

**PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA
W NOWYM SĄCZU**

INSTYTUT TECHNICZNY

PRACA DYPLOMOWA

**SYSTEM GENEROWANIA RAPORTÓW W PROCESIE
REKRUTACJI KANDYDATÓW NA STUDIA PROWADZONE W
PWSZ W NOWYM SĄCZU.**

Autor: Paweł Mysiński

Kierunek: Informatyka

Nr albumu: 20747

Promotor: dr inż. Antoni Ligęza

NOWY SĄCZ 2015

Spis treści

1. Wprowadzenie	5
1.1. Zagadnienie generowania raportów	5
1.2. Dotychczasowy proces generowania raportów	5
1.3. Cel i zakres pracy	6
2. Szablony raportów w systemie Latex	7
2.1. Środowisko kompilacji raportów	7
2.2. Idea działania szablonów	7
2.3. Tworzenie szablonów raportów do systemu rekrutacji	7
2.3.1. Wyświetlanie listy	8
2.3.2. Grupowanie	8
2.3.3. Aktualizacja daty	8
3. Projekt programu i implementacja	9
3.1. Algorytm działania systemu raportowania	9
3.2. Wybór języka programowania	9
3.3. Wybór środowiska programistycznego	9
3.4. Proces tworzenia programu	9
3.4.1. Utworzenie projektu	9
3.4.2. Wybór bibliotek zewnętrznych	9
3.4.3. Zarządzanie konfiguracją	9
3.4.4. Obsługa połączenia z bazą danych	9
3.4.5. Wykorzystane klasy typu kontener	9
3.4.6. Zarządzanie szablonami	9
3.4.7. Uzupełnianie szablonów	9
3.4.8. Klasa zarządzająca kompilatorem Latex	9
3.5. Kompilacja programu	9
4. Uruchomienie oraz testowanie systemu	10
4.1. Wstępne ustalenia	10
4.1.1. Specyfikacja maszyny do testów	10
4.1.2. Struktura katalogowa	10
4.2. Generowanie przykładowych danych	11
4.2.1. Tworzenie bazy danych	11
4.2.2. Stworzenie struktury bazy	12
4.2.3. Generowanie testowych danych osobowych	13
4.2.4. Generowanie kandydatów na studentów	14
4.3. Dodanie zapytań SQL do szablonów	15
4.4. Konfiguracja programu	20
4.5. Przebieg testów	21
4.6. Wyniki testu	23
5. Instrukcja obsługi	24
5.1. Wymagania systemowe	24
5.2. Przygotowanie pliku konfiguracyjnego	24

5.3.	Przygotowanie prostego szablonu	24
5.4.	Dodatkowe polecenia w szablonach	24
5.4.1.	Polecenie puste	24
5.4.2.	Grupowanie	24
5.5.	Uruchomienie programu	24
5.6.	Rozwiązywanie problemów	24
6.	Podsumowanie	25
	Bibliografia	26

1. Wprowadzenie

Dokumenty, listy, protokoły to nierozłączna część każdego procesu rekrutacji na uczelni wyższej. Ręczne tworzenie takich dokumentów może narażać wielu problemów. Przede wszystkim głównym problemem jest możliwość wystąpienia błędów człowieka. W nawet najlepiej zorganizowanej placówce szkolnej może wkraść się błąd, który spowoduje iż na przykład kandydat zostanie przypadkowo odrzucony lub przyjęty na studia. Usuwanie takich błędów może być czasami nie możliwe, dlatego ręczne tworzenie takich dokumentów zajmuje wiele czasu, aby mieć pewność, by nie popełnić błędu. Przy czym istotnym jest, że rekrutacja powinna przebiegać szybko ze względu na fakt, iż kandydaci potrzebują znać decyzję o przyjęciu w miarę szybko aby móc w razie czego zgłosić się do innych szkół w tym samym okresie. Dodać jeszcze należy, że podczas nieefektywnie zorganizowanej rekrutacji, w procesie tym udział muszą wziąć pracownicy dydaktyczni, którzy muszą poświęcać swój czas na okres rekrutacji.

1.1. Zagadnienie generowania raportów

Z rozwiązaniem tego problemu przychodzi informatyzacja procesu rekrutacji. Pozwala ona na eliminację praktycznie wszystkich błędów człowieka poprzez automatyczne korekty danych oraz auto uzupełnianie dokumentów danymi. Przyspiesza proces rekrutacji poprzez ułatwienie go dla osób prowadzących go oraz fakt iż dokumenty generowane są w bardzo krótkim czasie.

W procesie generowania raportów zawsze bierze udział pewien rodzaj bazy danych, z której pobierane są informacje. Dane są poddawane selekcji a następnie wklejane odpowiednie miejsce we wcześniej przygotowany szablon danego raportu. W ten sposób powstaje dokument wygenerowany z dynamicznych danych, zapisany w różnych formach. Najpopularniejszą formą zapisu jest *Portable Document Format* w skrócie *PDF*. Dokument wygenerowany do tego formatu pozwala na natychmiastowe przejrzanie zawartości oraz możliwość wydruku.

1.2. Dotychczasowy proces generowania raportów

Po przeanalizowaniu obecnego systemu można stwierdzić, iż jest on mało wydajny i wymaga poprawy. Do generowania dokumentów używana jest funkcja programu *IBExpert, Report Manager*. Stworzenie szablonu raportu w tym podprogramie jest procesem żmudnym oraz skomplikowanym dla osoby zajmującej się administracją bez wiedzy informatycznej. Dodatkowo należy dodać, że nie posiada on wystarczającej funkcjonalności, przez co proces generowania raportów nie przebiega w pełni automatycznie i wymaga pełnego nadzoru osoby wdrożonej w ten system. Dobrym przykładem na pokazanie niedoskonałości jest fakt, iż każdy raport musi być generowany oddzielnie dla każdego kierunku studiów, stopnia czy też formy. Wytwarza to problemy związane z powstawaniem błędów czy też segregacją raportów.

1.3. Cel i zakres pracy

Celem pracy, czyli informatyzacji procesu rekrutacji, jest zapewnienie pracownikom administracji jak i również osobom upoważnionym, dostępu do narzędzia użytecznego, bezpiecznego oraz prostego w stosowaniu, które w znacznym stopniu przyspieszy ten proces oraz zapewni minimalizację błędów, które mogą powstać podczas tworzenia dokumentów potrzebnych przy rekrutacji.

Osiągnięcie postawionego celu, wymaga wykonania następujących zadań:

1. Wybranie systemu generowania dokumentów z wcześniej uzupełnionych szablonów.
2. Wybranie języka programowania oraz środowiska tego języka do zaimplementowania programu, który będzie stanowił interfejs dla użytkownika oraz wykonywał algorytm uzupełniania szablonów raportów.
3. Utworzenie szablonów dokumentów w procesie rekrutacji odpowiadających tym, które są obecnie używane.
4. Zaimplementowanie programu zdolnego uzupełnić wykonane wcześniej szablony oraz zapewnić łatwe użytkowanie.
5. Przeprowadzenie testu działania systemu na wyczerpującej potrzeby liczbie kandydatów.
6. Przygotowanie dokumentacji oraz instrukcji obsługi systemu.

2. Szablony raportów w systemie Latex

2.1. Środowisko kompilacji raportów

blablabla nie mam pojęcia

2.2. Idea działania szablonów

Do wszystkich tych dokumentów potrzebny jest szablon w języku oprogramowania do zautomatyzowanego składu tekstu. W tej pracy został wybrany program LaTeX ze względu na jego możliwości automatyzacji procesu parsowania danych i uzupełniania nimi danych miejsc w tekście.

Stworzenie szablonów polega, więc na wcześniejszym przygotowaniu plików tex, zawierających wcześniej strukturę danego dokumentu z "pustymi" miejscami do uzupełnienia przez program. Do uzupełnienia tych miejsc można wykorzystać funkcję LaTeXu jaką jest tworzenie nowych środowisk z parametrami, gdzie odpowiednio parametry te będą wartościami, które zostaną wpisane w dane miejsce w danym dokumencie. Następnie wystarczy wywołać dane środowisko z odpowiednimi wartościami aby otrzymać uzupełniony dokument. Dane środowisko możemy wywoływać wielokrotnie od różnych wartości tworząc w ten sposób wiele dokumentów tego samego typu o różnych zmiennych wartościach takich jak np imię i nazwisko.

Do wytworzenia wywołań tych środowisk posłuży właśnie program stworzony w javie. Poprzez dodanie zapytania SQL w odpowiedniej formule do plików tex. Program DBRaportLatex wyszuka takie zapytanie i uzupełni szablon wywołaniami środowisk z wartościami parametrów, jakimi będą wartości pola z rekordów zapytania SQL.

2.3. Tworzenie szablonów raportów do systemu rekrutacji

W rekrutacji na uczelnie wykorzystuje się dokumenty, które należało dokładnie odwzorować w nowym systemie. Są to następujące dokumenty:

1. protokół przekazania
2. listy potwierdzenia podjęcia studiów
3. listy rankingowe
4. listy przyjętych
5. listy nieprzyjętych
6. decyzja o przyjęciu danego kandydata
7. decyzja o nieprzyjęciu danego kandydata

Przy tworzeniu szablonów wystąpiły powtarzające się problemy, które należało rozwiązać. Jako że rozwiązania tych problemów powtarzają się, to zamiast opisywania każdego szablonu po kolei, przedstawione poniżej zostały najważniejsze problemy, wynikające z tworzenia tych szablonów.

2.3.1. Wyświetlanie listy

Problem występujący w protokole przekazania oraz we wszystkich listach. Przykładowo potrzebujemy wyświetlić poniższą listę, gdzie oczywiście wartości pochodzą z bazy danych:

```
Lp. Tok studiów Liczba kopert
1 Informatyka -- niestacjonarne STUDIA pierwszego stopnia 1455
2 Informatyka -- stacjonarne STUDIA pierwszego stopnia 729
3 Mechatronika -- niestacjonarne STUDIA pierwszego stopnia 1447
...
```

2.3.2. Grupowanie

2.3.3. Aktualizacja daty

3. Projekt programu i implementacja

a

3.1. Algorytm działania systemu raportowania

3.2. Wybór języka programowania

3.3. Wybór środowiska programistycznego

3.4. Proces tworzenia programu

3.4.1. Utworzenie projektu

3.4.2. Wybór bibliotek zewnętrznych

3.4.3. Zarządzanie konfiguracją

3.4.4. Obsługa połączenia z bazą danych

3.4.5. Wykorzystane klasy typu kontener

3.4.6. Zarządzanie szablonami

3.4.7. Uzupełnianie szablonów

3.4.8. Klasa zarządzająca kompilatorem Latex

3.5. Kompilacja programu

4. Uruchomienie oraz testowanie systemu

Przed wdrożeniem programu do realnego systemu, program należy przetestować. Testy powinny być prowadzone na tymczasowej bazie danych, ponieważ idea testów jest taka, że po podmianie bazy danych na realną wszystko ma działać bez zmian. Zmienić się może tylko zapis połączenia z bazą danych w pliku konfiguracyjnym. Dzięki takiemu zabiegowi, będzie można być pewnym tego, że wszystko będzie działać na prawdziwej bazie danych. Do przeprowadzenia testów potrzebne będzie odtworzyć przyszłe środowisko, w którym będzie działać program, przygotować szablony dokumentów, które są wytwarzane w procesie rekrutacji oraz wygenerować dokumenty. Ostatnim już krokiem będzie sprawdzenie czy podczas tego procesu nie ma żadnych komplikacji oraz czy wygenerowane dokumenty nie zawierają błędów.

4.1. Wstępne ustalenia

4.1.1. Specyfikacja maszyny do testów

Testy przeprowadzone będą na komputerze o następującej specyfikacji:

```
OS Name Microsoft Windows 7 Ultimate
Version 6.1.7601 Service Pack 1 Build 7601
System Model      Z97-1stlisting
System Type       x64-based PC
Processor         Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz,
3301 Mhz, 4 Core(s), 4 Logical Processor(s)
Installed Physical Memory (RAM) 8,00 GB
Total Physical Memory 7,86 GB
Available Physical Memory 3,99 GB
Total Virtual Memory 9,86 GB
Available Virtual Memory 3,75 GB
Page File Space 2,00 GB
```

Do działania programu DBRaportlatex zainstalowana została JAVA w wersji:

```
C:\Users\Pawlos>java -version
java version "1.8.0_60"
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

4.1.2. Struktura katalogowa

Do przejrzystego przechowywania wszystkich rzeczy potrzebnych w systemie rekrutacji, potrzebna będzie odpowiednia struktura katalogów. Do przechowywania potrzebne będą:

- program: DBRaportLatex.jar
- plik konfiguracyjny: dblatexraportconfig.bat
- środowisko kompilacji latex
- szablony dokumentów
- plik bazy danych(tylko w testach)
- pliki wynikowe
- biblioteki potrzebne dla silnika bazy danych firebird w wersji embedded

Proponowana struktura więc może wyglądać następująco:

```

/
├─ Firebird
├─ output
├─ template
├─ texlive
├─ DBRaportLatex.jar
├─ dblatexraportconfig.bat
└─ rekrutacja.fdb

```

Do katalogu *texlive* przegrane zostało środowisko do kompilacji plików latex. W katalogu firebird są biblioteki tylko na czas testów do wersji embedded bazy danych. W katalogu *template* dgrane zostaną szablony dokumentów.

4.2. Generowanie przykładowych danych

W celu przetestowania systemu generowania raportów w procesie rekrutacji kandydatów na studia potrzebne będą testowe dane w dokładnie tej samej strukturze co w systemie rekrutacji, ponieważ w szablona latexu znajdują się zapytania SQL do danych tabel w bazie danych. W celu otrzymania tych danych potrzebne będzie: 1. Utworzenie nowej bazy danych na silniku Firebird'a 2. Stworzenie wystarczającej struktury tabel odzwierciedlającą strukturę w systemie uczelnianym. 3. Wygenerowanie dużej ilości testowych danych osobowych. 4. Uzupełnienie tabel danymi, które zostały wygenerowane wcześniej oraz dodanie do nich dodatkowych, jednocześnie losowych, informacji na temat procesu rekrutacji. Po wykonaniu tych kroków, powinna powstać baza danych do której bez problem program połączy się i wyciągnie z niej potrzebne dane dokładnie jak z realnej bazy danych.

4.2.1. Tworzenie bazy danych

Do stworzenia pliku bazy danych na silniku firebird'a posłużyć się można narzędziem dostępnym w katalogu bin zainstalowanego serwera firebird. Narzędzie to pozwala z linii komend tworzyć i łączyć się z bazami danych. W tym przypadku użyta zostanie komenda *CREATE DATABASE*

```

C:\Program Files\Firebird\bin>isql
SQL>CREATE DATABASE 'D:\test systemu\rekrutacja.fdb'
CON>user 'SYSDBA' password 'masterkey';

```

Po wykonaniu tego polecenia zostanie utworzona baza danych. Takie same dane należy teraz wpisać do pliku konfiguracyjnego programu czyli *DBRaportLatex.bat* aby program mógł się połączyć z tą bazą:

```

dbengine=firebirdsql
hostname=//localhost
port=3050
dbpath=D:\test systemu\rekrutacja.fdb
user=SYSDBA
password= masterkey

```

4.2.2. Stworzenie struktury bazy

Dotychczasowy system wykorzystywał tabelę (widok) która była generowana dynamicznie i która zawiera wszystkich studentów w rekrutacji. Zawiera ona wszystkie dane potrzebne do wytworzenia dokumentów. Jeden rekord to jeden student ze wszystkimi informacjami na jego temat. Dodatkowo jeszcze potrzebna jest tabela z informacjami na temat rekrutacji, takimi jak na przykład nazwisko i imię przewodniczącego komisji, czy data wydania decyzji przyjęcia studenta. Z tych tabel będą pobierane informacje, natomiast do wygenerowania danych potrzebne będą dwie dodatkowe tymczasowe tabele. Tabela główna z kandydatami(zapis SQL):

```

CREATE TABLE KANDYDAT_ALIGEZA (
    STUD_ID                INTEGER NOT NULL,
    OSOBA_ID                INTEGER NOT NULL,
    STUD_NRTECZKI           INTEGER,
    NAZWISKO                VARCHAR(100),
    IMIE                    VARCHAR(100),
    NAZWISKOIMIONA          VARCHAR(200),
    ADR_ULICA_MIEJSCOWOSC_NR_DOMU VARCHAR(200),
    ADR_KOD_POCZTOWY_POCZTA VARCHAR(100),
    STUDIA_NAZWA            VARCHAR(100),
    STUD_ILPUNKTOW           INTEGER,
    STUD_ILPUNKTOWKREM       INTEGER,
    TOKNAUKI_NAZWATOKU      VARCHAR(200),
    KIERUNEK                VARCHAR(100),
    SPEC_ID                 INTEGER,
    DATAPRZYJECIAPODANIA    DATE,
    TOKNAUKI_ID             INTEGER,
    OSOBA_PESSEL            VARCHAR(50),
    KIERUNEK_MY             VARCHAR(200),
    FORMA_STUDIOW_MY        VARCHAR(200),
    STOPIEN_STUDIOW_MY      VARCHAR(100),
    KIERUNEK_FORMA_SKROT_MY  VARCHAR(100),
    NR_DECYZJI              VARCHAR(100),
    CZY_PRZYJETY            INTEGER,
    DATA_DECYZJI           DATE,
    ILE_PUNKTOW             INTEGER,
    PANPANI                 CHAR(1)
);

```

Tabela z dodatkowymi informacjami(wartości przypisane są do kluczy tekstowych, jest to tablica asocjacyjna):

```
CREATE TABLE SETUP_ALIGEZA (
    KLUCZ    VARCHAR(50) NOT NULL,
    WARTOSC  VARCHAR(100)
);
```

Tymczasowa tabela do zaimportowania listy imion i nazwisk oraz losowych peseli.

```
CREATE TABLE DANE (
    IMIE_NAZ  VARCHAR(200),
    ADRES     VARCHAR(200),
    PESEL     VARCHAR(50),
    NAZWISKO  VARCHAR(100),
    IMIE      VARCHAR(100)
);
```

Tymczasowa tabela do procedury losowego uzupełniania informacji o rekrucie o kierunku jaki wybrał.

```
CREATE TABLE TOKNAUKI_ALIGEZA (
    TOKNAUKI_ID          INTEGER NOT NULL,
    KIERUNEK_MY          VARCHAR(50),
    FORMA_STUDIOW_MY    VARCHAR(50),
    STOPIEN_STUDIOW_MY  VARCHAR(50),
    KIERUNEK_FORMA_SKROT_MY VARCHAR(10),
    LICZBA_MIEJSC        SMALLINT,
    DATA_DECYZJI_OD     TIMESTAMP,
    DATA_DECYZJI_DO     TIMESTAMP,
    KOD_IKR              VARCHAR(3)
);
```

4.2.3. Generowanie testowych danych osobowych

Do wygenerowania kandydatów potrzebne jest imię, nazwisko oraz adres. Takie dane dostępne są w książkach telefonicznych. Posługując się jedną z nich stworzony został plik CSV o separatorze „;” zawierający po kolei imię z nazwiskiem, adres, pesel, nazwisko oraz imię. Pesel został dodany do każdej osoby jako losowy ciąg cyfr spełniający walidację peselu. Ze względu na fakt iż pesel został wygenerowany losowo, w wygenerowanych dokumentach nie będzie możliwości ustawienia płci osoby, dlatego też w bazie danych do testów 5 pierwszych kandydatów dostanie płeć żeńską, a reszta męską.

Struktura pliku:

```
Abram Andrzej; Lwowska 116;88071640299;Abram;Andrzej
Abram Halina; Ludwika Zamenhofa 2;86111210691;Abram;Halina
...
```

Tak sformatowany plik CSV, łatwo zaimportować do bazy danych do tabeli *dane* ze względu na identyczną kolejność danych w kolumnach. Do importu wykorzystana została funkcja programu *IBExpert*, *import data*. Jedna linijka w pliku zostaje zaimportowana jako jeden rekord, w którym każde pole po kolei odpowiada wartościom między średnikami. Zaimportowanych w ten sposób zostało 10001 rekordów (osób) do tabeli *dane* do dalszych manipulacji, co całkowicie wyczerpuje ilość kandydatów potrzebnych do testów. Ze statystyk wynika, że na uczelni średnio na rekrutację przypada około trzystu kandydatów.

4.2.4. Generowanie kandydatów na studentów

Kolejnym krokiem jest uzupełnienie tabeli z tokami studiów. W testach dodanych zostało 8 przykładowych toków nauki.

1	Informatyka	niestacjonarne	pierwszego stopnia	INF-n
2	Mechatronika	niestacjonarne	pierwszego stopnia	MT-n
3	Mechatronika	stacjonarne	pierwszego stopnia	MT-s
...				

Uzupełnienia wymaga także tabela z dodatkowymi informacjami *setup_aligeza* przykładowymi danymi:

dataWydaniaDecyzji	09.10.2015
miejsceWydaniaDecyzji	Nowy Sącz
przewodniczącyIKR	mgr inż. Sławomir Jurkowski
rokAkademicki	2015/2016
czyUwzględnicDateWydaniaDecyzji	N
...	

Mając już to wszystko potrzebna jest procedura, która utworzy listę kandydatów z tych wszystkich danych.

```
create procedure GENERUJ
returns (
    TESTCHAR varchar(50),
    TEST integer)
as
declare variable IMIE varchar(100);
declare variable NAZ varchar(100);
declare variable IMIENAZ varchar(200);
declare variable ADRES varchar(200);
declare variable PESEL varchar(50);
declare variable LICZNIK integer;
declare variable STOPIEN varchar(50);
declare variable KIERUNEK varchar(50);
declare variable FORMA varchar(50);
declare variable SKROT varchar(10);
declare variable RANDINT integer;
declare variable PUNKTY integer;
declare variable CZY_PRZYJETY integer;
declare variable DATA_DEC varchar(100);
begin
    licznik = 1;
    for select * from dane into
        :imienaz, :adres, :pesel, :naz, :imie
    do
        begin
            randint = CAST(round(rand()*7+1) as INTEGER);
            punkty = CAST(round(rand()*500) as INTEGER);
            if(punkty > 250) then czy_przyjety = 1;
            if(punkty <= 250) then czy_przyjety = 2;
```

```

select kierunek_my,forma_studiow_my,stopien_studiow_my,
kierunek_forma_skrot_my
FROM toknauki_aligeza where toknauki_id = :randint
INTO :kierunek,:forma,:stopien,:skrot;

select wartosc FROM setup_aligeza
WHERE klucz='dataWydaniaDecyzji'
INTO :data_dec;

INSERT INTO kandydat_aligeza
(stud_id,osoba_id,stud_nrteczki,nazwisko,imie,
nazwiskoimiona,adr_ulica_miejscowosc_nr_domu,
adr_kod_pocztowy_poczta,osoba_pesel,panpani,
studia_nazwa,toknauki_nazwatoku,kierunek,
kierunek_my,forma_studiow_my,stopien_studiow_my,
kierunek_forma_skrot_my, stud_ilpunktow,
stud_ilpunktowkrem,ile_punktow,czy_przyjety,
data_decyzji,nr_decyzji,dataprzyjeciapodania)
VALUES (:licznik,:licznik,cast(round(rand()*200+1) as integer),
:naz,:imie,:imienaz,:adres,
cast( 'Nowy Sacz 33-300' as varchar(100)),
:pesel,cast( 'M' as char(1)), :forma,:kierunek ||
' N inz. 3.50 2015/2016 zimowy',:kierunek,:kierunek,:forma,
:stopien,:skrot, :punkty,:punkty,:punkty,:czy_przyjety,
cast(:data_dec as DATE),'328/2015','2015-08-14');

licznik = :licznik + 1;
end
test = :licznik;
suspend;
end

```

Powyższa procedura z jednej osoby z tabeli *dane* tworzy jednego kandydata, losując mu tok nauki, ilość punktów oraz czy zostanie przyjęty lub nie. Dodawane są także pewne stałe wartości, podobne do tych w oryginalnej bazie danych, które nie wymagają uzmiennienia. Procedura ta, po jednorazowym wywołaniu, wygenerowała 10001 rekordów w tabeli *kandydat_aligeza*. Daje to wystarczającą ilość testowych kandydatów do przeprowadzenia testów. Baza danych z przeprowadzonych testów znajduje się w załączniku.

4.3. Dodanie zapytań SQL do szablonów

Stworzone szablony w poprzednim rozdziale nie zawierają zapytań, które odzwierciedlałyby wyselekcjonowanie danych z istniejącej struktury bazy. Zawierają tylko sztywne dane testujące środowiska latexowe. Aby szablony podczas procesu tworzenia dokumentów zostały uzupełnione poprawnymi danymi potrzeba je uzupełnić o odpowiednia zapytania SQL w odpowiedniej formie tak aby parser zawarty w programie był w stanie je znaleźć i wykonać.

Potrzebnych jest 9 zapytań, za pomocą których wyciągnięte zostaną dane do szablonów. Poniżej opisane zostały każde z zapytań oraz przykład jakie dane zostaną wyselekcjonowane z bazy przez to zapytanie. Ta sekcja skupia się tylko i wyłącznie na uzyskaniu potrzebnych danych do środowisk latexu, a nie jak one zostaną wykorzystane. Dokładny opis jak wygląda wykorzystanie takiego zapisu znajduje się w rozdziale 2. Dodać jeszcze należy że poniższe zapytania znajdują się pod koniec każdego szablonu, zaraz po zdefiniowaniu środowisk. Takie umiejscowienie tych zapytań sprawi, że dane zostaną uzupełnione zaraz pod tymi zapytaniami.

Pierwsze zapytanie które zostanie omówione jest to zapytanie o podstawowe parametry, informacje na temat rekrutacji. Znajduje się ono w pliku `aaasetup.tex`. Plik posiada 3 litery `a` w nazwie ponieważ program parsujący, otwiera pliki alfabetycznie a zapytanie to musi zostać wykonane jako pierwsze. Poniżej mamy przykład danych do zapisania:

```
\parametrRekrutacyjny{dataWydaniaDecyzji}{09.10.2015}
\parametrRekrutacyjny{miejsceWydaniaDecyzji}{Nowy Sącz}
\parametrRekrutacyjny{rokAkademicki}{2015/2016}
...
```

Aby uzyskać takie dane, potrzeba będzie pobrać wszystkie rekordy z tabeli `setup_aligeza` (struktura tabeli znajduje się w poprzedniej sekcji) klucz oraz wartość. Zapytanie nie będzie więc zbyt skomplikowane:

```
SELECT klucz,wartosc FROM setup_aligeza
```

Po wywołaniu go faktycznie otrzymamy dane potrzebne nam dane. Jednak aby program parsujący był w stanie znaleźć takie zapytanie w szablonie potrzeba je obudować w odpowiednią konstrukcję, która będzie odpowiadać wyrażeniu regularnemu zapisanemu w parserze. Należy także wpisać nazwę środowiska w odpowiednie miejsce między znakami `@@`. Dodatkowo musimy jeszcze postarać się aby kompilator latex'a zignorował nasz zapis i go nie wyświetlał. Do tego celu zastosowana została komenda `iffalse`.

```
\iffalse@@parametrRekrutacyjny@@
SELECT klucz,wartosc FROM setup_aligeza
@end@@\fi
```

Następnie zapytanie zawarte w pliku `protokolprzekazania.tex`. Występuje tu nowy problem utworzenia środowiska na początku oraz zamknięcia na końcu. Pokazane jest to na przykładowych danych poniżej:

```
\protokolprzekazaniaA{1}
\protokolprzekazania{Informatyka ---
niestacjonarne STUDIA pierwszego stopnia}{1455}
\protokolprzekazania{Informatyka ---
stacjonarne STUDIA pierwszego stopnia}{729}
\protokolprzekazania{Mechatronika ---
niestacjonarne STUDIA pierwszego stopnia}{1447}
...
\endprotokolprzekazaniaA
```

Pomijając na razie otwarcie środowiska `protokolprzekazaniaA{1}` oraz zamknięcie `endprotokolprzekazaniaA`, w zapytaniu potrzebujemy wyświetlić każdy tok nauczania oraz z agregowaną liczbę kandydatów na tym toku. Do agregacji użyjemy polecenia `group` na polu tok

by móc otrzymać liczbę kandydatów używając funkcji count(*). Dodatkowo potrzeba także wybrać kandydatów tylko posiadającą daną datę decyzji. Do tego posłuży podzapytanie, które pobierze datę decyzji z aktualnej rekrutacji i porówna ją z datą decyzji dla danego kandydata.

```
SELECT k.kierunek_my||' --- '||k.forma_studiow_my||
' STUDIA '||k.stopien_studiow_my AS tok, count(*) AS ile
FROM kandydat_aligeza k
WHERE k.data_decyzji=(SELECT wartosc FROM
  setup_aligeza WHERE klucz='dataWydaniaDecyzji')
GROUP BY tok
```

Takie zapytanie wyświetli nam dane pomiędzy tymi pominiętymi dotychczas komendami. Rozwiązaniem wyświetlenia tych komend jest funkcja programu uzupełniającego szablony, a dokładnie funkcja grupowania. Dokładny opis tej funkcji znajduje się w rozdziale ref. Do zapytania dodamy jako pierwsze pole wartość 1, a następnie użyta zostanie funkcja grupowania po 1 polu wewnątrz szablonu:

```
\iffalse @@protokolprzekazania@1@@
SELECT 1 AS n,k.kierunek_my||' --- '||
k.forma_studiow_my||' STUDIA '||
k.stopien_studiow_my AS tok, count(*) AS ile
FROM kandydat_aligeza k
WHERE k.data_decyzji=(SELECT wartosc FROM
  setup_aligeza WHERE klucz='dataWydaniaDecyzji')
GROUP BY tok
@END@\fi
```

Dla szablonu 'pocztex.tex' potrzebujemy prostą listę z adresami każdego kandydata z danej rekrutacji:

```
\iffalse@@pocztex@@
SELECT ka.nazwiskoimiona,
      ka.adr_ulica_miejscowosc_nr_domu,
      ka.adr_kod_pocztowy_poczta,
      ka.stud_nrteczki||' '||ka.kierunek_forma_skrot_my||' '||ka.s
FROM kandydat_aligeza ka
WHERE ka.data_decyzji=
(select wartosc from setup_aligeza
where klucz='dataWydaniaDecyzji')
ORDER BY ka.kierunek_forma_skrot_my, ka.nazwiskoimiona
@END@\fi
```

W kolejnym szablonie *listasekretariat.tex* potrzebne jest utworzenie list kandydatów dla każdego toku nauki. Przez tok nauki rozumie się każdą kombinację kierunku, formy oraz stopnia studiów. W szablonie zostało to rozwiązane tak, że każda lista otwiera się poprzez wywołanie komendy *listarsekretariatA* z trzema parametrami, którymi są właśnie kierunek, forma oraz stopień studiów. Na przykładowych danych wygląda to następująco:

```
\listarsekretariatA{drugiego stopnia}{Informatyka}{stacjonarne}
\listarsekretariat{ Ablewicz Monika}{39}
\listarsekretariat{ Abram Halina}{69}
```

```

...
\endlistarsekretariata
\listarsekretariata{drugiego stopnia}{Mechatronika}{stacjonarne}
\listarsekretariata{ Adamek Danuta}{141}
\listarsekretariata{ Adamek Urszula}{35}
\listarsekretariata{ Adamik Zbigniew}{95}
...
\endlistarsekretariata
...

```

W zapytaniu więc potrzebne będzie wybrać wszystkich kandydatów z datą decyzji aktualnej rekrutacji, a następnie wyświetlić kierunek, formę, stopień studiów, imię, nazwisko oraz numer teczek. W zapytaniu także potrzebne będzie sortowanie po polach, które będą grupowane w programie uzupełniającym szablon. Każde na tych polach w zapytaniu SQL użyjemy polecenia *ORDER BY*. Ostatecznie poinformujemy program, aby otrzymane dane grupował po 3 pierwszych polach:

```

\iffalse@@listarsekretariata@3@@
SELECT stopien_studiow_my, kierunek_my, forma_studiow_my,
        nazwiskoimiona, stud_nrteczki
FROM kandydat_aligeza
WHERE data_decyzji=
(select wartosc from setup_aligeza
 where klucz='dataWydaniaDecyzji')
ORDER BY kierunek_forma_skrot_my, nazwiskoimiona
@END@\fi

```

W szablonach znajdują się jeszcze 4 kolejne podobne listy: przyjętych, nieprzyjętych, wysyłkowa oraz rankingowa. Zasada utworzenia zapytań do tych list jest identyczna jak w liście do sekretariatu. Różnią się one będą jedynie pewnymi warunkami wewnątrz zapytań SQL czy też polami po których będą sortowane dane.

Lista przyjętych:

```

\iffalse@@listaprzyjetych@3@@
SELECT stopien_studiow_my, kierunek_my, forma_studiow_my,
        nazwiskoimiona
FROM kandydat_aligeza
WHERE czy_przyjety=1 and
        data_decyzji=
        (SELECT wartosc FROM setup_aligeza
         WHERE klucz='dataWydaniaDecyzji')
ORDER BY kierunek_forma_skrot_my,
        nazwiskoimiona
@END@\fi

```

Lista nieprzyjętych:

```

\iffalse@@listaprzyjetych@3@@
SELECT stopien_studiow_my, kierunek_my, forma_studiow_my,
        nazwiskoimiona
FROM kandydat_aligeza

```

```

WHERE czy_przyjety=2 and
      data_decyzji=(SELECT wartosc FROM setup_aligeza
                     WHERE klucz='dataWydaniaDecyzji')
ORDER BY kierunek_forma_skrot_my,
         nazwiskoimiona
@END@\fi

```

Lista wysyłkowa:

```

\iffalse @@listawysylkowa@3@@
SELECT stopien_studiow_my,kierunek_my,forma_studiow_my,
       nazwiskoimiona, adr_ulica_miejscowosc_nr_domu||'; '||
       ADR_KOD_POCZTOWY_POCZTA
FROM kandydat_aligeza
WHERE data_decyzji=(select wartosc from setup_aligeza
where klucz='dataWydaniaDecyzji')
ORDER BY kierunek_forma_skrot_my, nazwiskoimiona
@END@\fi

```

Lista rankingowa:

```

\iffalse @@listarankingowa@3@@
SELECT stopien_studiow_my,kierunek_my,forma_studiow_my,
       nazwiskoimiona,STUD_ILPUNKTOW
FROM kandydat_aligeza
WHERE data_decyzji=(select wartosc from setup_aligeza
where klucz='dataWydaniaDecyzji')
ORDER BY kierunek_forma_skrot_my, STUD_ILPUNKTOW desc
@END@\fi

```

W ostatnim szablonie wystąpił problem ilości parametrów jakie trzeba przekazać do poprawnego działania szablonu. W latex'ie można stworzyć środowisko o maksymalnie 9 parametrach, natomiast w szablonie *decyzja.tex* dla każdego kandydata potrzeba 12 różnych informacji. Rozwiązaniem tego problemu także było grupowanie przez program uzupełniania raportów. Grupowanie wytnie 3 pierwsze pola i wstawi je do nowego środowiska. W zapytaniu SQL wyświetlanie daty musi posiadać dodatkowe 0 przy datach typu 01.02.2015 gdzie właśnie liczby dni lub miesiące są mniejsze od 10:

```

\iffalse @@decyzja@3@@
SELECT ka.stopien_studiow_my,ka.kierunek_my
,ka.forma_studiow_my,
ka.stud_ilpunktow,ka.nr_decyzji,ka.nazwiskoimiona,
ka.adr_ulica_miejscowosc_nr_domu,
ka.adr_kod_pocztowy_poczta,
ka.stud_nrteczki||' '||ka.kierunek_forma_skrot_my||
' '||ka.stud_id,czy_przyjety,
extract(day from ka.dataprzyjeciapodania)||'. '||
substring(100+extract(month from ka.dataprzyjeciapodania)
from 2 for 2)||'. '||
extract(year from ka.dataprzyjeciapodania),panpani
FROM kandydat_aligeza ka

```

```
WHERE ka.data_decyzji=(select wartosc from setup_aligeza
where klucz='dataWydaniaDecyzji')
ORDER BY ka.kierunek_forma_skrot_my, ka.nazwiskoimiona
@END@\fi
```

4.4. Konfiguracja programu

W głównym katalogu znajduje się plik konfiguracyjny programu o nazwie *dblatexraportconfig.bat*. Plik ten jest jednocześnie plikiem wsadowym oraz zawiera konfigurację programu *DBRaportlatex.jar*. Część kodu batch uruchamia program DBRaportlatex razem z odpowiednimi parametrami do uruchomienia embedded firebird:

```
@echo off
java -Djava.library.path=.\firebird -jar dbraportlatex.jar
pause
exit
```

Następnie od liniiki zawierającej *#dbLatexRaportConfig* zaczynają się zmienne konfiguracyjne:

— Ścieżka do szablonów latex

```
templatepath=template/
```

— Ścieżka do uzupełnionych szablonów latex oraz skompilowanego pliku pdf

```
output=output/
```

— Ścieżka do uzupełnionych szablonów latex oraz skompilowanego pliku pdf

```
output=output/
```

— Kodowanie szablonów latex

```
encodingtex=UTF-8
```

— Konfiguracja połączenia z bazą danych

```
dbengine=firebirdsql
hostname=embedded
dbpath=rekrutacja.fdb
user=SYSDBA
password=
dbencoding=WIN1250
```

— Ścieżka do kompilatora latex

```
pdflatexpath=texlive\2010min\bin\win32\pdflatex.exe
```

— Lista plików do kompilacji

```
pdfcompilemainfile=main.tex,main.tex
```

4.5. Przebieg testów

W bazie danych znajduje się 10001 kandydatów. Należy spodziewać się tyle samo wygenerowanych na listach. Program został uruchomiony z pliku wsadowego *dblatexraport-config.bat*. Poniżej znajduje się wydruk z konsoli po uruchomieniu programu z krótkimi opisami poszczególnych procesów:

1. Ładowanie pliku konfiguracyjnego przez program:

```
Config loaded: dblatexraportconfig.bat
pattern=
templatepath=template/
output=output/
encodingtex=UTF-8
dbengine=firebirdsql
hostname=embedded
dbpath=rekrutacja.fdb
user=SYSDBA
password=*****
dbencoding=WIN1250
pdflatexpath=texlive\2010min\bin\win32\pdflatex.exe
pdfcompilemainfile=main.tex,main.tex
```

2. Połączenie z bazą danych:

```
Connection to database: OK
```

3. Ładowanie szablonów do pamięci:

```
Loaded file: template\aaasetup.tex
Loaded file: template\decyzja.tex
Loaded file: template\listanieprzyjetych.tex
Loaded file: template\listaprzyjetych.tex
Loaded file: template\listarankingowa.tex
Loaded file: template\listarsekretariat.tex
Loaded file: template\listawysylkowa.tex
Loaded file: template\main.tex
Loaded file: template\pocztex.tex
Loaded file: template\protokolprzekazania.tex
```

4. Wywoływanie zapytań SQL oraz uzupełnianie szablonów:

```
aaasetup.tex - processing SQL statements
no returns SQL1
9 records SQL2
```

```
decyzja.tex - processing SQL statements
10001 records SQL1
```

```
listanieprzyjetych.tex - processing SQL statements
5051 records SQL1
```

```
listaprzyjetych.tex - processing SQL statements
```

4950 records SQL1

listarankingowa.tex - processing SQL statements
10001 records SQL1

listarsekretariat.tex - processing SQL statements
10001 records SQL1

listawysylkowa.tex - processing SQL statements
10001 records SQL1

main.tex - processing SQL statements

pocztex.tex - processing SQL statements
10001 records SQL1

protokolprzekazania.tex - processing SQL statements
8 records SQL1

5. Zapisywanie uzupełnionych szablonów do katalogu wynikowego:

Saved to file: output/aaasetup.tex
Saved to file: output/decyzja.tex
Saved to file: output/listanieprzyjetych.tex
Saved to file: output/listaprzyjetych.tex
Saved to file: output/listarankingowa.tex
Saved to file: output/listarsekretariat.tex
Saved to file: output/listawysylkowa.tex
Saved to file: output/main.tex
Saved to file: output/pocztex.tex
Saved to file: output/protokolprzekazania.tex

6. Wywoływanie kompilatora Latex dla głównego pliku *main.tex*, co spowoduje skompilowanie wszystkich szablonów:

Executing shell command:
cmd /c start texlive\2010min\bin\win32\pdflatex.exe --output-directory
output/main.texExecuting shell command:
cmd /c start texlive\2010min\bin\win32\pdflatex.exe --output-directory
output/main.tex

WORK DONE!!!

Directory pdf results output: output/

Program wykonał się bez żadnych błędów i wykonał swoje zadanie. Uzupełnienie szablonów trwało niżej 1 sekundy, natomiast podwójna kompilacja wszystkich szablonów około jednej minuty, co spełnia założenia projektu.

4.6. Wyniki testu

Wszystkie szablony zostały uzupełnione poprawnie danymi z bazy danych oraz kompilacja systemu Latex przebiegła bezproblemowo. W pliku wynikowym PDF ze wszystkimi dokumentami gotowymi do druku wszystko jest zgodnie z oczekiwaniami. Wygenerowany plik PDF z testów dołączony jest w załączniku.

5. Instrukcja obsługi

a

5.1. Wymagania systemowe

5.2. Przygotowanie pliku konfiguracyjnego

5.3. Przygotowanie prostego szablonu

5.4. Dodatkowe polecenia w szablonach

5.4.1. Polecenie puste

5.4.2. Grupowanie

5.5. Uruchomienie programu

5.6. Rozwiązywanie problemów

6. Podsumowanie

Bibliografia