawić dodatkowe krotki tzw. *fantomy*. Są to takie krotki, które powstały w wyniku wstawiania wykonanego przez inne transakcje.

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

2.2. Integracja bazy danych z programami

Większość baz danych jest używana poprzez aplikacje klienckie lub aplikacje przechowujące dane w bazie danych. Przy łączeniu środowiska programistycznego aplikacji z bazą danych występuje problem przekazywania danych. Może się one nieznacznie różnić w zależności od implementacji bazy danych i używanego języka zapytań. W naszych rozważaniach skupię się na bazach relacyjnych, które używają języka SQL w aplikacjach jako najbardziej rozpowszechnionych.

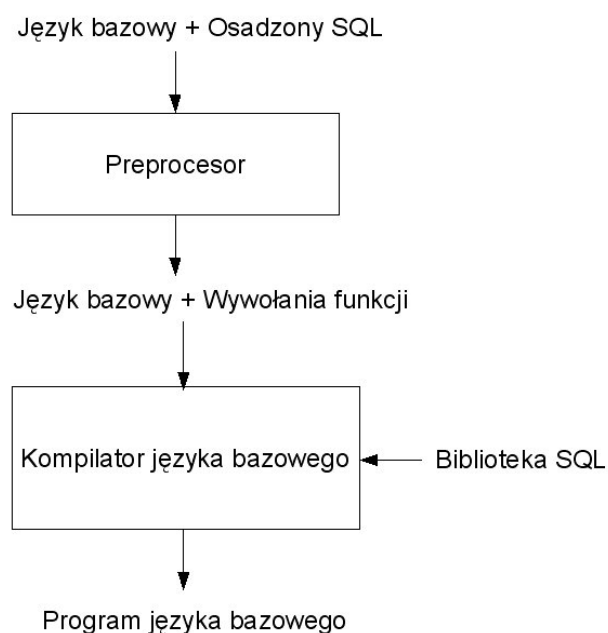
Prawie każdy producent bazy danych udostępnia interpretery umożliwiające wprowadzanie oraz wykonanie zapytań i poleceń. Są one jednak mało wygodne i zbyt trudne dla zwykłych użytkowników.

W dotychczasowych przykładach używałem generycznego interfejsu języka SQL. W praktyce instrukcje SQL stanowią niewielkie fragmenty większego oprogramowania. Połączenie instrukcji SQL z konwencjonalnym językiem programowania stwarza problem *niezgodności impedancji*. Oznacza to, że model danych SQL różni się bardzo od modelu innych języków. SQL jako jądro stosuje relacyjny model danych natomiast języki konwencjonalne operują modelem typów strukturalnych z podstawowymi typami danych. W wyniku tego nie istnieje bezpośrednie przełożenie między strukturami wsypującymi w języku SQL, a innymi językami. Z tego powodu trzeba tworzyć dodatkowe mechanizmy umożliwiające przepływ danych między fragmentami programu zapisanymi w SQL, a innymi językami. Takimi mechanizmami są [26]:

1. Statyczny SQL
2. Dynamiczny SQL
3. Interfejs Programowania Aplikacji API

2.2.1. Osadzony SQL

Osadzony SQL (ang. *Embedded SQL*) jest standardem dla łączenia SQL z siedmioma językami bazowymi. Programista tworzy program w języku bazowym (nazywany też zewnętrznym), ale z osadzonymi instrukcjami SQL, które nie należą do języka bazowego. Gotowy program jest analizowany przez preprocesor, który powoduje zamianę kodu SQL na kod sensowny dla kompilatora języka bazowego. Interpretacja SQL w najprostszym wypadku sprowadza się do wywołania funkcji, której argumentem jest tekst instrukcji SQL i która uruchamia tę instrukcję. Po fazie preprocesowania program jest kompilowany w zwykły sposób. Na rysunku 2.5 pokazany został algorytm przetwarzania programów z osadzonym językiem SQL.



Rys. 2.5. Schemat przetwarzania programów z osadzonymi instrukcjami SQL

2.2.2. Dynamiczny SQL

Główną różnicą pomiędzy osadzonym, a dynamicznym SQL jest sposób kompilacji zapytań. Mianowicie w dynamicznym SQL instrukcje dla bazy danych nie są znane w całości w fazie kompilacji. Program pobiera je dopiero w trakcie działania. Natomiast same zapytania do bazy są kompilowane na bieżąco w trakcie działania programu. Powoduje to niewielki spadek prędkości aplikacji (nie zauważalny), ale umożliwia to zapytania ad hoc.

Dostawcy systemów baz danych dostarczają biblioteki z definicjami niezbędnych funkcji dla języków bazowych. Umożliwia to wywoływanie funkcji z zapytaniem SQL bez

zagłębiania się w implementację. To podejście nazywa się *interfejsem poziomu wywołań* lub CLI (ang. *Call Level Interface*) [2].

2.2.3. Interfejs programów użytkowych (API)

Interfejs Programowania Aplikacji (ang. Application Programming Interface) jest to interfejs między aplikacją a systemem zewnętrznym. Ma on na celu ułatwienie implementacji komunikacji pomiędzy programami, systemami lub usługami oraz ukrycie szczegółów implementacji. Interfejsem tym jest najczęściej zbiór bibliotek zawierających funkcje obsługi systemu zewnętrznego. Systemem zewnętrznym może być system operacyjny, baza danych i itp. Gdy systemem zewnętrznym jest DBMS to API udostępnia w aplikacji funkcje odpowiedzialne za otwarcie i zamknięcie połączenia, wykonanie polecenia (np. SQL), procedur, wyzwalaczy itp. Jednym z najpopularniejszych API dla baz danych jest interfejs ODBC (ang. Open DataBase Connectivity, otwarte łącze baz danych) firmy Microsoft. Większość programów łączy się bazami danych za pomocą API dostarczonych od producentów baz danych lub stworzonych przez inne przedsiębiorstwa. Powodem takiej sytuacji jest wielokrotne wykorzystanie raz napisanego kodu i uproszczenie wymiany danych pomiędzy programem i bazą danych. Dodatkowo najpopularniejsze API umożliwiają połączenie z najpopularniejszymi bazami danych bez zmiany wywoływanych funkcji.

Założmy że posiadamy aplikację obsługującą dział księgowości przechowującą dane w relacyjnej bazie danych MySQL. Z pewnych względów musimy zmienić bazę na relacyjną bazę danych MS SQL. Ponieważ są to różne bazy danych (inna implementacja modelu relacyjnego) musimy rozwiązać szereg problemów. Dla nas najistotniejszym w tym przykładzie będzie sposób łączenia się z bazą danych. Jeżeli nasza aplikacja do obsługi działu księgowości nie korzystała z API np. ODBC to będzie trzeba przepisać na nowo aplikację w miejscach korzystających z bazy danych. W przypadku korzystania z API wystarczy poinformować je o zmianie bazy danych, a o resztę ono zadba np. o emulację transakcji jeśli baza ich nie obsługuje.

Interfejs programowania aplikacji jest bardzo użyteczny w sieciowych systemach baz danych. Programista nie musi martwić się o sposób przesyłania informacji w środowisku sieciowym [6]. Jedyną dodatkową pracą, którą musi wykonać to podać: adres bazy, port, użytkownika, hasło, nazwę bazy i korzystać przy komunikacji z bazą z funkcji API z którego korzysta np. ODBC.

2.3. Wrapper (osłona, translator)

Osłona przystosowuje lokalne API danego serwera do modelu kanonicznego i wspólnego języka zapytań. W ten sposób heterogeniczne dane znajdujące się na różnych serwerach są przystosowane do przetwarzania przy pomocy języka zapytań. Osłona dokonuje prostego odwzorowania struktur danych nie zmieniając ich schematu pojęciowego. Moduł przetwarzania zapytań zainstalowany na serwerze umożliwia definiowanie perspektyw. Perspektywa na danym serwerze przystosowuje obiekty danego serwera do schematu federacyjnej bazy danych opartego na modelu kanonicznym, czyli zmienia schemat pojęciowy lokalnych danych danego serwera widziany z punktu widzenia federacji. Lokalne aplikacje działające na lokalnym serwerze pozostają oczywiście bez zmian. Tradycyjnie, definicje perspektyw są własnością języka zapytań i są dostępne wyłącznie w języku zapytań. Ta własność nie jest obligatoryjna. Np. w systemach opartych na standardzie CORBA definicje perspektyw mają charakter wyrażeń IDL (ang. *Interface Definition Language*) definiujących koncepcyjny obraz obiektów CORBA. Ten obraz jest następnie implementowany w klasycznym lub obiektowym języku programowania, np. w C++, poprzez specjalne fragmenty oprogramowania zwane adapterami (adapters) lub osłonami (wrappers).

Funkcjonuje także termin mediator, który zdaniem jego pomysłodawców posiada coś „inteligentnego” w swojej konstrukcji. Granica pomiędzy tymi pojęciami jest jednak rozmyta i nie wiadomo do którego momentu pewien kod można nazwać „osłoną”, od którego „mediatorem”, a od którego „perspektywą”. Istnieją prace dotyczące standardów dla mediatorów, które utożsamiają pojęcie mediatora i perspektywy [7].

3. Synchronizacji danych pomiędzy bazami danych

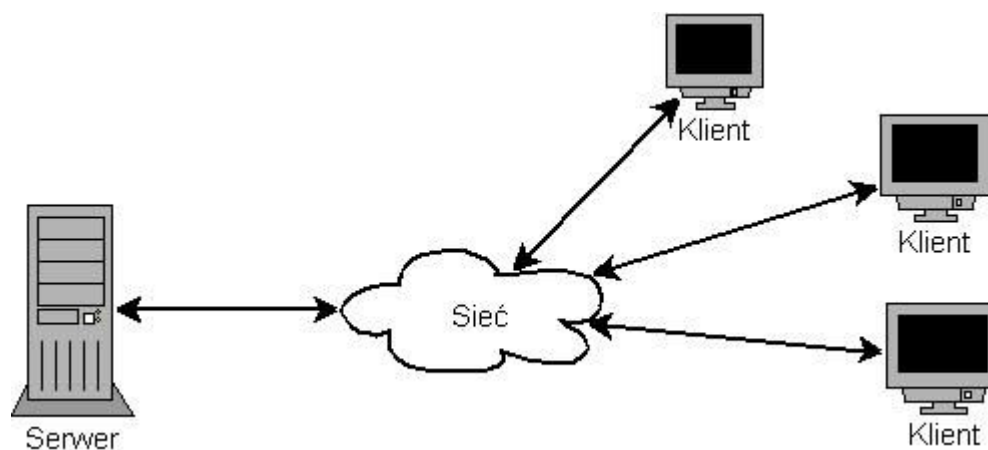
3.1. Systemy rozproszone

Systemem rozproszonym nazywamy taki system, w którym przetwarzanie informacji odbywa się na wielu komputerach, często znacznie oddalonych geograficznie (od kilku metrów do dziesiątków tysięcy kilometrów) [27].

Rozproszeniem danych określamy fragmentaryzację danych (ich podział) lub ich replikację do różnych konfiguracji sprzętowych i programistycznych. O rozproszeniu możemy też mówić w odniesieniu do funkcji czego przykładem są systemy typu klient-serwer (przykładem są strony WWW). Obecnie prawie wszystkie systemy są systemami rozproszonymi. Katalizatorem do tej zmiany był Internet. Umożliwił on łatwą komunikację, a co za tym idzie budowę tanich, szybkich, uniwersalnych i niezawodnych sieci komputerowych. Przykładem systemów rozproszonych są: sieci bankomatów, systemy rezerwacji biletów, sieci P2P itp.

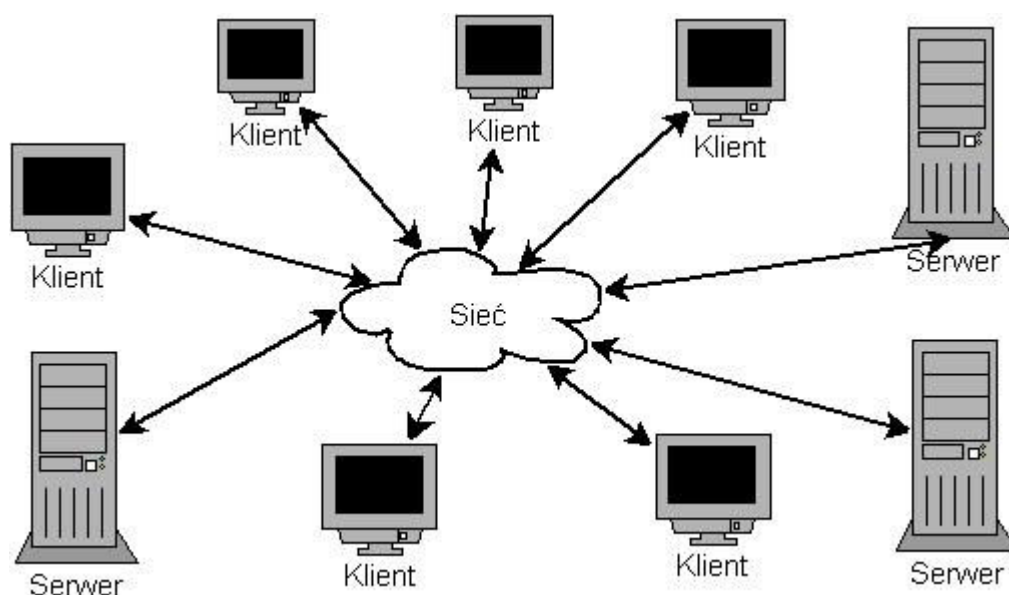
Popularne architektury rozproszenia [27]:

- Klient-serwer (rys. 3.1): rozproszony system ma wyróżniony węzeł zwany serwerem oraz szereg podłączonych do niego węzłów zwanych klientami. Związek nie jest symetryczny: serwer wykonuje usługi zlecane przez klientów, nie może im odmówić i nie może im zlecić wykonanie usług.



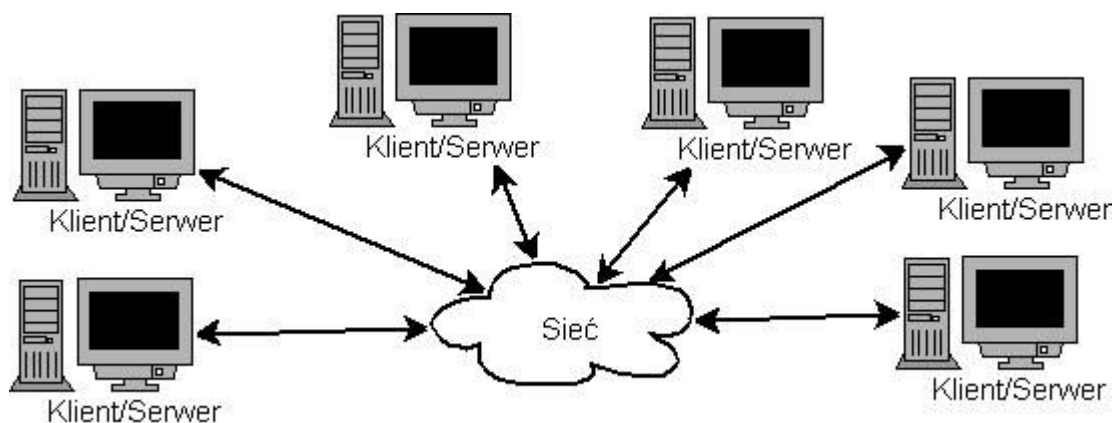
Rys. 3.1. Architektura klient-serwer [28]

- Klient-multi-serwer (rys. 3.2): jest rozwiązaniem architektury klient-serwer, w której istnieją przynajmniej dwa serwery np. WWW.



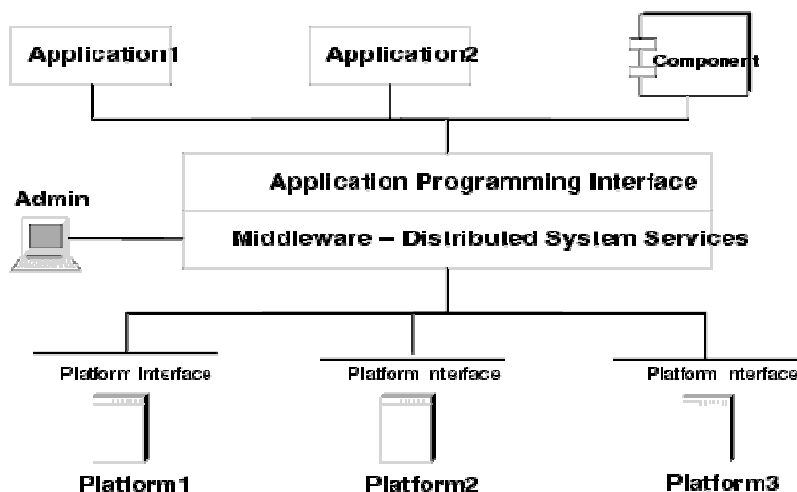
Rys. 3.2. Architektura klient-multi-serwer [28]

- Koleżeńska (P2P ang. *peer-to-peer*, rys. 3.3): wiele węzłów świadczy sobie wzajemne usługi poprzez bezpośrednie połączenie; nie ma wyraźnego podziału na usługodawców i usługobiorców. Np. Gnutella, Napster, Kazaa.



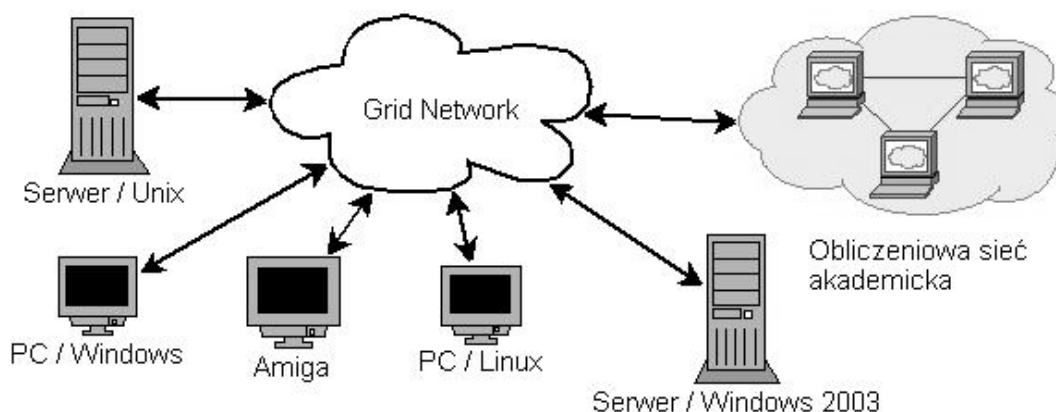
Rys. 3.3. Architektura koleżeńska

- Architektura oparta na oprogramowaniu pośredniczącym (ang. *middleware*, rysunek 3.4): nie występuje podział na klientów i serwery. Aplikacje komunikują się z oprogramowaniem pośredniczącym, które integruje sprzęt, systemy operacyjne, bazy danych, aplikacje i dostarcza potrzebnych danych.



Rys. 3.4. Architektura oparta na oprogramowaniu pośredniczącym [29]

- Architektury gridowe (rys. 3.5): wirtualny komputer sumujący zasoby wielu komputerów w sposób przezroczysty dla użytkowników.



Rys. 3.5. Architektura grid

3.2. Rozproszone bazy danych

Rozproszona baza danych: jest to zbiór składający się z wielu logicznie ze sobą powiązanych elementów bazy danych, oddalonych geograficznie i połączonych ze sobą poprzez sieć komputerową [27].

Zasadniczym jej celem jest to aby dla użytkownika wyglądała jak jedna scentralizowana baza danych. Powinny mieć ona trzy rodzaje przezroczystości:

1. Przezroczystość geograficzną. Użytkownicy nie muszą zdawać sobie sprawy z istnienia bazy rozproszonej. Dla nich wyniki zapytań prezentowane są tak jak by znajdowały się w jednym miejscu.
2. Przezroczystość fragmentaryzacji. Użytkownicy nie wiedzą czy dane są podzielone i jak są podzielone.

3. Przezroczystość replikacji. Użytkownicy nie muszą wiedzieć jak i kiedy dane są powielane.

Można to zobrazować wyobrażając sobie firmę obuwniczą posiadającą pięć oddziałów i centralę. Każdy oddział posiada swoją lokalną bazę danych opisującą działalność oddziału np. sprzedaż butów, a centrala replikę wszystkich danych księgowych (z każdego oddziału). Nasze dane dotyczące stanów magazynowych są pofragmentowane pomiędzy oddziałami. Analityk z centrali potrzebuje dowiedzieć się jakie są stany magazynowe półbutów w oddziałach. Zadając pytanie bazie danych nie zdaje sobie sprawy z potrzeby skorzystania danych z pięciu oddziałów (przezroczystość geograficzna) a pomimo to otrzymuje potrzebne dane.

Rozproszone bazy danych są znacznie bardziej skomplikowane od scentralizowanych baz danych, a szybkość komunikacji sieciowej jest wolniejsza niż lokalna pomiędzy procesorem, pamięcią ulotną i pamięcią masową. Jednak te wady są mało istotne w porównaniu do zalet którymi są:

1. Odwzorowanie podziału geograficznego organizacji. Jeżeli organizacja ma charakter narodowy lub ponadnarodowy to system komputerowy może odwzorowywać podział geograficzny organizacji na departamenty, oddziały itp.
2. Przechowywanie danych w miejscu gdzie są one potrzebne umożliwia lepszą kontrolę na tymi danymi. Dane o asortymencie butów w oddziale w Polsce może edytować tylko personel tego działu.
3. Replikacja danych zwiększa niezawodność systemu. Jeżeli organizacji np. bank musi pracować nieprzerwanie to awaria bazy danych może poważnie zagrozić przedsiębiorstwu (finansowo, reputacji itp.). Dlatego powszechne jest utrzymywanie dwóch bliźniaczych baz danych. Głównej (ang. *master*) i zapasowej (ang. *slave*) będącej aktualną kopią bazy głównej. W chwili awarii głównej bazy danych jej obowiązki natychmiast przejmują zapasowa baza danych.
4. Rozproszenie danych umożliwia zwiększenie wydajności systemu. Jeżeli większość zapytań będzie kierowana do lokalnych baz danych to szybkość wyszukiwania i aktualizacji informacji będzie większa niż do jednej dużej centralnej bazy danych. Ponieważ opóźnienia w komunikacji sieciowej rosną wraz z odległością.

3.2.1. Transakcje rozproszone

Jeśli chcemy, żeby grupa operacji dotyczących kilku środowisk była chroniona przez transakcję, musimy użyć transakcji rozproszonych [10] będących konsekwencją rozproszenia danych. Mogą one obejmować procesy w wielu lokalizacjach. Dlatego model transakcji musiał ulec zmianie. Nie jest to już jeden fragment kodu wykonywany przez jeden procesor, komunikujący się z jednym planistą i z jednym menedżerem dziennika. Rozproszona transakcja składa się z komunikujących się *składowych transakcji* znajdujących się w różnych lokalizacjach i komunikujących się z lokalnym planistą i z menedżerem dziennika.

Pierwszym problemem jaki się nasuwa to zatwierdzanie transakcji rozproszonych. Jak już pisałem w rozdziale 2.1.2 dotyczącym transakcji muszą być one zatwierdzone w całości lub odrzucone. W rozproszonych bazach danych system zarządzanie bazą danych korzysta ze skomplikowanego protokołu, dzięki któremu może zdecydować czy zatwierdzić transakcję rozproszoną czy ją odrzucić. By zaprezentować omawianą ideę skupię się na najczęściej stosowanej koncepcji *zatwierdzania dwufazowego*. W tym celu musimy zdefiniować następujące założenia:

- Wszystkie bazy danych wchodzące w skład transakcji rozproszonej mogą zapewnić atomowość lokalnym transakcją. Co pozwoli nam zachować atomowość transakcji globalnej.
- Nie ma dziennika globalnego. Każda lokalna baza danych prowadzi lokalny dziennik czynności.
- Każda transakcja ma swojego *koordynatora* decydującego czy transakcję zatwierdzić czy odrzucić. Koordynatorem może być np. lokalizacja rozpoczynająca transakcję globalną.
- Protokół zatwierdzania dwufazowego wymaga przesyłania komunikatów między koordynatorem i innymi lokalizacjami. Dlatego wymaga się by każdy komunikat był zapisany w lokalizacji, która go wysłała. Dzięki czemu w razie konieczności można odtworzyć z dziennika dane transakcji.

Transakcja rozproszona zatwierdzania dwufazowego składa się z dwóch części. Fazy pierwszej zwanej głosowaniem oraz fazy drugiej zwanej zakończeniem [30]. Prześledźmy algorytm zatwierdzania dwufazowego [2].

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Faza I:

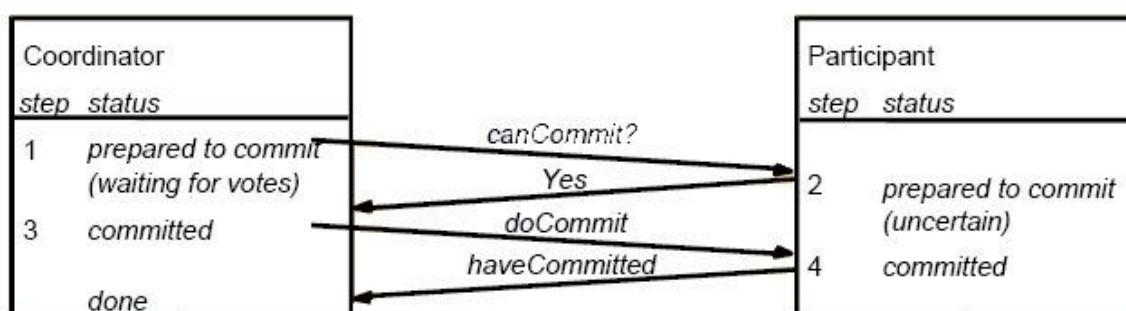
1. Koordynator umieszcza w swoim lokalnym dzienniku zapis przygotować do zatwierdzenia transakcję.
2. Koordynator wysyła komunikat przygotować się do zatwierdzenia transakcji do wszystkich lokalizacji ze składowymi transakcji (także do siebie).
3. W każdej lokalizacji, po otrzymaniu komunikatu podejmuje się decyzję czy zatwierdzić lokalną transakcję czy też ją przerwać. Jeśli transakcja jeszcze się nie zakończyła to można opóźnić podjęcie decyzji, ale trzeba wysłać odpowiedź.
4. Jeśli lokalizacja chce zatwierdzić transakcję to uzyskuje stan *przedzatwierdzeniem*. W takim przypadku transakcja nie może zostać przerwana chyba, że taką decyzję podejmie koordynator. Aby transakcja uzyskała status *przedzatwierdzeniem* musi wykonać następujące czynności:
 - a. Wykonać czynności zapewniające, że lokalna transakcji nie zostanie przerwana nawet jeśli nastąpi awaria systemu lokalnego.
 - b. Umieścić wpis transakcja gotowa do zatwierdzenia w dzienniku lokalnym.
 - c. Do koordynatora należy przesłać komunikat transakcja gotowa do zatwierdzenia.
5. Jeśli lokalna transakcja ma zostać przerwana należy w dzienniku zapisać nie zatwierdzać transakcji i przesłać komunikat o nie zatwierdzeniu składowej globalnej transakcji do koordynatora. Wówczas można bezpiecznie przerwać lokalną transakcję, ponieważ jeśli co najmniej jedna składowa transakcji globalnej zostanie przerwania, to przerywa się całą globalną transakcję.

Faza II:

1. Jeśli od wszystkich składowych transakcji globalnej przyszły komunikaty transakcja lokalna gotowa do zatwierdzenia to koordynator podejmuje decyzję o zatwierdzeniu całej transakcji. Wówczas koordynator:
 - a. Zapisuje w swoim dzienniku lokalnym zatwierdzenie transakcji globalnej
 - b. Przesyła komunikat zatwierdzenia transakcji lokalnych do wszystkich lokalizacji baz danych objętych transakcją.
2. Jeśli koordynator z jednej lub wielu baz danych otrzymał komunikat nie zatwierdzać to:
 - a. W swoim lokalnym dzienniku zapisuje przerwać transakcję.

- b. Przesyła komunikat przerwać transakcję do wszystkich baz danych objętych transakcją.
3. Jeśli lokalizacja będąca składową transakcji otrzymuje komunikat zatwierdź transakcję, to transakcja zostaje zatwierdzona i w dzienniku lokalizacji zapisuje się zatwierdzono transakcję.
4. Jeśli lokalizacja otrzymuje komunikat przerwania transakcji, to transakcja lokalna zostaje wycofana i w dzienniku lokalizacji zapisuje się przerwano transakcję.

Na rysunku 3.6 został pokazany plan zatwierdzania dwufazowego.



Rys. 3.6. Plan rozproszonej transakcji zatwierdzania dwufazowego [30]

Najpopularniejszymi typami rozproszonych baz danych są:

1. System typu klient-serwer
2. Jednorodna rozproszona baza danych
3. Niejednorodna rozproszona baza danych (heterogeniczne)
4. Federacyjny system baz danych

Zanim omówimy wymienione typy rozproszonych baz danych musimy zapoznać się z replikacją danych.

3.2.2. Replikacja danych

Replikacja jest procesem powielania danych, w efekcie której w różnych węzłach rozproszonej bazy danych lub sieci komputerowej otrzymujemy kopie tych samych zestawów rekordów danych. Węzłem będziemy nazywać bazę danych będącą składową środowiska replikacji. Proces replikacji jest niezbędny wszędzie tam, gdzie absolutnie nie możemy sobie pozwolić na utratę danych i zwykle kopie są nie wystarczające np. banki. Oprócz tego jej zaletą jest skrócenie czasu dostępu do danych, uniezależnienie się od przepustowości sieci i wydajności zdalnych węzłów. Wadą replikacji jest konieczność uaktualniania repliki. Taki

proces nazywamy synchronizacją (ang. *synchronization*) lub odświeżaniem (ang. *refreshing*) repliki.

Z synchronizacją repliki związane są dwie kwestie techniczne: moment synchronizacji i sposób synchronizacji. Ze względu na moment synchronizacji odświeżanie dzielimy na synchroniczne i asynchroniczne. W replikacji synchronicznej proces synchronizacji następuje w chwili modyfikacji danych, a przesłanie i zastosowanie zmian są częścią transakcji węzła lokalnego (lokalnej bazy danych). Zatwierdzenie transakcji lokalnej oznacza, że zmiany zostały z powodzeniem przesłane i zaaplikowane we wszystkich pozostałych węzłach środowiska (w bazach na które odbywała się replikacja). Wystąpienie błędów przy przesyłaniu zmian do któregośkolwiek z węzłów (np. na skutek awarii sieci) powoduje niemożność dokończenia lokalnej transakcji węzła, który zapoczątkował cały proces i w konsekwencji jej wycofanie. Dlatego replikacja synchroniczna wymaga stałych i niezawodnych połączeń pomiędzy węzłami środowiska.

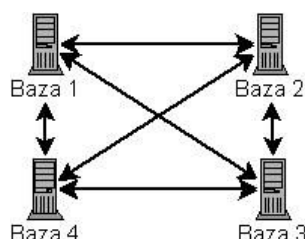
Natomiast replikacja asynchroniczna wszystkie lokalne zmiany danych z zatwierdzonej transakcji węzła lokalnego najpierw składa w lokalnej strukturze zwanej kolejką transakcji, a następnie okresowo pobiera i przesyła dane do pozostałych węzłów środowiska. Operacje przesłania i zaaplikowania zmian są realizowane jako osobna transakcja (na każdą kolejną replikę). Replikacja asynchroniczna w odróżnieniu od synchronicznej jest nie wrażliwa na utratę połączenia pomiędzy niektórymi węzłami środowiska. Ponieważ nie powoduje to wstrzymania replikacji pomiędzy pozostałymi węzłami oraz przerywania lokalnych transakcji w węźle. Replikacja asynchroniczna może być wykonywana okresowo (z zadany interwałem czasowym) lub manualnie (uruchomiona przez człowieka) [31].

Ze względu na sposób synchronizacji dzielimy replikację na pełną (ang. *complete, full*) i przyrostową (ang. *incremental, fast*). W pierwszym wypadku odświeżanie polega na każdorazowym skopiowaniu wszystkich danych źródłowych. W odróżnieniu od pełnej replikacji replikacja przyrostowa polega na kopiowaniu wyłącznie zmian danych źródłowych, które wystąpiły od czasu ostatniej synchronizacji.

Dzieląc replikację ze względu na użyte techniki wyróżniamy dwie grupy: Do pierwszej grupy należy analiza logu transakcji bazy danych oraz przenoszenie zapisanych w tym logu transakcji do innego serwera baz danych. Natomiast w drugiej grupie znajduje się reagowanie na zmiany danych poprzez wyzwalacze, zapis powstałych zmian w dodatkowej bazie danych i późniejsze ich przesyłanie do innego serwera przez tzw. proces replikacyjny [11].

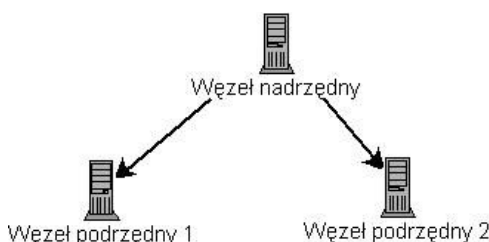
W zależności od wzajemnych relacji między węzłami baz danych uczestniczących w replikacji wyróżniamy następujące typy środowisk replikacji: *replikację multimaster*, *replikację migawkową* oraz *replikację hybrydową*.

W środowisku *replikacji multimaster* (ang. *multimaster replication*) wszystkie węzły replikacji są równoważne czyli każdy węzeł jest węzłem nadrzędnym (ang. *master*). Oznacza to że jakakolwiek zmiana w którymkolwiek węźle sieci jest natychmiast uwzględniana we wszystkich węzłach środowiska lub wcale. Wszystkie replikowane dane są identyczne we wszystkich węzła zarówno jeśli chodzi o wartość jak i strukturę danych. Przykładowe środowisko replikacji multimaster przedstawia rysunek 3.7.



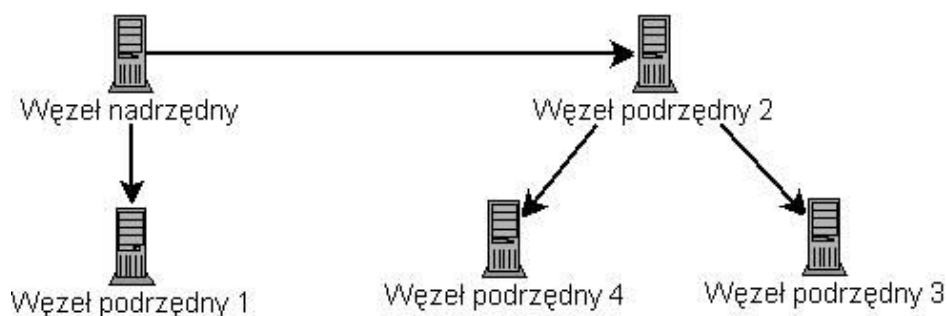
Rys. 3.7. Przykładowe środowisko replikacji multimaster

Środowisko *replikacji migawkowej* (ang. *materialized view replication*) zazwyczaj tworzy jeden węzeł nadrzędny (ang. *master*) oraz wiele węzłów migawkowych określanych także jako podrzędnych (ang. *slave*). Dane z bazy źródłowej (nadrzędnej) są źródłem danych dla węzłów migawkowych (podrzędnych) należących do grupy replikacji. Taką sytuację przedstawia rysunek 3.8. W architekturze wielowarstwowej węzeł migawkowy (podrzędny) może być także węzłem nadrzędnym będącym źródłem dla innych migawek.



Rys. 3.8. Przykład środowiska replikacji migawkowej

Replikacja hybrydowa (ang. *hybrid replication*) jest kombinacją środowisk replikacji multimaster i replikacji migawkowej. Środowisko replikacji hybrydowej zawiera wiele węzłów nadrzędnych wraz z wieloma węzłami migawkowymi (podrzędnymi). Przykładowe środowisko replikacji hybrydowej przedstawia rysunek 3.9.

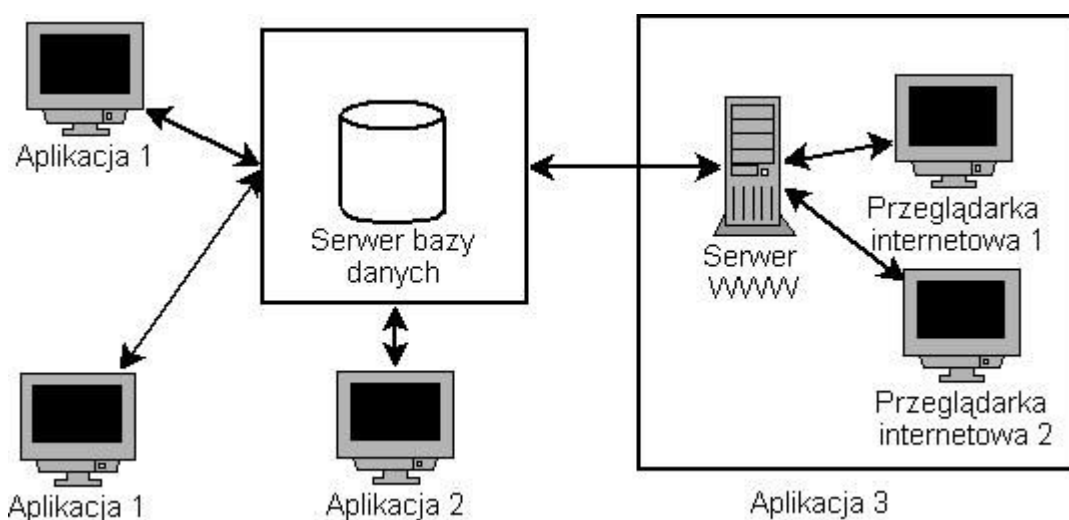


Rys. 3.9. Przykładowe środowisko replikacji hybrydowej

Rozmieszczenie kopii danych w wielu węzłach sieci stwarza problem uaktualniania danych, tak by transakcje realizowane w różnych węzłach sieci operowały na zestawie takich samych rekordów. Dobór odpowiedniego sposobu powielania danych i przyjęcie właściwej strategii ich aktualizacji uzależnione jest od zadań stawianych systemowi przez jego użytkowników [32].

3.3. System typu klient-serwer

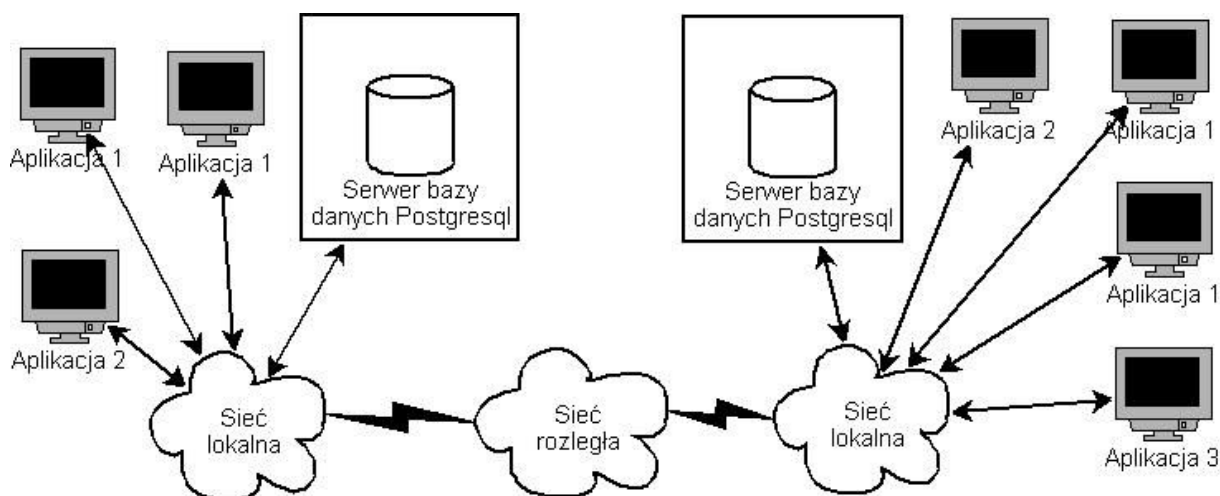
Termin *klient-serwer* nie ma jednoznacznego znaczenia. Dla niniejszego opracowania przyjmijmy, że rozproszona baza danych typu klient serwer jest to system bazy danych obsługujący wielu klientów poprzez sieć. System ten jest najtańszą i najprostszą rozproszoną bazą danych. Dostarcza on tylko informacji przechowywanych w bazie danych, a cała logika biznesowa jest zaimplementowana w aplikacjach łączących się z bazą danych. Przykładowy system typu klient serwer przedstawia rysunek 3.10.



Rys. 3.10. Przykładowy system typu klient-serwer

3.4. Jednorodna rozproszona baza danych

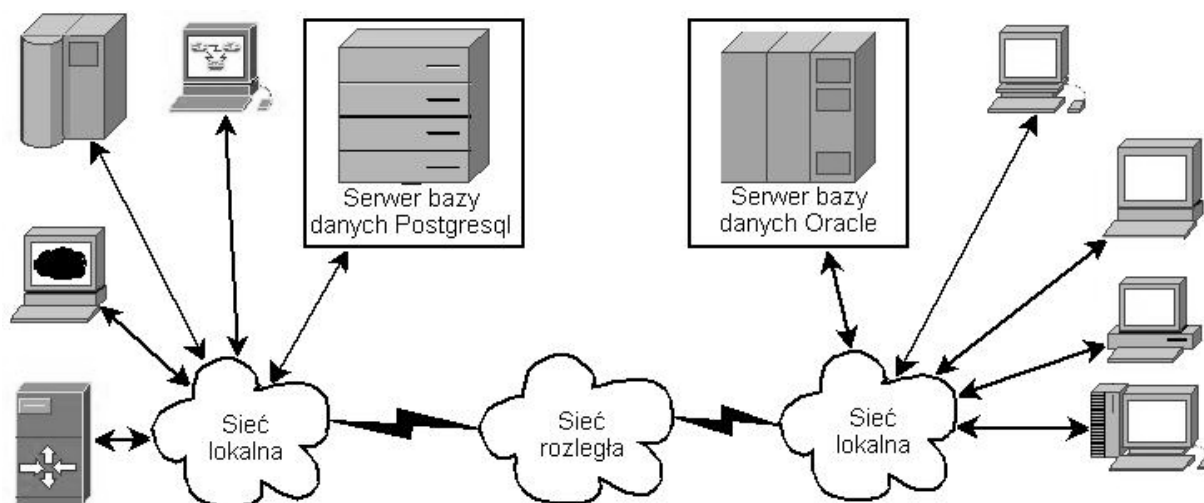
Systemy typu klient-serwer zazwyczaj działają na krótkich łączach sieciowych i na jednej bazie danych. Rozproszone systemy zawierają wiele serwerów baz danych i na ogół na odległych połączeniach. Jednorodne rozproszone bazy danych są rozłożone pomiędzy dwa lub więcej systemów, w którym każdy oparty jest na tym samym rodzaju systemu zarządzania bazą danych. Najczęściej jednorodny rozproszony system bazy danych działa także na takim samym rodzaju sprzętu i systemu operacyjnego. Przykładowy systemem typu klient-serwer został pokazany na rysunku 3.11. W systemie tym są dwie bazy danych PostgreSQL działająca pod kontrolą systemu operacyjnego linux i pracująca na komputerze rodziny x86.



Rys. 3.11. Przykładowa jednorodna rozproszona baza danych

3.5. Niejednorodna rozproszona baza danych

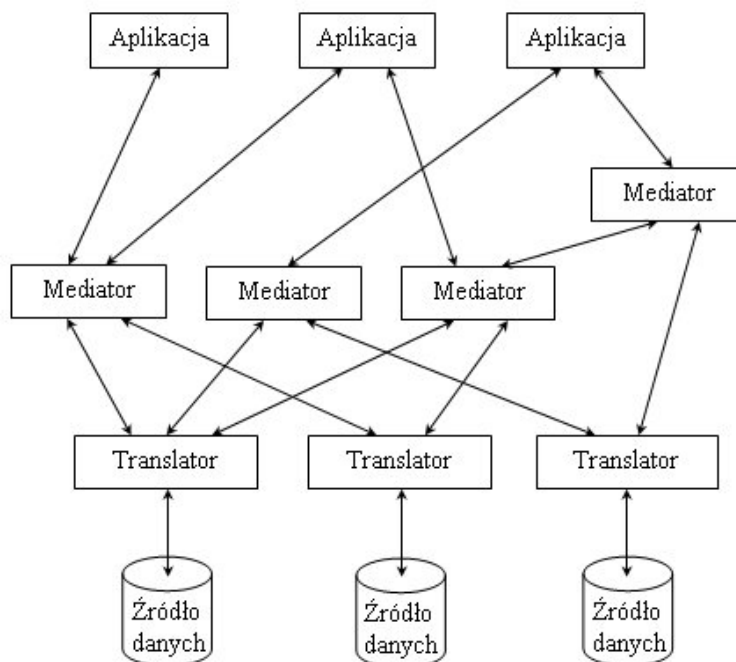
W niejednorodnej rozproszonej bazie danych często zwanej także heterogeniczną konfiguracje sprzętowe i systemy operacyjne są różne. Przykładowy system niejednorodnej rozproszonej bazy danych został pokazany na rysunku 3.12. W polskim oddziale firmy może działać baza danych MS SQL na komputerze rodziny x86 z systemem operacyjnym Windows 2000, a w niemieckim oddziale firmy jest baza danych Oracle na komputerze rodziny SPARC z systemem operacyjnym Linux.



Rys. 3.12. Przykładowa niejednorodna rozproszona baza danych

3.6. Mediatory

Mediator jest to oprogramowanie wykorzystujące wiedzę związaną z określonym obszarem zastosowań do wyboru źródeł informacji i uzyskania z nich informacji w celu przekazania ich pytającym o nie aplikacjom. Wspomagają one tworzenie wirtualnych perspektyw integrujących wiele źródeł. Analogicznie do urzeczywistnionych relacji w hurtowni danych integrujących wiele źródeł danych. W odróżnieniu od hurtowni danych mediatory nie przechowują żadnych danych. Tylko umożliwiają przeźroczysty dostęp do niejednorodnych źródeł informacji, wśród których mogą również znaleźć się źródła, które nie są systemami bazy danych.



Rys. 3.13. Systemy oparte na architekturze mediator [33]

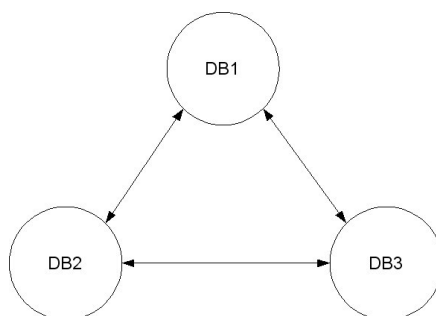
Przykładowy system oparty o architekturze mediatora pokazano na rysunku 3.13. Bloki symbolizujące translator są osłonami opisanymi w rozdziale 2.3.

Na początku użytkownik (np. poprzez aplikację) wysyła zapytanie do mediatora. Ponieważ mediator nie posiada własnych danych na podstawie zapytanie użytkownika decyduje z jakiego źródła danych skorzystać i jak zapytać translator o dane potrzebne do skonstruowania odpowiedzi dla aplikacji (użytkownika). Mediator w celu utworzenia odpowiedzi może pytać dowolną ilość wrapperów dowolną ilość razy. Sensowne jest stworzenie takiej logiki mediatorów by jak najmniejszym kosztem (ilością zapytań i ilością przetworzonych danych) były w stanie skonstruować odpowiedź. Ta cecha jest często utożsamiana z inteligencją mediatorów.

3.7. Federacyjne bazy danych

Jedną z prostszych metod zintegrowania wielu baz danych sprowadza się do utworzenia połączeń jeden do jeden pomiędzy wszystkimi bazami danych, które mają się komunikować ze sobą. Takie połączenia umożliwiają bazie danych (np. DB1) wysłać zapytanie do innej bazy danych (np. DB2) sformułowane za pomocą pojęć które rozumie system docelowy (DB2). Taką architekturę połączeń baz danych nazywamy federacyjną bazą danych. Głównym problemem takiego systemu komunikacji jest ilość fragmentów kodu służących do obsługi zapytań. Jeżeli mamy n baz danych, w których każda baza chce komunikować się z $n - 1$ baz to ilość fragmentów kodu służącego do komunikacji wynosi

$n(n-1)$. Taką sytuację ilustruje rysunek 3.14. Przedstawia on trzy bazy danych, w których każda z baz chce komunikować się dwoma pozostałymi. W sumie potrzeba sześciu fragmentów kodów umożliwiających komunikację.



Rys. 3.14. Federacyjna baza danych złożona z trzech baz danych

Transakcje w federacyjnej bazie danych możemy podzielić na dwa rodzaje:

1. Lokalne transakcje – są wykonywane autonomicznie przez lokalny DBMS. Federacja nic o niej nie wie i zazwyczaj nie może się dowiedzieć.
2. Globalne transakcje – są wykonywane przez aplikacje globalne. Składają się z szeregu podtransakcji wykonywanych przez lokalne DBMS (dla tej transakcji).

Załóżmy, że nasza federacyjna baza danych przedstawia trzy oddziały firmy sprzedającej buty. Przyjrzyjmy się możliwej strukturze danych w bazie DB1 i DB2 oraz przykładowej komunikacji pomiędzy tymi bazami danych.

Baza DB1 zapisuje informacje o butach jako relację:

```
Buty(model, kolor, rozmiar, typ, producent, ilość)
```

Baza DB2 zapisuje informacje o butach jako dwie relacje:

```
Buty(typ, model, producent)
```

```
Rodzaj(model, kolor, rozmiar, ilość)
```

Jak widzimy struktura baz danych jest niejednorodna. Dlatego baza DB1 chcąc dowiedzieć się jaka jest ilość półbutów (typ) w magazynie w oddziale drugim (DB2) musi sformułować zapytanie tak by baza DB2 je zrozumiała.

```
SELECT SUM(ilość) FROM Buty as b, Rodzaj as r WHERE b.model = r.model  
AND b.typ = `półbuty`
```

W przypadku gdy baza DB2 chce się dowiedzieć ile jest półbutów (typ) w magazynie w oddziale pierwszym (DB1) wysyła do bazy DB1 następujące zapytanie.

```
SELECT SUM(ilość) FROM Buty WHERE typ = `półbuty`
```

Federacyjna baza danych może łączyć odpowiedzi składowych zapytań. W ten sposób w naszej przykładowej federacyjnej bazie danych analityk chcący dowiedzieć się jakie są stany magazynowe pół butów w oddziale pierwszym (DB1) i drugim (DB2) otrzymał by łączną

odpowieź (wykonał by transakcje globalną), a baza danych wykonała by dwa zapytania, których wynik by złączyła.

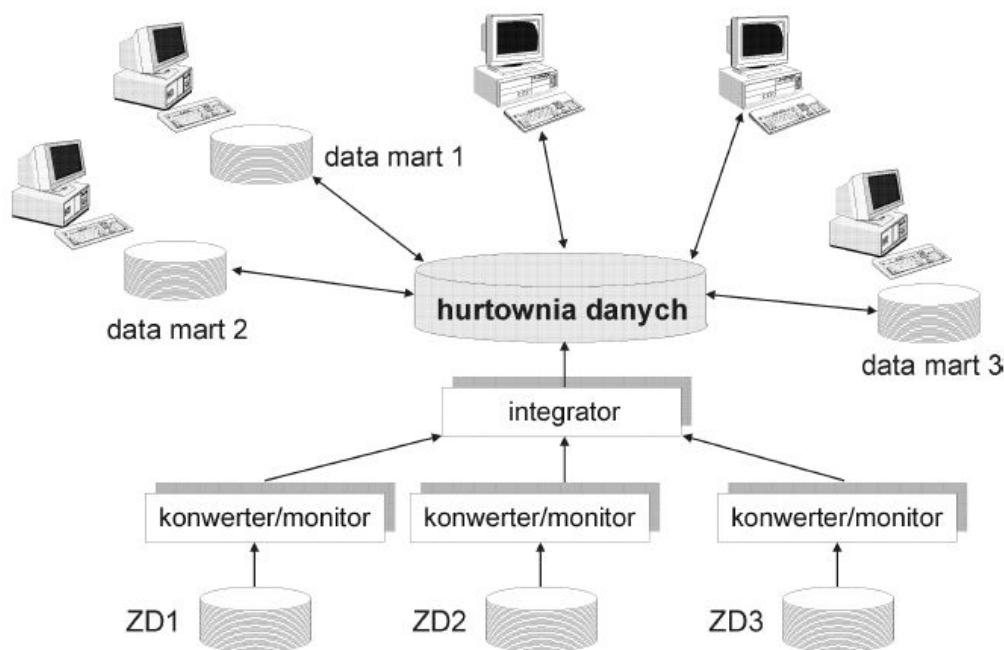
```
(SELECT SUM(ilość) FROM Buty as b, Rodzaj as r WHERE b.model =  
r.model AND b.typ = `półbuty`)  
  
UNION  
  
(SELECT SUM(ilość) FROM Buty WHERE typ = `półbuty`)
```

Tabela 3.1 Wynik złączenia zapytania do dwóch baz federacyjnej bazy danych

Oddział	Ilość półbutów
Polska (DB1)	5002
Niemcy (DB2)	10234

3.8. Hurtownie danych

Hurtownie danych, zwane również magazynami danych (ang. *data warehouses*) stają się obecnie niezbędnym komponentem systemów informatycznych w dużych firmach i instytucjach. Z technologicznego punktu widzenia hurtownia danych jest ogromną bazą danych, do której wczytuje się dane z tzw. produkcyjnych źródeł danych [34]. Dane te często pochodzą z wielu źródeł, są one zintegrowane i przeznaczone wyłącznie do odczytu. W praktyce hurtownie są bazami danych integrującymi dane z wszystkich pozostałych systemów bazodanowych w firmie [35]. Najczęściej hurtownie danych zasilane są w nocy ze względu na dużą ilość danych jakie trzeba przesłać. Na rysunku 3.15 zobrazowany została przykładowy system hurtowni danych.



Rys. 3.15. Podstawowa architektura systemu hurtowni baz danych [34]

Na bazie hurtowni danych pracują aplikacje analityczne (ang. *On-line Analytical Processing* - OLAP) lub eksploatacji danych (ang. *data mining*). Aplikacje te są zorientowane na przetwarzanie danych historycznych i zagregowanych. Operują one najczęściej na milionach przetworzonych (przefiltrowanych, zagregowanych itp.) rekordów. Czas wykonywania zapytań często jest długi od kilku minut do kilku dni. Jednak zakres analizy jaka może być wykonana przekracza możliwości zwykłych baz danych. Hurtownie danych potrafią odpowiedzieć na przykład na takie pytania jak: Jaki jest rozkład sprzedaży półbutów w województwie małopolskim? Jaki jest trend sprzedaży kosmetyków w ostatnich kilkunastu tygodniach?

Tak więc, przetwarzanie w aplikacjach analitycznych charakteryzuje się operacjami odczytu dużych wolumenów danych, przetwarzanych następnie przez złożone funkcje analityczne. Zawartość hurtowni danych poddana złożonej analizie i eksploracji danych staje się bezcennym źródłem wiedzy wykorzystywanej w procesach decyzyjnych. Dzięki szybkiej analizie bazującej na pełnej i aktualnej informacji o stanie firmy, kadra zarządzająca może podejmować właściwe decyzje o strategicznym znaczeniu dla rozwoju danego przedsiębiorstwa [34].

4. Algorytm synchronizacji produktów

Algorytm, który omówię będzie dotyczyć dwóch najpopularniejszych w swoim segmencie programów na polskim rynku. Są nimi Subiekt GT (w wersji 1.10) będący systemem obsługi sprzedaży małych i średnich firm oraz sklepem internetowym osCommerce (w wersji 2.2 Milestone 2 Update 060817).

Producent Subiekta firmy Insert reklamuje swój program następująco [15]:

„Subiekt GT to nowoczesny system obsługi sprzedaży dla małych i średnich firm, stworzony z myślą o firmach poszukujących bardzo przyjaznego, a jednocześnie niezwykle sprawnego narzędzia wspomagającego całościową obsługę działu handlowego, sklepu, punktu usługowego, rzemieślniczego itp. Subiekt GT jest sprawnym i szybkim systemem wyposażonym w najnowsze rozwiązania interfejsowe czyniące go programem bardzo ergonomicznym i przyjaznym dla użytkownika. Jednocześnie jego wszechstronność i możliwości konfiguracji czynią go produktem niemal dla każdego. Subiekt GT jest częścią nowej linii InsERT GT, która zawiera również Rachmistrza GT - program do prowadzenia księgowości w formie książki przychodów i rozchodów lub ewidencji podatku zryczałtowanego oraz Rewizora GT - program do prowadzenia księgi handlowej, który dzięki dołączanemu bezpłatnie modułowi mikroGratyfikant GT umożliwia także podstawową obsługę plac. Motorem bazy danych zastosowanym w programach linii InsERT GT jest Microsoft SQL Server 2005 [36], który w bezpłatnej wersji Express Edition jest dostarczany wraz z systemem.”

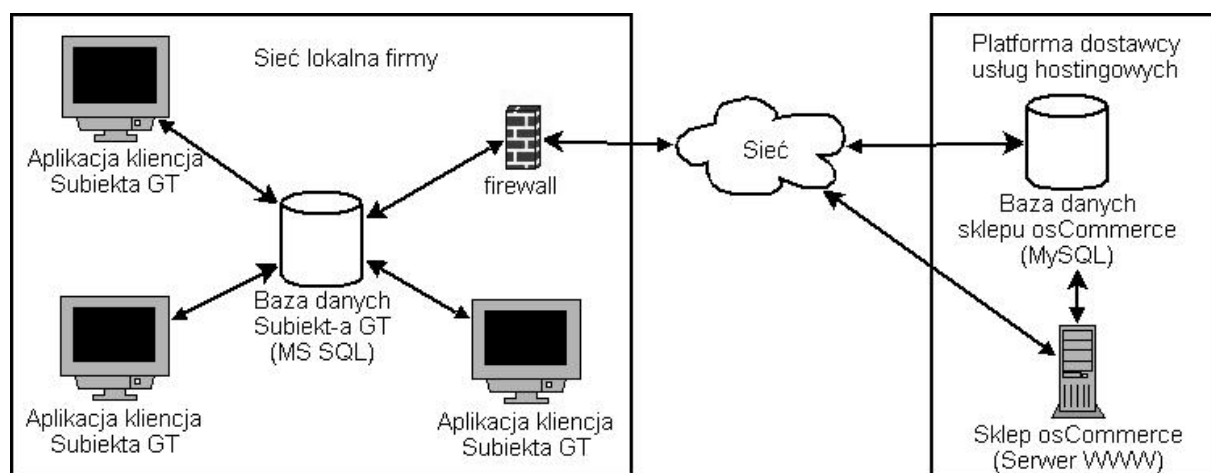
Sklep osCommerce i platforma programistyczna na której został oparty w odróżnieniu od Subiekta udostępniony jest na licencji GPL (ang. *GNU GENERAL PUBLIC LICENSE* [18]) o otwartym kodzie źródłowym (ang. *Open Source*). Oznacza to że możemy używać go w zastosowanych komercyjnych bez żadnych opłat z tego tytułu oraz modyfikować kod pod warunkiem zachowania tej samej licencji. Sam sklep można scharakteryzować następująco [16]:

osCommerce jest darmowym sklepem internetowym posiadającym dużą ilość dodatków i funkcji pozwalających właścicielowi na dostosowanie i utrzymywanie sklepu przy minimalnych kosztach i niewielkim nakładzie pracy. osCommerce skupia dużą społeczność programistów i właścicieli sklepów przekraczającą 100 000 osób. Na stronie projektu zarejestrowanych było w marcu 2007 roku 12 477 sklepów (rejestracja jest dobrowolna). Społeczność skupiona wokół projektu wzajemnie rozwiązuje swoje problemy i rozszerza możliwości sklepu. Językiem programowania użytym do napisania sklepu jest język

skryptowy PHP [37], a dedykowaną bazą danych jest relacyjna baza danych MySQL [38]. Preferowanym serwerem WWW jest Apache [39].

4.1. Założenia

Środowisko pracy programów może być różne w zależności od polityki bezpieczeństwa w firmie. Na rysunku 4.1 został przedstawione najbardziej restrykcyjne środowisko pracy aplikacji Subiekta GT i osCommerce. Spełniając tak sformułowane kryteria zapewnimy poprawną pracę algorytmu w większości przypadków.



Rys. 4.1. Środowisko pracy programów Subjekt GT i osCommerce

Środowisko pracy programu Subjekt GT:

- Program Subjekt GT pracuje z bazą danych MS SQL w wersji 2000 lub 2005.
- Dostawca usług internetowych (ang. *Internet Service Provider*, IPS) nie udostępnia adresu IP (ang. *Internet Protocol address*). Co uniemożliwia nam połączenia się z bazą danych Subiekta z innych lokalizacji niż sieć lokalna.
 - a. W obrębie sieci lokalnej firmy można ustanowić połączenia z bazą danych
 - b. Na serwerze bazy danych lub jednym z klientów można zainstalować program realizujący nasz algorytm

Środowisko pracy sklepu internetowego osCommerce:

- Publiczny adres IP bazy danych,
- Publiczny adres IP serwera WWW,
- Serwer WWW obsługujący PHP w co najmniej wersji trzeciej,
- Baza danych MySQL.

Ponieważ środowiska pracy składowych naszego systemu w którym będzie następowała wymiana informacji jest niejednorodne ale dane o produktach są wspólne

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

dlatego środowiska pracy możemy zakwalifikować do rozproszonych niejednorodnych baz danych. Nasz algorytm synchronizacji produktów musi uwzględniać następujące wymagania:

- Replikować dane migawkowo - bazą danych typu master jest baza Subiekta,
- Replikować dane przyrostowo,
- Konwertować dane z bazy master do bazy slave,
- Replikować tylko dane przeznaczone do sklepu internetowego: dane produktów, typ produktu, kategorie/grupy, stawki podatkowe.

Typ produktu rozumiemy jako wariant tego samego produktu. Nie występuje on w Subiekcie w formie nam odpowiadającej. Dlatego zakładamy, że symbol produktu używany w Subiekcie traktujemy jako złożenie symbolu i typu oddzielonego znakiem specjalnym („symbol” „znak specjalny” „symbol typu”). Przykładowo jeżeli mamy samochód o nazwie handlowej ECAR, który może być w trzech kolorach biały (typ B), zielony (typ Z), niebieski (typ N) to w Subiekcie musimy dodać trzy produkty (pozycje) o symbolach ECAR-B, ECAR-Z, ECAR-N, a w sklepie będziemy widzieć jeden produkt o symbolu ECAR, który będzie posiadał trzy opcje do wyboru B, Z i N. Jeżeli dla samochodu ECAR-B wpisujemy w Subiekcie w polu opis nazwę biały to w sklepie będziemy mieć następujące opcje do wyboru: biały, Z, N.

Każdy typ produktu posiada następujące cechy: cena detaliczną netto, cena hurtowa netto, ilość, symbol typu (np. B), opis typu (np. biały).

Dane produktu w sklepie traktujemy jako sumę danych poszczególnych typów. W przypadku dublowania się danych pierwszeństwo mają dane o wcześniejszych literach alfabetu. Danymi produktu są: symbol, nazwa, opis, cenna detaliczna netto najtańszego typu, cenna hurtowa netto najtańszego typu, stawka podatkowa, sumaryczną ilość sztuk poszczególnych typów, kategoria, adres odnośnika do strony WWW o produkcie. Przykładowo założymy że mamy następujące produkty w Subiekcie.

Symbol	Nazwa	Opis	Ilość sztuk	Cenna detaliczna netto
ECAR-B		Super ekologiczny samochód spalający tylko 2l/100km	1	200 000,00
ECAR-Z	Ekologiczny samochód	Samochód spalający 10l/100km	10	210 000,00

Po zsynchronizowaniu baz danych w sklepie mamy widzieć produkt „Ekologiczny samochód” o symbolu „ECAR” i opisie „Super ekologiczny samochód spalający tylko 2l/100km”, którego minimalna cena wynosi 200 000,00 i 11 sztukach w magazynie. Samochód ma dwa warianty B (1 sztuka o cenie 200 000,00) i Z (10 sztuk o cenie 210 000,00).

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Obsługa grup/kategorii produktów w sklepie obsługuje dowolną ilość zagłębień kategorii. Ma on także możliwość przypisania do niej zdjęcia oraz opisu we wszystkich zdefiniowanych w sklepie językach (np. polski, angielski, niemiecki). Subiekt umożliwia obsługę grup, ale nie ma możliwości ustalenia podgrup. W przypadku sklepów internetowych jest to nie dopuszczalne, ponieważ nawigacja po takim sklepie była by bardzo niewygodna. Dlatego założyliśmy trzy poziomy kategorii specjalnie definiowane w Subieckie. Pole grupa w Subieckie określa dwa poziomy kategorii rozdzielone znakiem specjalnym (np. Samochody – Osobowe). Pierwsza nazwa przez znakiem specjalnym jest kategorią nadrzędną, a druga podrzędną. Załóżmy że chcemy stworzyć następujące kategorie:

<u>Kategoria główna</u>	<u>Podkategoria</u>	<u>Zdefiniowana grupa w Subieckie</u>
Samochody		(nie wymagane)
	- Osobowe	Samochody - Osobowe
	- Ciężarowe	Samochody - Ciężarowe
Rowery		(nie wymagane)
	- Górskie	Rowery - Górskie
	- Szosowe	Rowery - Szosowe
Deskorolki		Deskorolki

Trzeci poziom kategorii jest określany w cechach towaru i brany pod uwagę tylko w sytuacji gdy produkt ma zdefiniowaną dwupoziomową grupę. Jeżeli w cechach towaru dla rowerów górskich wpisujemy wyczynowe i to samo zrobimy dla deskorolek. To w przypadku deskorolek nie powstanie podkategoria. Nasz algorytm ograniczymy tylko do dodawania nowych grup i przypisywania do nich produktów. Usuwaniem grup musi się już zająć pracownik obsługujący sklep usuwając w panelu administracyjnym sklepu kategorie. Powodem takiego założenia jest uniemożliwienie utracenia danych kategorii (zdjęcia i opisów) ze sklepu. Taka sytuacja mogła by zajść gdy przykładowo z kategorii deskorolki pracownik usunął by wszystkie produkty lub gdy żaden z produktów nie byłby przeznaczony do sklepu internetowego. Kolejnym wymaganiem w odniesieniu do grup jest branie pod uwagę tylko grup dla których przynajmniej jeden produkt jest przeznaczony do sklepu internetowego. Jeżeli mamy w Subieckie grupę „Stare produkty” która nigdy nie ma być dostępna w sklepie to produkty przypisane do tej grupy nie mogą być oznaczone „pozycja przeznaczona do sklepu internetowego”.

Każdy produkt w Subieckie i sklepie posiada przypisaną stawkę podatkową. W sklepie będziemy tylko dodawać nowe stawki podatkowe. Z takich samych powodów integralnościowych jak w przypadku kategorii.

4.2. Możliwe sposoby synchronizacji

Przy uwzględnieniu założeń z rozdziału 4.1 mamy tylko jedną możliwość synchronizowania produktów. Jest nią program na serwerze bazy danych lub komputerze w sieci lokalnej firmy mającym połączenie z bazą danych. Może on być uruchamiany automatycznie w zadanych odstępach czasu lub manualnie przez użytkownika. Jego zadaniem jest replikacja przyrostowa danych o produktach. Czyli pobranie ich z bazy źródłowej i docelowej, przetłumaczenie ich na wspólny schemat i znalezienie nieaktualnych danych. Następnie nowe dane przetłumaczyć na schemat bazy docelowej (sklepu) i uaktualnić bazę danych. Algorytmem takiego programu zajmiemy się w rozdziale 4.4, a w rozdziale 5.2 jego implementacją.

W przypadku gdy nad całym procesem eksploatacji bazy danych czuwa administrator. Stale monitorujący zmiany w oprogramowaniu (Subiekta, sklepu, baz danych) i reagujący na możliwe uszkodzenia w czasie rzeczywistym. Można pominąć założenie o braku modyfikacji/ingerencji w bazę danych Subiekta GT. W takiej sytuacji dodatkowo mielibyśmy następujące możliwości synchronizowania produktów:

1. Wyzwalacze SQL logujące zmiany w tabeli tymczasowej. Każda zmiana w produktach, grupach, stawkach podatkowych była by rejestrowana w tabeli tymczasowej, z której program synchronizujący dane pobierał by dane do aktualizacji lub nazwę tabeli i identyfikator zmodyfikowanej krotki. Takie podejście gwarantuje nam zmniejszenie ilości zapytań do obydwu baz i co w konsekwencji skutkowało by zwiększeniem szybkości synchronizacji, zmniejszeniem obciążenia sieci, zmniejszeniem obciążenia bazy danych. Do wad tego rozwiązania należy zaliczyć dodatkowe operacje wstawiania krotek do tabeli tymczasowej co za tym idzie niewielkie spowolnienie (nie większe niż 1s zazwyczaj dziesiątki ms) edycji danych.
2. Poprzez moduły na trwale zapamiętane (PSM), które przy każdej modyfikacji produktu łączyły by się z bazą sklepu i modyfikowały dane tego samego produktu w sklepie. Ze względu na stopień skomplikowania i zależnych czynników zewnętrznych (połączenia sieciowe, platformy sprzętowe) rozwiązanie to ma zwiększoną szansę na desynchronizację danych. Zaletą tego rozwiązania jest synchronizacja danych w czasie rzeczywistym. Ponieważ dla transakcji wymagających wielu odwołań do bazy

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

danych, procedury składowane pozwalają uniknąć opóźnień związanych z funkcjonowaniem sieci komputerowej [12].

Przy pominięciu założenia braku połączenia z bazą danych Subiekta z poza sieci lokalnej otrzymujemy możliwość zintegrowania skryptu synchronizującego ze sklepem. W tej sytuacji można rozważyć następujące możliwości synchronizacji:

1. Uruchamianie synchronizacji produktów cyklicznie w zadanych odstępach czasu,
2. Uruchomienie synchronizacji produktów przez użytkownika

Obydwa sposoby niezależnie od implementacji charakteryzują się następującymi wadami:

- Przy większej ilości produktów należy zadbać o przedłużenie maksymalnego czasu wykonywania się skryptu (zazwyczaj administrator ogranicza czas wykonywania skryptu do 60s z możliwością przedłużenia).
- Zwiększone obciążenia serwera sklepu. W przypadku usług dzielonego hostingu administratorzy nie zgodzą się na działania skryptu.
- Włamanie na serwer sklepu ujawni adres, użytkownika i hasło bazy danych Subiekta i umożliwi kradzież danych.

4.3. Schematy baz danych

Każda z synchronizowanych baz danych była projektowana do innego zastosowania i stworzona została przez innego producenta. Dlatego schemat baz danych, które będą synchronizowane jest innych. Czyli nasz zintegrowany system baz danych będzie niejednorodny. Jedynym wspólnym wykładnikiem jest relacyjny model danych. Przy omawianiu struktury baz danych skupię się tylko na istotnych relacjach opisujących produkty, typy produktów, kategorie produktów, stawki podatkowe.

4.3.1. Schemat bazy danych programu Subiekt GT

Przy eksploatacji danych bazy Subiekta nie korzystamy z perspektyw z uwagi na nieprzewidywalność ich struktury w przyszłości. Podstawowa struktura danych powinna być zachowana w przyszłości dlatego możemy założyć że będzie stała w czasie. W zamieszczonych relacjach zostały pominięte atrybuty nie używane podczas replikacji.

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Tabela 4.1 sl_StawkaVAT - Słownik stawek VAT

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	vat_Id	Identyfikator stawki VAT	int
2	vat_Nazwa	Nazwa stawki VAT	varchar (50)
3	vat_Stawka	Wartość stawki.	money

Tabela 4.2 sl_GrupaTw - Słownik grup towarowych

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	grt_Id	Identyfikator grupy	int
2	grt_Nazwa	Nazwa grupy	varchar (50)

Tabela 4.3 tw_CechaTw - Tabela złączeniowa towaru i cech

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	cht_Id	Identyfikator cechy towaru	int
2	cht_IdTowar	Identyfikator towaru	int
3	cht_IdCecha	Identyfikator cechy ze słownika cech	int

Tabela 4.4 sl_CechaTw - Słownik cech towarów

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	ctw_Id	Identyfikator cechy towaru	int
2	ctw_Nazwa	Nazwa cechy	varchar (50)

Tabela 4.5 tw_Stan - Stany magazynowe towarów

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	st_TowId	Identyfikator towaru	int
2	st_MagId	Identyfikator magazynu	int
3	st_Stan	Ilość towaru na danym magazynie	money
4	st_StanRez	Stan rezerwacji towaru na danym magazynie	money

Tabela 4.6 tw__Towar – Ewidencja towarów

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	tw_Id	Identyfikator towaru	int
2	tw_Zablokowany	Czy towar jest zablokowany (0 - nie; 1 - tak)	bit
3	tw_Rodzaj	Rodzaj towaru (1 - towar; 2 - usługa; 4 - opakowanie; 8 - komplet)	int
4	tw_Symbol	Symbol towaru	varchar (20)
5	tw_Nazwa	Nazwa towaru	varchar (50)
6	tw_Opis	Opis	varchar (255)
7	tw_IdVatSp	Stawka VAT przy sprzedaży	int
8	tw_IdGrupa	Grupa	int
9	tw_WWW	Adres WWW	varchar (255)
10	tw_SklepInternet	Czy pozycja przeznaczona do sklepu internetowego	bit
11	tw_Pole1	Własne pole 1	varchar (50)

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

12	tw_Uwagi	Uwagi	varchar (255)
13	tw_Charakter	Pełna charakterystyka towaru	text

Tabela 4.7 tw_Cena - Ceny sprzedaży

L.p.	Nazwa	Opis	Typ
1	tc_Id	Identyfikator	int
2	tc_IdTowar	Identyfikator towaru którego dotyczy cena	int
3	tc_CenaNetto1	Cena sprzedaży netto 1	money
4	tc_CenaNetto2	Cena sprzedaży netto 2	money
5	tc_CenaBrutto1	Cena sprzedaży brutto 1	money
6	tc_CenaBrutto2	Cena sprzedaży brutto 2	money

4.3.2. Schemat bazy danych sklepu osCommerce

Cały schemat bazy danych sklepu znajduje się na dołączonej płycie CD w pliku tep_database-pr2.2-CVS.pdf w katalogu w plikami instalacyjnymi sklepu osCommerce. W zamieszczonych relacjach zostały pominięte atrybuty nie używane podczas synchronizacji. Atrybuty, które zostały dodane do bazy danych by móc przechowywać informacje nie przewidziane w sklepie osCommerce, ale przewidziane w Subieckie GT i naszej funkcjonalności zostały wypisane czcionką pogrubioną.

Tabela 4.8 geo_zones - Grupy stref podatkowych

L.p.	Nazwa	Opis	Typ
1	geo_zone_id	Identyfikator strefy	Integer (5)
2	geo_zone_name	Nazwa strefy	Varchar (32)
3	geo_zone_description	Opis strefy podatkowej	Varchar (255)

Tabela 4.9 tax_class - Grupa podatkowa

L.p.	Nazwa	Opis	Typ
1	tax_class_id	Identyfikator grupy podatkowej	Integer (5)
2	tax_class_title	Nazwa grupy podatkowej	Varchar (32)
3	tax_class_description	Opis grupy podatkowej	Varchar (255)

Tabela 4.10 tax_rates - Stawka podatkowa

L.p.	Nazwa	Opis	Typ
1	tax_rates_id	Identyfikator stawki podatkowej	Integer (5)
2	tax_zone_id	Identyfikator strefy podatkowej	Integer (5)
3	tax_class_id	Identyfikator grupy podatkowej	Integer (5)
4	tax_rate	Wartość podatku	Decimal (9,4)
5	tax_description	Opis podatku	Varchar (255)

Tabela 4.11 categories - Kategorie

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	categories_id	Identyfikator kategorii	Integer (5)
2	parent_id	Identyfikator kategorii nadrzędnej	Integer (5)

Tabela 4.12 categories_description - Opis kategorii

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	categories_id	Identyfikator kategorii	Integer (5)
2	language_id	Numer języka	Integer (5)
3	categories_name	Nazwa kategorii	Varchar (32)

Tabela 4.13 languages - Języki

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	languages_id	Identyfikator języka	Integer (5)
2	name	Nazwa języka	Varchar (32)
3	code	Kod	Char (2)

Tabela 4.14 products_options - Rodzaje opcji

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	products_options_id	Identyfikator opcji	Integer (5)
2	language_id	Identyfikator języka	Integer (5)
3	products_options_name	Nazwa opcji	Varchar (32)

Tabela 4.15 products_options_values - Wartości opcji

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	products_options_values_id	Identyfikator wartości opcji	Integer (5)
2	languages_id	Identyfikator języka	Integer (5)
3	products_options_values_name	Nazwa wartości opcji	Varchar (64)

Tabela 4.16 productes_attributes – Typy produktów

<i>L.p.</i>	<i>Nazwa</i>	<i>Opis</i>	Typ
1	products_attributes_id	Identyfikator typu	Integer (5)
2	products_id	Identyfikator produktu	Integer (5)
3	options_id	Identyfikator opcji	Integer (5)
4	options_values_id	Identyfikator wartości opcji	Integer (5)
5	options_values_price	Cena typu	Decimal (15,4)
6	options_quantity	Dostępna ilość sztuk typu	Integer (6)
7	wholesale_price	Hurtowa cena netto	Decimal (15,4)
8	options_values_description	Opis typu	Varchar(255)

Tabela 4.17 products_to_categories - Relacja złączenia produktów i kategorii

L.p.	Nazwa	Opis	Typ
1	products_id	Identyfikator produktu	Integer (5)
2	categories_id	Identyfikator kategorii	Integer (5)

Tabela 4.18 products - Produkty

L.p.	Nazwa	Opis	Typ
1	products_id	Identyfikator produktu	Integer (5)
2	products_quantity	Ilość produktu	Integer (4)
3	products_model	Typ produktu	Varchar (12)
4	products_price	Cena netto produktu	Decimal (15,4)
5	products_tax_class_id	Identyfikator grupy podatku	Integer (5)
6	wholesale_price	Cena hurtowa produktu	Decimal (15,4)

Tabela 4.19 products_description - Opis produktów

L.p.	Nazwa	Opis	Typ
1	Products_id	Identyfikator produktu	Integer (5)
2	Language_id	Identyfikator języka	Integer (5)
3	Products_name	Nazwa produktu	Varchar (64)
4	Products_description	Opis produktu	Text
5	Products_url	Adres do linka dodatkowo opisującego produkt	Varchar (255)

4.4. Algorytm synchronizacji produktów

Algorytm synchronizacji produktów jest algorytmem programu synchronizującego bazę danych Subiekta GT i sklepu internetowego osCommerce. Jest on zgodny z założeniami punktu 4.1 i sposobem synchronizacji omówionym w pierwszym akapicie rozdziału 4.2.

Stosowane konwencje i uogólnienia:

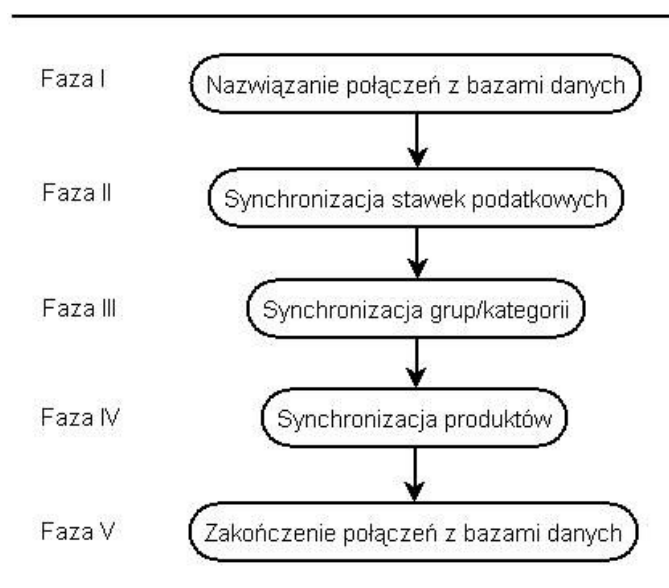
- Podczas opisu działania algorytmu pomijamy sposób implementacji przechowywania danych w języku implementującym algorytm. W zależności od programisty implementującego algorytm może to być klasa, struktura, tablica asocjacyjna, itp.
- Instrukcje SQL realizujące opisywane operacje w bazie danych będą wyróżnione czcionką bez szeryfowymi (Przykładowa instrukcja SQL). Zwracane i używane wartości w zapytaniach będą podkreślone (np. zwrócona wartość).
- Bazę danych Subiekta GT (MS SQL) będziemy nazywać bazą źródłową i oznaczać ją SRC_DB.
- Bazę danych sklepu osCommerce (MySQL) będziemy nazywać bazą docelową i oznaczać ją DST_DB.

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

- Operację konwersji kodowania znaków z bazy SRC_DB do bazy DST_DB oznaczać będziemy jako funkcję `konwersja_kodowania(zmienne_poddane_konwersji)`. Domyślnym stroną kodowania napisów przechowywanych w bazie danych programu Subiekt GT jest polska strona kodowania Windows (windows-1250). Natomiast w przypadku sklepu osCommerce zależy to od stosowanego języka. Ponieważ synchronizowany sklep będzie obsługiwał domyślnie polski język dlatego będziemy stosować polską stronę kodowania ISO (ISO-8859-2) i do niej konwertować dane.

Algorytm synchronizacji produktów przedstawionych na rysunku (4.2) można podzielić na pięć głównych faz:

- I. Nawiązanie połączeń z bazami danych – Grupuje wszystkie operacje zajmujące się nawiązaniem połączenia z bazą danych, sprawdzaniem połączenia i obsługi transakcji. Niepowodzenie, którejkolwiek z operacji przerywa synchronizację.
- II. Synchronizacja stawek podatkowych – Sprawdza, aktualizuje i dodaje stawki podatkowe produktów. Wszystkie operacje w tej fazie na bazie DST_DB są traktowane jako jedna transakcja. Niepowodzenie tej fazy wycofuje transakcję i przerywa synchronizację ponieważ może to doprowadzić do sytuacji gdy produkt nie będzie miał przypisanej stawki podatkowej, stawka będzie zła, błędnych operacji dodawania produktu, błędnych operacji księgowania zamówień, błędnej ceny wyświetlonej w sklepie.
- III. Synchronizacja grup/kategorii – Sprawdza aktualizuje i dodaje grupy/kategorie produktów. Wszystkie operacje w tej fazie na bazie DST_DB są traktowane jako jedna transakcja. Niepowodzenie tej fazy wycofuje transakcję i przerywa synchronizację ponieważ może to doprowadzić do sytuacji, w której produkt nie będzie miał przypisanej kategorii. Co za tym idzie nie będzie widziany w sklepie.
- IV. Synchronizacja produktów – Sprawdza, aktualizuje, dodaje i usuwa produkty. Wszystkie operacje dotyczące jednego produktu są uważane za jedną transakcję. W przypadku niepowodzenia operacji na produkcie wycofuje wykonane na nim operacje i przechodzi do następnego produktu. W przypadku gdy baza DST_DB nie obsługuje transakcji przerywa synchronizację.
- V. Zakończenie połączeń z bazami danych – Zatwierdza transakcję i zamyka połączenie z bazami danych.



Rys. 4.2. Główne fazy algorytmu

4.4.1. Nawiązanie połączeń z bazami danych

- I.1. Ustanowić połączenie z bazą SRC_DB
- I.2. Ustanowić połączenie z bazą DST_DB
- I.3. Sprawdzić czy baza DST_DB obsługuje transakcje

4.4.2. Synchronizacja stawek podatkowych

- II.1. Pobrać stawki podatkowe z bazy SRC_DB

```
SELECT vat_Id, vat_Nazwa, vat_Stawka, vat_Symbol FROM sl_StawkaVAT;
```
- II.2. Przekonwertować kodowanie napisów wartości atrybutu vat_nazwa.

```
Konwersja_kodowania(vat_Nazwa);
```
- II.3. Pobrać identyfikator Europejskiej grupy strefy podatkowej z bazy DST_DB

```
SELECT geo_zone_id FROM geo_zones WHERE geo_zone_name = 'Europe';
```
- II.4. Jeżeli zapytanie zwróciło dane przejdź do pkt. II.6.
- II.5. Dodaj Europejską grupę strefy podatkowej i zapamiętaj identyfikator ostatnio wstawianej krotki.

```
INSERT INTO geo_zones (geo_zone_name, geo_zone_description, date_added)  
VALUES('Europe', '', NOW());
```
- II.6. Pobrać identyfikatory i nazwy grupy podatkowych z bazy DST_DB

```
SELECT tax_class_id, tax_class_title FROM tax_class;
```
- II.7. Porównać pobrane nazwy grup podatkowych z bazy SRC_DB z grupami z bazy DST_DB
- II.8. Jeżeli są wszystkie grupy podatkowe to przejdź do pkt. II.10.
- II.9. Dodaj brakującą grupę do bazy DST_DB

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

vat Nazwa - pkt. II.1

```
INSERT INTO tax_class (tax_class_title, tax_class_description,  
date_added) VALUES('vat Nazwa', '', NOW());
```

II.10. Pobrać stawki podatkowe z bazy DST_DB.

geo zone id - pkt. II.3

```
SELECT tax_description, tax_rates_id, tax_class_id, tax_rate FROM  
tax_rates WHERE tax_zone_id='geo zone id ';
```

II.11. Porównać pobrane stawki podatkowe z bazy SRC_DB ze stawkami z bazy DST_DB

II.12. Jeżeli są wszystkie stawki podatkowe to przejdź do punktu II.14.

II.13. Dodaj brakującą stawkę podatkową do bazy DST_DB

geo zone id - pkt. II.3

tax class id - pkt. II.6

vat Stawka - pkt. II.1

vat Nazwa - pkt. II.1

```
INSERT INTO tax_rates (tax_zone_id, tax_class_id, tax_priority,  
tax_rate, tax_description, date_added) VALUES('geo zone id', '  
tax class id', 0, 'vat Stawka', 'vat Nazwa', NOW());
```

II.14. Jeżeli nie zmieniły się stawki podatkowe to przejdź do punktu II.16.

II.15. Uaktualnij zmienione wartości stawki podatkowej

tax rate - pkt. II.10

tax rates id - pkt. II.10

```
UPDATE tax_rates SET tax_rate='tax rate', last_modified=NOW() WHERE  
tax_rates_id='tax rates id';
```

4.4.3. Synchronizacja grup/kategorii

III.1. Pobrać kod i identyfikator języka polskiego z bazy DST_DB

```
SELECT languages_id, code FROM languages WHERE code LIKE 'PL'
```

III.2. Pobrać kategorie produktów z DST_DB

languages id - pkt. III.1

```
SELECT categories.categories_id, parent_id, categories_name FROM  
categories, categories_description WHERE categories.categories_id =  
categories_description.categories_id AND  
categories_description.language_id = languages id;
```

III.3. Utworzyć drzewo kategorii sklepu

III.4. Pobrać z bazy DST_DB grupy produktów przeznaczonych do sklepu

```
SELECT DISTINCT sl_GrupaTw.grt_Id, sl_GrupaTw.grt_Nazwa FROM sl_GrupaTw  
INNER JOIN tw__Towar ON sl_GrupaTw.grt_Id = tw__Towar.tw_IdGrupa WHERE  
(tw__Towar.tw_SklepInternet = 1) AND (tw__Towar.tw_Zablokowany = 0)  
ORDER BY grt_Id;
```

III.5. Przekonwertować kodowanie napisów wartości atrybutu grt_Nazwa.

```
Konwersja_kodowania(grt_Nazwa);
```

III.6. Podzielić grt_Nazwa na kategorię nadrzędną i podrzędną rozdzieloną znakiem specjalnym.

III.7. Stworzyć pierwszy poziom drzewa kategorii Subiekta.

III.8. Jeżeli wszystkie z liści pierwszego poziomu drzewa kategorii z Subiekta występują w drzewie sklepu to przejdź do pkt. III.10.

III.9. Dodaj brakującą kategorię pierwszego poziomu w sklepie.

nazwa kategorii nadrzędnej - pkt. III.6

languages_id - pkt. III.1

```
(*) INSERT INTO categories (parent_id, date_added) VALUES (0, NOW());  
INSERT INTO categories_description (categories_id, language_id,  
categories_name) VALUES (identyfikator krotki wstawionej instrukcja (*),  
languages_id, 'nazwa kategorii nadrzędnej');
```

III.10. Stworzyć drugi poziom drzewa kategorii Subiekta.

III.11. Jeżeli wszystkie z liści drugiego poziomu drzewa kategorii z Subiekta występują w drzewie sklepu to przejdź do pkt. III.13.

III.12. Dodaj brakującą kategorię drugiego poziomu w sklepie.

nazwa kategorii podrzędnej - pkt. III.6

languages_id - pkt. III.1

identyfikator kategorii nadrzędnej - pkt. III.9

```
(*) INSERT INTO categories (parent_id, date_added) VALUES  
(identyfikator kategorii nadrzędnej, NOW());  
INSERT INTO categories_description (categories_id, language_id,  
categories_name) VALUES (identyfikator krotki wstawionej instrukcja (*),  
languages_id, 'nazwa kategorii podrzędnej');
```

III.13. Pobrać z bazy SRC_DB ostatnie podkategorie (3).

```
SELECT sl_CechaTw.ctw_Nazwa, tw__Towar.tw_Id, tw__Towar.tw_Rodzaj,  
tw__Towar.tw_IdGrupa, sl_CechaTw.ctw_Id FROM tw__Towar INNER JOIN  
tw_CechaTw ON tw__Towar.tw_Id = tw_CechaTw.cht_IdTowar INNER JOIN  
sl_CechaTw ON tw_CechaTw.cht_IdCecha = sl_CechaTw.ctw_Id AND  
tw_CechaTw.cht_IdCecha = sl_CechaTw.ctw_Id WHERE  
(tw__Towar.tw_SklepInternet = 1) AND (tw__Towar.tw_Zablokowany = 0)  
ORDER BY tw__Towar.tw_IdGrupa;
```

III.14. Przekonwertować kodowanie napisów wartości atrybutu ctw_Nazwa.

```
Konwersja_kodowania(ctw_Nazwa);
```

III.15. Stworzyć trzeci poziom drzewa kategorii Subiekta.

III.16. Jeżeli wszystkie z liści trzeciego poziomu drzewa kategorii z Subiekta występują w drzewie sklepu to przejdź do pkt. III.19.

III.17. Dodaj brakującą kategorię trzeciego poziomu w sklepie.

nazwa ostatniego poziomu kategorii - pkt. III.14

languages_id - pkt. III.1

identyfikator kategorii nadrzędnej - pkt. III.12

```
(*) INSERT INTO categories (parent_id, date_added) VALUES  
(identyfikator kategorii nadrzędnej, NOW());  
INSERT INTO categories_description (categories_id, language_id,  
categories_name) VALUES (identyfikator krotki wstawionej instrukcja (*),  
languages_id, 'nazwa ostatniego poziomu kategorii');
```

4.4.4. Synchronizacja produktów

IV.1. Pobrać produkty z bazy SRC_DB

```
SELECT tw__Towar.tw_Id, tw__Towar.tw_Symbol, tw__Towar.tw_Nazwa,  
tw__Towar.tw_Opis, tw_Cena.tc_CenaNetto1, tw_Cena.tc_CenaNetto2,  
tw_Cena.tc_CenaBrutto1, tw_Cena.tc_CenaBrutto2, tw__Towar.tw_IdGrupa,  
tw__Towar.tw_WWW, tw__Towar.tw_Pole1, tw__Towar.tw_Uwagi,  
tw__Towar.tw_Charakter, tw__Towar.tw_IdVatSp FROM tw__Towar INNER JOIN  
tw_Cena ON tw__Towar.tw_Id = tw_Cena.tc_IdTowar WHERE  
(tw__Towar.tw_SklepInternet = 1) AND (tw_Zablokowany=0);
```

IV.2. Przekonwertować kodowanie napisów wartości atrybutu tw_Charakter,

tw_Uwagi, tw_Opis, tw_IdGrupa z bazy SRC_DB.

```
Konwersja_kodowania(tw_Charakter, tw_Uwagi, tw_Opis, tw_IdGrupa);
```

IV.3. Jeżeli wersja Subiekta jest starsza niż 1.10 przejdź do pkt. IV. 4.

IV.4. Przekonwertuj opis z atrybutu tw_Charakter z rtf-a do na czysty tekst i przejdź do pkt. IV.5.

IV.5. Przekonwertuj opis z atrybutu tw_Charakter z html-a do na czysty tekst.

IV.6. Dla każdego pobranego produktu w punkcie IV1 pobierz stan magazynowy z bazy DST_DB.

tw_Id - pkt. IV.1

```
SELECT c.st_Stan, c.st_StanRez FROM tw_Stan c WHERE (st_TowId = tw_Id)  
AND (st_magId='1');
```

IV.7. Stwórz drzewo produktów Subiekta w którym pierwszy poziom liści identyfikuje główny produkt, a drugi poziom liści zawiera typy produktu.

IV.8. Znajdź najniższą cenę typu każdego produktu i przypisz ją produktowi.

IV.9. Znajdź pierwsze niepuste wystąpienie (w kolejności alfabetycznej): nazwy (tw_Uwagi), opisu (tw_charakter), adresu WWW (tw_WWW) i przypisz je produktowi.

IV.10. Zsumuj stan magazynowy typów i zapisz go w produkcie.

IV.11. Pobranie z bazy DST_DB nazwy opcji dla języka polskiego.

```
languages_id - pkt. III.1  
SELECT products_options_name, products_options_id FROM products_options  
WHERE language_id = languages_id;
```

IV.12. Pobranie relacji kategorii i produktów (przypisanych produktów do kategorii).

```
SELECT * FROM products_to_categories;
```

IV.13. Pobranie produktów z bazy DST_DB

```
languages_id - pkt. III.1  
SELECT * FROM products, products_description WHERE  
products.products_id=products_description.products_id AND  
products_description.language_id = languages_id
```

IV.14. Stwórz drzewo produktów osCommerce w którym pierwszy poziom liści identyfikuje główny produkt, a drugi poziom liści zawiera typy produktu.

IV.15. Porównaj drzewa produktów SRC_DB i DST_DB.

IV.16. Jeżeli żaden z produktów lub jego typów w bazie DST_DB nie wymaga aktualizacji przejdź do pkt. IV.19.

IV.17. Jeżeli dane produktu w bazie DST_DB nie wymagają aktualizacji to przejdź do pkt.

IV18.

```
Ilość - pkt. IV.10  
CenaDetalicznaNetto - pkt. IV.8  
CenaHurtowaNetto - pkt. IV.8  
GrupaVat - pkt. IV.7  
products_id - pkt. IV.14  
tw_Uwagi - pkt. IV.7  
languages_id - pkt. III.1  
tw_Uwagi - pkt. IV.9  
tw_WWW - pkt. IV.9  
IdGrupa - IV.14  
UPDATE products SET products_quantity = 'Ilość',  
products_price='CenaDetalicznaNetto',  
wholesale_price='CenaHurtowaNetto', products_tax_class_id='GrupaVat',  
products_last_modified=NOW() WHERE products_id='products_id';
```

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

```
UPDATE products_description SET products_name='tw Uwagi',
products_description='tw Uwagi', products_url='tw WWW' HERE language_id
= languages_id AND products_id=products_id";
UPDATE products_to_categories SET categories_id='IdGrupa' WHERE
products_id='products_id';
```

IV.18. Jeżeli dane typu produktu w bazie DST_DB nie wymagają aktualizacji przejdź do pkt.

IV 19.

```
oprions values price - pkt. IV 7
options quantity - pkt. IV 7
wholesale price - pkt. IV 7
options values description - pkt. IV 7
products id - pkt. IV14
options id - pkt. IV. 14
options values id - pkt. IV. 14
UPDATE products_attributes SET
options_values_price='oprions values price', price_prefix='',
options_quantity='options quantity', wholesale_price='wholesale price',
options_values_description='options values description' WHERE
products_id='products id' AND options_id='options id' AND
options_values_id='options values id'
```

IV.19. Jeżeli nie brakuje żadnego typu w bazie DST_DB przejdź do pkt. IV.20

```
languages id - pkt. III.1
typ produktu - pkt. IV7
SELECT products options values id FROM products_options_values WHERE
language_id = $nr_jezyka_pl ORDER BY products_options_values_id DESC
LIMIT 1;
kolejny wolny numer = products options values id +1
INSERT INTO products_options_values (products_options_values_id,
language_id, products_options_values_name) VALUES('kolejny wolny numer',
'languages id', 'typ produktu';
Dla pozostałych języków.
INSERT INTO products_options_values (products_options_values_id,
language_id, products_options_values_name) VALUES('kolejny wolny numer',
'identyfikator innego języka', '');
```

IV.20. Jeżeli nie brakuje żadnego produktu w bazie DST_DB przejdź do pkt. V 21

```
Symbol - pkt. IV7
Ilość - pkt. IV10
CenaDetalicznaNetto - pkt. IV8
CenaHurtowaNetto - pkt. IV8
GrupaVat - pkt. IV7
```

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

products_id - pkt. IV14

tw Uwagi - pkt. IV7

languages_id - pkt. III.1

tw Uwagi - pkt. IV9

tw WWW - pkt. IV9

IdGrupa - IV14

```
(*) INSERT INTO products (products_quantity, products_model,
products_price, wholesale_price, products_weight, products_status,
products_tax_class_id, manufacturers_id, products_date_added)
VALUES('Ilość', 'Symbol', 'CenaDetalicznaNetto', 'CenaHurtowaNetto', 0,
1, 'GrupaVat', 0, NOW());
```

```
INSERT INTO products_to_categories (products_id, categories_id)
VALUES('identyfikator krotki wstawionej instrukcja (*)', 'IdGrupa');
INSERT INTO products_description (products_id, products_name,
products_description, products_url, language_id)
VALUES('identyfikator krotki wstawionej instrukcja (*)', 'tw Uwagi',
'tw Uwagi', 'tw WWW', 'languages_id');
```

Dla pozostałych języków.

```
INSERT INTO products_description (products_id, products_name,
products_description, products_url, language_id)
VALUES('identyfikator krotki wstawionej instrukcja (*)', '', '', '',
'identyfikator innego języka');
```

IV.21. Jeżeli są produkty DST_DB, których nie ma w bazie SRC_DB to je usuń.

products_id - pkt. IV14

```
DELETE FROM products WHERE products_id=products_id;
DELETE FROM products_attributes WHERE products_id=products_id;
DELETE FROM products_description WHERE products_id=products_id;
DELETE FROM products_to_categories WHERE products_id=products_id;
```

4.4.5. Zakończenie połączeń z bazami danych

- V.1. Zatwierdzić transakcję DST_DB.
- V.2. Zamknąć połączenie z bazą SRC_DB.
- V.3. Zamknąć połączenie z bazą DST_DB.
- V.4. Usunąć zbędne zmienne.

5. Implementacja algorytmu

5.1. Uzasadnienie wyboru środowiska

Środowisko w którym będę implementować algorytm synchronizujący produkty to PHP [37]. Mój wybór można uzasadniać następującymi czynnikami:

- Środowisko programistyczne i uruchomieniowe dostępne jest na większości systemów operacyjnych,
- Jednolita platforma programistyczna sklepu i skryptu,
- Stabilne i dojrzałe środowisko,
- Otwarta licencja [18] i wolna zarazem,
- Dobra dokumentacja.

Źródła i pliki binarne PHP można pobrać ze strony projektu [37]. Skrypt był testowany i wdrożony na systemie operacyjnym Windows XP. Podczas pracy wykorzystywane było środowisko programistyczne oparte na Eclipse [40] z rozszerzeniami. Kod skryptu jest udokumentowany zgodnie ze składnią phpDocumentor [41]. Instalację poprawek do sklepu wykorzystujących zmodyfikowany kod bazy należy wykonać samemu uprzednio modyfikując strukturę bazy danych. Instrukcje SQL modyfikujące bazę danych zostały zmieszczone na płycie CD.

5.2. Implementacja

Implementując algorytm z rozdziału 4 w PHP istniały trzy możliwości napisania kodu: kod proceduralny, kod obiektowy, kod proceduralno-obiektowy. Każda z tych możliwości ma swoje wady i zalety. Podczas implementacji użyłem kodu proceduralno-obiektowego ponieważ pozwoliło mi to przejrzeć i napisać cały kod. Skrypt udostępniam na wolnej licencji GPL.

W stosunku do algorytmu z rozdziału 4 podczas implementacji została zmieniona następująca funkcjonalność skryptu:

1. Dodano raport końcowy podsumowujący synchronizację,
2. Zrezygnowano z transakcji ponieważ baza danych MySQL przy silniku bazodanowym ustawionym na MyISAM nie obsługuje transakcji. Emulacja transakcji ze względów wydajnościowych została zaniechana,
3. Dodano logowanie przebiegu transakcji.

Skrypt został podzielony na 11 plików:

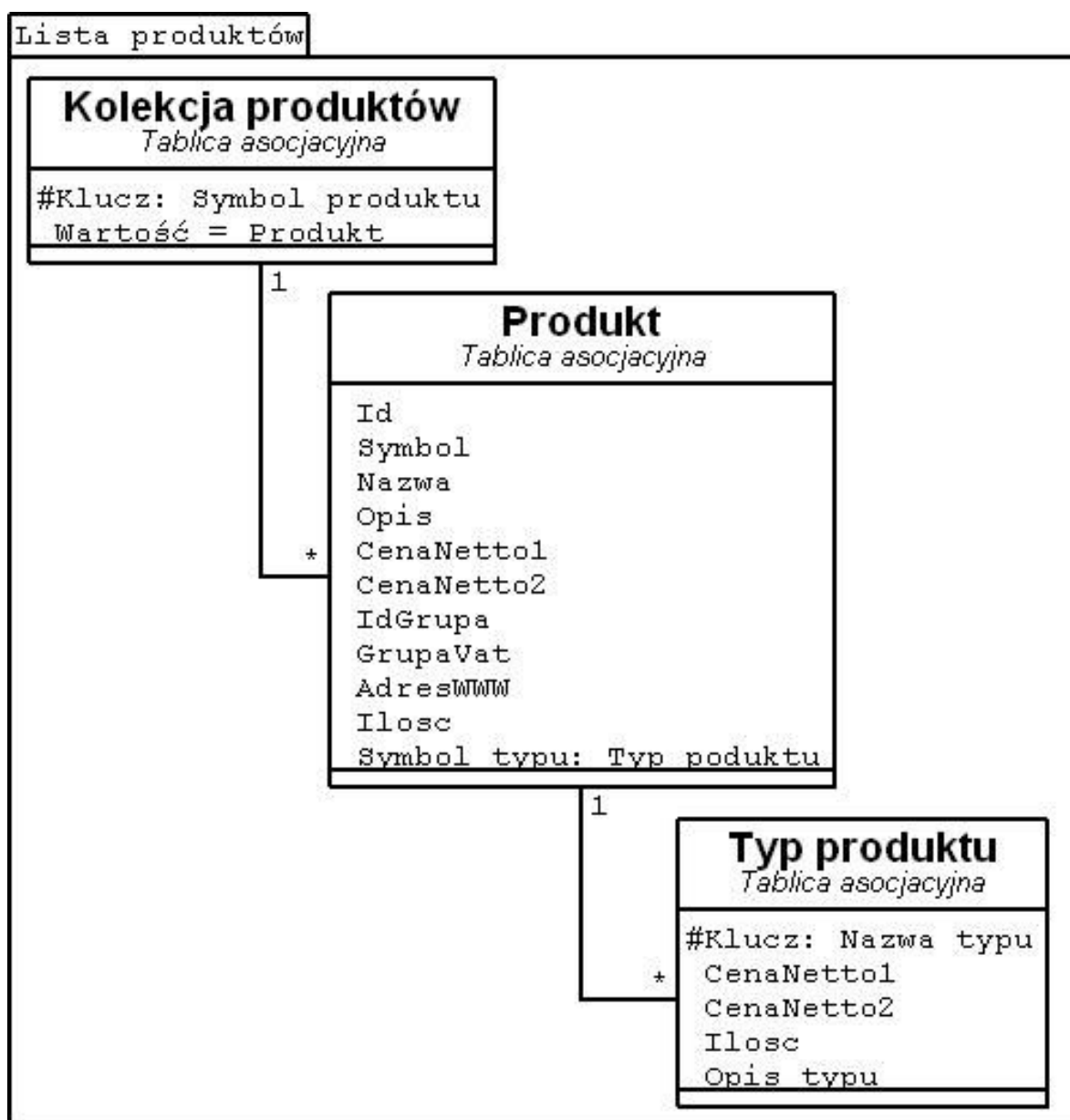
1. config.php
2. functions/function.php
3. class/databases.php
4. class/mssql.php
5. class/mysql.php
6. synchronizacja_podatkow.php
7. synchronizacja_grup.php
8. drzewa_produkow.php
9. synchronizacja_produkow.php
10. kasowanie.php
11. raport.php

Każdy z plików jest odpowiednio skomentowany dlatego nie będę cytował fragmentów kodu.

Operację na bazie danych wykonuje za pomocą metod własnej klasy db, której instancje obsługują połączenia z bazami danych MySQL i MS SQL. Ze względu na złączenia i czytelność kodu zrezygnowałem z tworzenia tablic zmiennych przechowujących dane przesyłane pomiędzy bazą danych, a metodami. Zamiast tego przesyłam do metod całe instrukcje SQL.

Konwersje kodowania znaków wykonuję wykorzystując iconv oddzielnie danych do bazy danych i na standardowe wyjście. Czyszczenie opisów produktów ze standardu rtf (wersja wcześniejsza niż 1.10) wykonuję przy pomocy zamiany znaków w zmiennej, a dla standardu HTML przy pomocy funkcji strip_tags.

Produkty przechowuję w wielowymiarowej tablicy asocjacyjnej, w której klucz tablicy jest symbolem produktu, a wartości kolejną tablicą z danymi produktu. Typy produktów przechowuję w tablicy z danymi produktu, w której nazwa zmiennej typu jest kluczem nowej tablicy przechowującej dane typu produktu. Sytuację tą obrazuje rysunek 5.1.



Rys. 5.1. Diagram wielowymiarowej tablicy asocjacyjnej przechowującej produkty i typy

5.3. Testy wydajności

Testy wydajnościowe skryptu prowadzone były na testowej bazie danych wzorowanej na strukturze bazy danych firmy Cyber Tattoo Factory [42] w Krakowie. Firma ta chętnie uczestniczyła w pracach rozwojowych i testowych programu. Za to dziękuję Panu Tomaszowi Rachwałowi.

Testy wydajnościowe zostały przeprowadzone na komputerze o następujących parametrach:

Producent: ASUS
Rodzaj: Laptop

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Typ:	A6Km
Procesor:	AMD Turion 64 Mobile Technology ML-37, 2011MHz
Ram:	1024 MB, DDR 333
System operacyjny:	Windows XP Home Edition
Dysk twardy:	Hitachi 80GB, średni czas dostępu 12ms, 5400 obr/min
Baza danych Subiekta:	MS SQL, wersja 8.00.760
Łącze internetowe:	DSL 4MB (512MB/4096MB)

W testach będą wykorzystywane dwie bazy danych sklepu:

Lokalna (oznaczana L):	MySQL, wersja 5.0.27
Zdalna (oznaczana Z):	MySQL, wersja 4.1.21

Założenia pomiarowe:

- wszystkie czasy podawane są w sekundach z dokładnością do 1ms,
- czas wykonywania zapytań nie uwzględnia czasu potrzebnego na przekazanie danych z funkcji obsługującej zapytanie do kodu wywołującego,
- sklep obsługuje cztery języki,
- losowość opóźnień zdalnej i lokalnej bazy danych nie wpływa znacząco na poziom oszacowanej wydajności.

Test porównawczy nr 1:

Pierwszy test porównawczy ma za zadanie określić różnicę w czasie zsynchronizowania identycznych (pod względem danych) baz danych Subiekta i sklepu oraz określenie stałej minimalnej ilości zapytań do baz.

Warunki początkowe:

- Bazy są zsynchronizowane:	tak
- Ilość produktów w Subiekcie przeznaczonych do sklepu internetowego:	3949
- Ilość produktów w sklepie:	716
- Ilość typów produktów w sklepie:	3646
- Ilość kategorii i podkategorii:	62
- Ilość klas i stawek podatkowych:	12

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

Czasy zapytań do bazy Subiekta	L	L
Czas całkowity 3953 zapytań typu Select:	0,954202	1,127718
Czas całkowity 3953 zapytań:	0,954202	1,127718
Czasy zapytań do bazy sklepu	L	Z
Czas całkowity 9 zapytań typu Select:	0,115913	2,619526
Czas całkowity 9 zapytań:	0,115913	2,619526
Podsumowanie	L	Z
Czas wykonywania kodu:	1,233415	1,790384
Czas wykonywania zapytań do bazy danych:	1,070115	3,747244
Czas wykonywania skryptu:	2,303530	5,537628

Wnioski:

Czas wykonywania kodu mojego algorytmu w PHP mieści się w granicach 1 – 1,5s. Przy czym czas wykonywania kodu dla zdalnej bazy danych jest średnio o dłuższy 0,75s. W przypadku większej ilości produktów niż ~4 tys czas wykonania kod wydłuża się średnio o 1s na każde 4 tysiące produktów.

Czas wykonywania zapytań do bazy danych jest najbardziej znaczący przy synchronizacji i w dużej mierze zależy od szybkości połączenia sieciowego, wydajności bazy danych i jej obciążenia. Minimalna stała ilość zapytań do bazy danych Subiekta bez względu na ilość produktów jest równa 4. Czas ich wykonywania jest mniejszy niż 0,5s. Pozostałe zapytania do bazy źródłowej odnoszą się do trzeciego poziomu kategorii, który jest zapisany w każdym produkcie. W przypadku rezygnacji z trzeciego poziomu zagłębienia kategorii lub innego jej zapisywania czas zapytań do bazy Subiekta można by ograniczyć do mniej niż 0,5s. Minimalna stała ilość zapytań do bazy danych sklepu bez względu na ilość produktów i ich typów jest równa 9. W przypadku lokalnej bazy danych sklepu czas wykonania tych zapytań jest mniejszy niż 200ms. Natomiast zdalna baza danych udostępniona przez firmę hostingową potrzebuje na to 2,5 – 3,5 s.

Test porównawczy nr 2:

Drugi test porównawczy ma określić czas potrzeby na synchronizację baz danych przy docelowym użytkowaniu skryptu. Symuluje on wykonanie modyfikacji dwóch produktów i trzech typów po której użytkownik zainicjował synchronizację w celu odwzorowania zamian w sklepie internetowym.

Warunki początkowe:

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

- Bazy są zsynchronizowane:	nie
- Ilość produktów wymagających aktualizacji:	2
- Ilość typów wymagających aktualizacji:	3
- Ilość produktów w Subiekcie przeznaczonych do sklepu internetowego:	3949
- Ilość produktów w sklepie:	716
- Ilość typów produktów w sklepie:	3646
- Ilość kategorii i podkategorii:	62
- Ilość klas i stawek podatkowych:	12

Czasy zapytań do bazy Subiekta	L	L
Czas całkowity 3953 zapytań typu Select:	1,176936	1,172745
Czas całkowity 3953 zapytań:	1,176936	1,172745
Czasy zapytań do bazy sklepu	L	Z
Czas całkowity 9 zapytań typu Select:	0,127382	3,253001
Czas całkowity 15 zapytań typu Update:	0,003563	1,834411
Czas całkowity 24 zapytań:	0,130945	5,087413
Podsumowanie	L	Z
Czas wykonywania kodu:	1,001033	1,654768
Czas wykonywania zapytań do bazy danych:	1,307881	6,260158
Czas wykonywania skryptu:	2,308914	7,914926

Wnioski:

Czas wykonywania programu dla lokalnej bazy danych wydłużył się średnio o 10ms, a dla zdalnej bazy danych średnio o 2s w czasie których zostało wykonane 15 aktualizacji krotek w bazie danych sklepu. Na czas wydłużonej aktualizacji i dodawania krotek w bazie danych sklepu wpływają indeksy założone na atrybutach relacji. Dodatkowo decydujący wpływ na tak niską wydajność mają czynniki wymienione we wniosku z testu porównawczego nr 1.

Test porównawczy nr 3:

Trzeci test porównawczy symuluje pierwszą synchronizację baz danych.

Warunki początkowe:

- Bazy są zsynchronizowane:	nie
- Ilość produktów wymagających dodania:	716

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

- Ilość typów wymagających dodania:	3646
- Ilość produktów w Subieckie przeznaczonych do sklepu internetowego:	3949
- Ilość produktów w sklepie:	0
- Ilość typów produktów w sklepie:	0
- Ilość kategorii i podkategorii:	62
- Ilość klas i stawek podatkowych:	12

<u>Czasy zapytań do bazy Subiekta</u>	<u>L</u>	<u>L</u>
Czas całkowity 3953 zapytań typu Select:	0,797667	3,041177
Czas całkowity 3953 zapytań:	0,797667	3,041177
<u>Czasy zapytań do bazy sklepu</u>	<u>L</u>	<u>Z</u>
Czas całkowity 497 zapytań typu Select:	0,087592	50,297299
Czas całkowity 10208 zapytań typu Insert:	2,030953	1018,883791
Czas całkowity 1 zapytań typu Query:	0,000162	0,094066
Czas całkowity 10706 zapytań:	2,118706	1069,275156
<u>Podsumowanie</u>	<u>L</u>	<u>Z</u>
Czas wykonywania kodu:	1,400875	2,352264
Czas wykonywania zapytań do bazy danych:	2,916373	1072,316333
Czas wykonywania skryptu:	4,317248	1074,668597

Wnioski:

Czas synchronizacji lokalnej bazy danych zwiększył się nie znacznie w porównaniu do zdalnej bazy danych, która w przybliżeniu synchronizowała się 18 minut!!!. W celu skrócenia czasu synchronizacji bez zmiany firmy hostingowej można by wykonać następujące modyfikacje algorytmu i sklepu:

- Usunięcie obsługi wielojęzycznych nazw typów. Zaletą jest mniejsza ilość zapytań aktualizacji, wstawiania i usuwania (jedno zamiast czterech). Zwiększona szybkość sklepu (mniejsza ilość krotek w relacji i brak wyszukiwania języka dla krotki). Wadą jest brak kompatybilności z oficjalnym wydaniem sklepu.
- Dodawania wyzwalacza dodającego opisy produktów i typów. Zaletą jest zmniejszenie ilości wysyłanych zdalnie zapytań do bazy danych (jedno zamiast czterech). Wadą jest taka sama ilość zapytań wykonywanych przez wyzwalacza lokalnie (dla wyzwalacza) na zdalnej bazie danych.
- Synchronizacja tylko jednego języka (polskiego). W przypadku braku opisu w innych językach niż domyślny dodać wyświetlanie opisu w języku domyślnym. Zaletą jest mniejsza

ilość zapytań aktualizacji, wstawiania i usuwania (jedno zamiast czterech). Wadą jest mniejsza wydajność sklepu internetowego (zwiększona ilość zapytań do bazy danych).

Test wydajności zdalnej bazy danych nr 1:

Test wydajności zdalnej bazy danych ma za zadanie określić czasochłonność operacji wstawiania, aktualizacji i usuwania. W celu przeprowadzenia testu zawartość bazy danych sklepu została odtworzona z kopii stworzonej dwa miesiące wcześniej, a następnie wykonano synchronizację danych.

Warunki początkowe:

- Bazy są zsynchronizowane:	nie
- Ilość produktów i typów wymagających dodania:	nieznana
- Ilość produktów i typów wymagających aktualizacji:	nieznana
- Ilość produktów i typów wymagających usunięcia:	nieznana
- Ilość produktów w sklepie po synchronizacji:	716
- Ilość typów produktów w sklepie po synchronizacji:	3646
- Ilość produktów w Subiekcie przeznaczonych do sklepu internetowego:	3949
- Ilość kategorii i podkategorii:	62
- Ilość klas i stawek podatkowych:	12

Czasy zapytań do bazy Subiekta	L	zap./sec
Czas całkowity 3953 zapytań typu Select:	0,792724	4986,6
Czas całkowity 3953 zapytań:	0,792724	4986,6
Czasy zapytań do bazy sklepu	Z	zap./sec
Czas całkowity 9 zapytań typu Select:	3,471240	2,6
Czas całkowity 285 zapytań typu Insert:	35,885232	7,9
Czas całkowity 2582 zapytań typu Update:	446,473333	5,78
Czas całkowity 192 zapytań typu Delete:	27,839685	6,89
Czas całkowity 3068 zapytań:	513,669490	5,98
Podsumowanie	Z	
Czas wykonywania kodu:	2,430077	
Czas wykonywania zapytań do bazy danych:	514,462214	
Czas wykonywania skryptu:	516,892291	

Wnioski:

Najbardziej czasochłonnym typem operacji modyfikujących bazę danych jest aktualizacja, której wydajność można przyjąć na poziomie 5,8 zapytania/sekundę. Przyczyną tego stanu są indeksy założone na atrybutach typu znakowego, które przyspieszają wykonywanie skryptów sklepu internetowego wykorzystujących intensywnie bazę danych. Wydawać by się mogło że usuwanie wierszy powinno być szybsze od wstawiania nowych wierszy jednak w naszym teście wydajność usuwania jest równa około 6,9 zapytania/sekundę. Zaskakujący może wydawać się fakt że wstawianie nowych krotek okazało się najwydajniejsze pomimo tego że, teoretycznie powinno być najwolniejsze. Wydajność wstawiania do bazy danych szacunkowo można określić na 8 zapytania/sekundę uwzględniając oprócz testu wydajności nr 1 (7,9 zap/sec.) test porównawczy nr. 3 (10 zap/sec).

Ogólne wnioski:

Przeprowadzone testy wykazały, że kod skryptu ma przeciętny czas wykonywania i można by go zoptymalizować. Przewidywany czas wykonywania zoptymalizowanego skryptu z wyłączeniem zapytań SQL powinien wykonywać się szybciej o około 1s. Przy zmianie założeń projektowych dotyczących kategorii i optymalizacji ilości oraz składni zapytań SQL przewidywany czas wykonywania wszystkich zapytań do bazy danych powinien skrócić się o minimum 10%. Dalsze skrócenie czasu wykonywania synchronizacji możliwe jest tylko na poziomie wykorzystywanej bazy danych i łącza sieciowego. W przypadku serwera dedykowanego z szybkim łączem internetowym pomiędzy bazami danych czas wykonywania synchronizacji powinien skrócić się o minimum 10%.

Podsumowanie

Wraz ze zwiększającą się ilością przechowywanych danych w bazach danych integracja tych baz stała się nieodłącznym elementem w rozbudowanych systemach informatycznych. Rozwiązując problem integracji baz danych należy przeprowadzić analizę istniejących systemów poddawanych integracji i efektów jakie mają być uzyskane. Metod realizacji jest kilkanaście i zależą one od przewidywanych efektów. W przypadku analizy danych są to hurtownie danych, a w przypadku rozproszonych systemów fragmentaryzujących dane mogą to być federacyjne bazy danych. Budowa narzędzi wspierających proces integracji danych jest na tyle złożona, że uniwersalne narzędzia nigdy nie powstaną. Dlatego programista lub administrator bazy danych przeprowadzający integrację baz danych musi się odznaczać dużą wiedzą o informacjach jakie są przechowywane w integrowanych źródłach danych i technicznych aspektach realizacji przedsięwzięcia. Wypracowanie kompromisu pomiędzy przewidywanymi efektami, a możliwościami realizacji może okazać się kluczowe przy realizacji projektu, tak jak to miało miejsce w przypadku integracji Subiekta GT i sklepu osCommerce. Bez odpowiednich założeń i uzgodnień z użytkownikami programu integracja by się nie powiodła, ponieważ przeniesienie wszystkich danych mogłoby się okazać nie zasadne i dodatkowo uczynić proces synchronizacji mało wydajnym, a sklep internetowy niepotrzebnie skomplikowanym. Realizacji synchronizacji produktów została zrealizowana zgodnie z algorytmem z wyłączeniem obsługi transakcji, co było konsekwencją kompromisu pomiędzy funkcjonalnością a wydajnością. Projekt skryptu został wdrożony w firmie Cyber Tattoo Factory w Krakowie, która była zainteresowana jego testowaniem, użytkowaniem i dalszym rozwijaniem programu.

Cel pracy został zrealizowany.

Bibliografia

1. Benon-Davies P.: Systemy baz danych. Wyd. WNT, Warszawa 1998
2. Garcia-Molina H. Ullman D. J. Widom J.: Systemy baz danych. Pełen wykład. Wyd. WNT, Warszawa 2006
3. Kukuczka J.: Relacyjne bazy danych. Wyd. Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 2000
4. C.J Date, Wprowadzenie do systemów baz danych., wydanie drugie całkowicie zmienione, Wyd. WNT, Warszawa, 2000
5. Augustyn D. Chłopek M.: Bazy danych w lokalnych sieciach komputerowych oraz systemach wielozadaniowych. Wyd. Politechniki Śląskiej, Gliwice 1994
6. Connolly T. Begg C.: Systemy baz danych. Wyd. RM, Warszawa 2004
7. Subiekta K.: Teoria i konstrukcja obiektowych języków zapytań. Wyd. Polsko-Japońska Wyższa Szkoła Technik Komputerowych, Warszawa 2004
8. Turalski S.: Wprowadzenie do DB2 9. Software Developer's Journal Extra, nr 21, 2007, s. 8
9. Damian Dudek.: Integralność danych, wyzwacze i wydajność. Software Developer's Journal, nr 11, 2005, s. 40-49
10. Sokulski J.: Architektura baz danych Berkeley DB. Software 2.0, nr 12, 2002, s. 33-37
11. Foremniak P. Szarwas P: Slony-I, system replikacji danych dla PostgreSQL. PHP solutions nr 10, 2005, s 58-59
12. Bogusz J.: MaxDB w praktyce. Software 2.0, nr 12, 2004, s. 24-27
- i8. <http://www.giodo.gov.pl>
- i9. <http://isip.sejm.gov.pl/servlet/Search?todo=open&id=WDU20011281402>
- i10. http://www.insert.com.pl/oferta/sprzedaz_i_produkcja/subiekt_gt/mozliwosci_programu.html
- i11. <http://www.oscommerce.com/>
- i12. <http://www.gnu.org/licenses/fdl.html>
- i13. <http://www.gnu.org/licenses/gpl.html>
- i14. <http://www.gnu.org/>
- i15. <http://encyklopedia.pwn.pl/haslo.php?id=3875256>
- i16. http://pl.wikipedia.org/wiki/Baza_danych
- i17. <http://www.knight.nazwa.pl/UMK/BD/Modele.pdf>

- i18. <http://misto.e-informatyka.pl/papers/dsms-intro.pdf>
- i19. <http://studia.elka.pw.edu.pl/pub/BADA.A/BADA2.ppt>
- i20. http://maczek.info.bielsko.pl/~zbiju/ATH/semestr_III/2003.11.18.Skrypt_do_egzaminu_Pana_L.Grinke_-_by_Endriu.pdf
- i21. <http://www.im.pwr.wroc.pl/~zato/2006L/bazy/slajdy/uwagi/baza-polaczenie.pdf>
- i22. [http://www.ipipan.waw.pl/~subieta/wyklady/Systemy%20rozproszone%20SYR%20\(podyplomowe\)%202002/SYR1.ppt](http://www.ipipan.waw.pl/~subieta/wyklady/Systemy%20rozproszone%20SYR%20(podyplomowe)%202002/SYR1.ppt)
- i23. <http://www.si.pjwstk.edu.pl/dydaktyka/JPS/old/RozproszFederBazyDanych1..5.ppt>
- i24. <http://docs.sun.com/source/819-2574/intro.html>
- i25. <http://www.eti.pg.gda.pl/katedry/kask/pracownicy/Michal.Piotrowski/srpliki/Transkacje%20rozproszone.pdf>
- i26. http://www.ploug.org.pl/seminarium/seminarium_VII/pliki/5a.pdf
- i27. http://zti.iinf.polsl.gliwice.pl/bd3/Replikacje_Instr.doc
- i28. [http://zti.iinf.polsl.gliwice.pl/bd3/RozprBD\(1\).ppt](http://zti.iinf.polsl.gliwice.pl/bd3/RozprBD(1).ppt)
- i29. http://ploug.org.pl/konf_04/materialy/pdf/hurtownie_danych.pdf
- i30. http://pl.wikipedia.org/wiki/Hurtownia_danych
- i31. <http://www.microsoft.com/sql/default.mspix>
- i32. <http://www.php.net>
- i33. <http://www.mysql.com>
- i34. <http://httpd.apache.org/>
- i35. <http://www.eclipse.org/>
- i36. <http://www.phpdoc.org/>
- i37. <http://www.cybertattoo.eu/>

Aneks

Płyta załączona do niniejszej pracy jest jej integralną częścią na której znajdują się:

praca_mgr.pdf – pracą magisterską

prezentacja.pdf – prezentacja pracy magisterskiej

/licencje/ – katalog z licencjami

/dokumentacja/ – katalog z dokumentacją skryptu

/install/ – katalog z dokumentacją instalacyjną skryptu i plikami pomocniczymi

/osCommerce/ – katalog zawierający wersję instalacyjną sklepu osCommerce

/patched_osCommerce/ – katalog zawierający przerobione pliki sklepu osCommerce

/StoC/ – katalog ze skryptem synchronizującym

GNU Free Documentation License

Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the

Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near

the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If

Techniki integracji baz danych i synchronizacji przesyłania danych między nimi.

the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Spis rysunków

Rys. 1.1. Budowa bazy danych.....	8
Rys. 1.2. Diagram związków encji dla baz danych filmów [2]	15
Rys. 1.3. Relacja studenci	15
Rys. 1.4. Przykładowa klasa student.....	17
Rys. 1.5. Przykładowa metoda klasy student.....	18
Rys. 1.6. Reprezentacja danych semistrukturalnych filmów i gwiazd [2]	19
Rys. 1.7. Schemat hierarchicznego modelu danych dla działu administracji [22]	20
Rys. 1.8. Przykładowe drzewo hierarchicznego modelu danych dla schematu bazy zamieszczonego na rysunku 1.7 [22].....	20
Rys. 1.9. Przykładowy wycinek sieciowej bazy danych [24].....	21
Rys. 2.1. Definicja relacji studenci w języku SQL z kluczem głównym utworzonym na atrybucie Nr_indeksu	24
Rys. 2.2. Instrukcje SQL tworzące więzy UNIQUE w relacji studenci	25
Rys. 2.3 . Instrukcje SQL definiujące relację kursy oraz związek zapisy pomiędzy studentami i kursami	25
Rys. 2.4. Instrukcje SQL definiujące relację wykładowcy i rozszerzające relację kursy o klucz obcy z relacji wykładowcy	27
Rys. 2.5. Schemat przetwarzania programów z osadzonymi instrukcjami SQL	35
Rys. 3.1. Architektura klient-serwer [28]	38
Rys. 3.2. Architektura klient-multi-serwer [28]	39
Rys. 3.3. Architektura koleżeńska	39
Rys. 3.4. Architektura oparta na oprogramowaniu pośredniczącym [29]	40
Rys. 3.5. Architektura grid.....	40
Rys. 3.6. Plan rozproszonej transakcji zatwierdzania dwufazowego [30].....	44
Rys. 3.7. Przykładowe środowisko replikacji multimaster.....	46
Rys. 3.8. Przykład środowiska replikacji migawkowej	46
Rys. 3.9. Przykładowe środowisko replikacji hybrydowej.....	47
Rys. 3.10. Przykładowy system typu klient-serwer.....	47
Rys. 3.11. Przykładowa jednorodna rozproszona baza danych.....	48
Rys. 3.12. Przykładowa niejednorodna rozproszona baza danych	49
Rys. 3.13. Systemy oparte na architekturze mediator [33].....	50
Rys. 3.14. Federacyjna baza danych złożona z trzech baz danych.....	51

Rys. 3.15. Podstawowa architektura systemu hurtowni baz danych [34].....	53
Rys. 4.1. Środowisko pracy programów Subiekt GT i osCommerce	55
Rys. 4.2. Główne fazy algorytmu	65
Rys. 5.1. Diagram wielowymiarowej tablicy asocjacyjnej przechowującej produkty i typy ..	74

Spis tabel

Tabela 2.1 Zmiana ilość wolnych miejsc w czasie trwania transakcji w przykładowej linii lotniczej.....	32
Tabela 3.1 Wynik złączenia zapytania do dwóch baz federacyjnej bazy danych	52
Tabela 4.1 sl_StawkaVAT - Słownik stawek VAT	60
Tabela 4.2 sl_GrupaTw - Słownik grup towarowych.....	60
Tabela 4.3 tw_CechaTw - Tabela złączeniowa towaru i cech.....	60
Tabela 4.4 sl_CechaTw - Słownik cech towarów.....	60
Tabela 4.5 tw_Stan - Stany magazynowe towarów	60
Tabela 4.6 tw__Towar – Ewidencja towarów	60
Tabela 4.7 tw_Cena - Ceny sprzedaży.....	61
Tabela 4.8 geo_zones - Grupy stref podatkowych	61
Tabela 4.9 tax_class - Grupa podatkowa	61
Tabela 4.10 tax_rates - Stawka podatkowa	61
Tabela 4.11 categories - Kategorie	62
Tabela 4.12 categories_description - Opis kategorii	62
Tabela 4.13 languages - Języki	62
Tabela 4.14 products_options - Rodzaje opcji.....	62
Tabela 4.15 products_options_values - Wartości opcji	62
Tabela 4.16 productes_attributes – Typy produktów	62
Tabela 4.17 products_to_categories - Relacja złączenia produktów i kategorii.....	63
Tabela 4.18 products - Produkty	63
Tabela 4.19 products_description - Opis produktów	63

OŚWIADCZENIE

Oświadczam, że niniejsza praca nie narusza praw autorskich w rozumieniu Ustawy z dnia 4.02.1994 o prawie autorskim i prawach pokrewnych (Dz. U. nr 24 poz. 83)

Czytelny podpis

.....