Baza danych – MPK

Wojciech Franczuk

I. Projekt koncepcji, założenia

1. Temat projektu

Celem projektu było napisanie bazy danych relacyjnej umożliwiającej zarządzanie Miejskim Przedsiębiorstwem Komunikacyjnym, oraz aplikacji do sterowania tą bazą danych. Z tego powodu, większość danych zamieszczonych w bazie to informacje o pracownikach, głównie kierowcach, oraz pojazdach, zajezdniach, czy też grafik rozkładu pracy pracowników. W bazie mpk nie może jednak zabraknąć informacji o przystankach, o połączeniach między przystankami oraz o liniach aktualnie kursujących. Potrzebna jest więc rozbudowana baza z dobrze przemyślanym diagramem ERD.

2. Analiza wymagań użytkownika

Pomoc w zarządzaniu przedsiębiorstwem ma się brać m.in. z możliwości automatycznego przydzielania zmian dla kierowców, wyznaczania pensji pracowników oraz wyznaczania czasu podróży całą linią na podstawie czasów potrzebnych na podróż pomiędzy poszczególnymi przystankami. Dodatkowo, wszystkie wyżej wymienione informacje są łatwo dostępne z poziomu aplikacji z przyjaznym dla użytkownika interfejsem. Z aplikacji użytkownik-admin ma też możliwość dodawania danych, modyfikowania istniejących w niej danych oraz usuwania ich.

II. Projekt diagramów (konceptualny)

1. Budowa i analiza diagramu przepływu danych DFD:

Przepływ danych w projekcie jest bardzo prosty. Aplikacja napisana w Javie łączy się z serwerem Pascal za pomocą tunelu ssh, następnie łączy się z bazą danych w PSQL, zawartą na tym serwerze. Informacje z bazy danych przesyłane są spowrotem do aplikacji użytkownika, w której to są przetwarzane i pokazywane w aplikacji w czytelnej dla użytkownika formie. Jakiekolwiek informacje lub polecenia wpisywane przez użytkownika wykorzystują uzyskane połączenie do wysyłania wiadomości do bazy i wyświetlania otrzymanych z powrotem danych na ekran.

2. Definiowanie encji oraz ich atrybutów

W każdej tabeli będe potrzebował klucza głównego, pochodzącego albo z tej tabeli, albo będącego kombinacją kluczy zewnętrznych. W mojej bazie danych muszę stworzyć następujące tabele:

-**Pracownicy** - tabela w której będe przechowywał informacje o wszystkich pracownikach przedsiębiorstwa. Będą w niej przechowywane oczywiste atrybuty dla ludzi, jak imię czy nazwisko, ale także stanowisko jakie pełni on w firmie, pensja oraz informacja o tym, kto jest jego przełożonym.

-**Kierowcy** – relacja ta jest rozszerzeniem relacji pracownicy. Będe w niej przechowywał dodatkowe informacje na temat pracowników ze stanowiskiem „Kierowca”, takie jak to, czy mogą prowadzić tramwaj, autobus czy oba, oraz wiadomość o tym jaki pojazd prowadzą.

-**Pojazdy –** ta relacja będzie przechowywała informację o tym w jakiej zajezdni pojazd będzie przetrzymywany w nocy, oraz o tym jaki jest to pojazd.

-**Modele** – ta tabela będzie przechowywać informację o poszczególnych modelach. Czy jest to model tramwaju czy autobusu? Jaka firma produkuje ten model i wreszcie jaka jest jego nazwa.

-**Zajezdnie –** ta tabele będzie przechowywać informację o pojemości oraz adresie

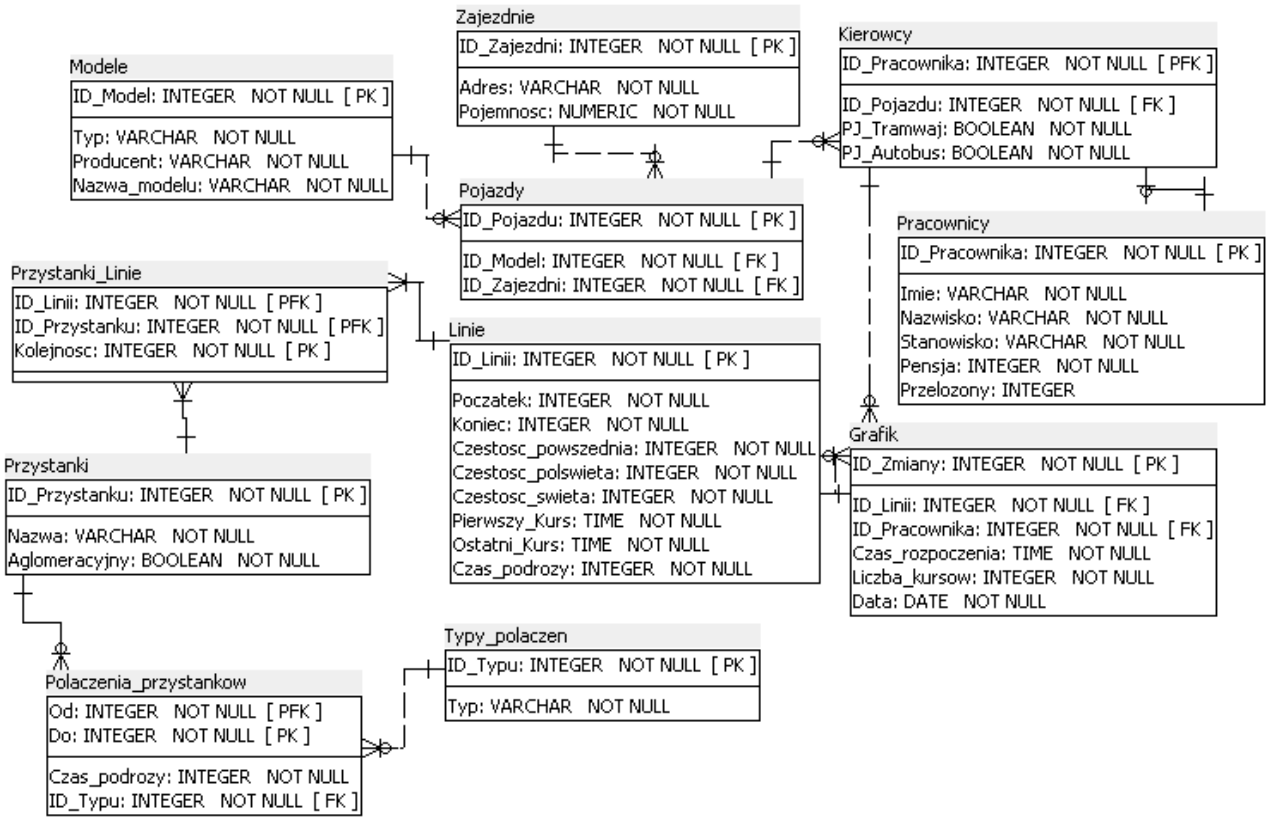
poszczególnych zajezdni.

-**Grafik –** w tej tabelii będe przechowywał informację o tym w jaki dzień pracuje jaki kierowca, jak długo oraz na jakiej linii.

-**Linie** – relacja ta będzie przechowywać informację o przystankach początkowych i końcowych danej linii, czasach w jakich ta linia kursuje, częstości w jakich ta linia kursuje oraz ogólnego czasu podróży tą linią.

-**Przystanki –** Informacje o przystankach i ich nazwach. Jako że będą połączone z liniami relacją wiele-do-wielu, trzeba będzie stworzyć tabele pośrednią między nimi.

-**Przystanki_polaczenia** – informacje o połączeniach między przystankami, czy istnieją, jakie są (np. czy mogą je pokonać tramwaje, czy autobusy czy oba, do tego można by wykorzystać tablicę słownikową) oraz jak długo zajmuje przejeżdzanie między nimi.



Rys 1. Diagram ERD projektu.

III. Projekt logiczny

1. Tworzenie projektu w języku SQL:

Plik automatycznie wygenerowany w programie SQL power architect jest załączony w projekcie pod nazwą Create_MPK.sql. Diagram ERD widoczny jest na rysunku powyżej. Dokładny opis techniczy znajduje się w punkcie drugim. Wszystkie tabele znajdują się w schemacie MPK na pascalu w bazie danych u5franczuk. Automatycznie wygenerowane zostały też sekwencje do wstawiania id do poszczególnych tabel.

2. Tabele i typy danych

Domyślnie wszystkie atrybuty są NOT NULL, chyba że napisane jest inaczej.

Domyślnie wszystkie pola typu INTEGER mają nałożone ograniczenie za pomocą domeny nie_ujemne, która nie pozwala na wprowadzanie do nich wartości ujemnych.

- Pracownicy – Tabela przechowywująca podstawowe informacje o pracownikach:

ID_pracownika pole typu INTEGER, automatycznie ustalane na podstawie sekwencji przy dodawaniu danych.

Imie – Imie pracownika, pole typu VARCHAR ograniczone domeną od_dużej wymagającą rozpoczęcia słowa od dużej litery.

Nazwisko – nazwisko pracownika, pole typu VARCHAR, ograniczone domeną od_dużej.

Stanowisko – nazwa stanowiska, które wykonuje pracownik, pole typu VARCHAR.

Pensja – pole typu INTEGER przechowywujące aktualną pensję pracownika w groszach. Dla kierowców zależy od liczby przejechanych godzin.

Przelozony – pole typu INTEGER, przechowuje ID_Pracownika, który jest przełożonym danego pracownika. Może być NULL, na przykład dla kierownika.

- Kierowcy – Tabela przechowywująca dodatkowe informacje na temat pracowników o stanowisku Kierowca:

ID_pracownika pole typu INTEGER, klucz obcy z tabeli Pracownicy. Tabele te połączone są relacją 1-do-1 za pomocą tego klucza.

ID_pojazdu – Pole typu INTEGER, klucz obcy z tabeli Pojazdy. Tabele te połączone są relacją 1-do-wiele za pomocą tego klucza.

PJ_Tramwaj – pole typu BOOLEAN, które przechowuje informacje o tym, czy dany kierowca posiada uprawnienia do prowadzenia tramwaju.

PJ_autobus – pole typu BOOLEAN, które przechowuje informacje o tym, czy dany kierowca posiada prawo jazdy na autobus.

- Pojazdy – Tabela przechowywująca informacje o pojazdach:

ID_Pojazdu pole typu INTEGER, klucz główny tabeli pojazdy. Automatycznie ustalane na podstawie sekwencji.

ID_Model pole typu INTEGER, klucz obcy z tabeli Modele. Tabele te połączone są relacją 1-do-wiele za pomocą tego klucza.

ID_zajezdni – Pole typu INTEGER, klucz obcy z tabeli Zajezdnie. Tabele te połączone są relacją 1-do-wiele za pomocą tego klucza.

- Zajezdnie – Tabela przechowywująca informacje o zajezdniach:

ID_zajezdni pole typu INTEGER, klucz główny tabeli. Automatycznie ustalane na podstawie sekwencji.

ID_pojazdu – Pole typu INTEGER, klucz obcy z tabeli Pojazdy. Tabele te połączone są relacją 1-do-wiele za pomocą tego klucza.

Pojemnosc – pole typu INTEGER, które przechowuje informacje o ilości pojazdów jakie można składować w danej zajezdni.

- Modele – Tabela przechowywująca informacje o zajezdniach:

ID_Model pole typu INTEGER, klucz główny tabeli. Automaycznie wyznaczany za pomocą sekwencji.

Typ – Pole typu VARHAR, nazwa określająca czy jest to model Tramwaju czy Autobusu.

Producent – pole typu VARCHAR, przechowuję nazwę firmy produkującą dany model.

Nazwa modelu – pole typu VARCHAR, przechowuje nazwę modelu pojazdu.

- Grafik – Tabela przechowywująca informacje o czasie pracy pracowników:

ID_Zmiany pole typu INTEGER, klucz główny tabeli. Automaycznie wyznaczany za pomocą sekwencji.

ID_linii – Pole typu INTEGER, klucz obcy z tabeli Linie. Tabele te połączone są relacją 1-do-wiele.

ID_pracownika – pole typu INTEGER, klucz obcy z tabeli pracownicy. Tabele te połączone są relacją 1-do-wiele.

Czas_rozpoczecia – pole typu TIME, przechowuje informacje o czasie rozpoczęcia zmiany.

Liczba kursów – pole typu INTEGER, przechowuje informację i liczbie kursów w obie strony jakie kierowca wykonał/ma wykonać w czasie danej zmiany.

Data – pole typu DATE, przechowuje informacje o tym w jaki dzień odbyła się dana zmiana.

- Linie – Tabela przechowywująca informacje o zmianach pracowników:

ID_Linii - pole typu INTEGER, klucz główny tabeli. Automaycznie wyznaczany za pomocą sekwencji.

Początek – Pole typu INTEGER, przechowuje ID_przystanku, na którym rozpoczyna się ta linia.

Koniec – pole typu INTEGER, przechowuje ID_przystanku, na którym kończy się ta linia.

Częstość_powszednia – pole typu INTEGER, przechowuje informacje o tym jak często kursuje linia w dni powszednie [poniedziałek; piątek]. Jednostka czasu to minuta

Częstość_półświęta – pole typu INTEGER, przechowuje informacje o tym jak często kursuje linia w soboty. Jednostka czasu to minuta.

Częstość_święta – pole typu INTEGER, przechowuje informacje o tym jak często kursuje linia w niedziele. Jednostka czasu to minuta.

Pierwszy_kurs – pole typu TIME, przechowuje informacje o której godzinie wyjeżdża pierwszy kurs.

Ostatni_kurs – pole typu TIME, przechowuje informacje o której godzinie najpóźniej wyjeżdża ostatni kurs.

Czas_podrozy – pole typu INTEGER, przechowuje informacje o sumarycznym czasie podróży między przystankami. Uzupełniane automatycznie na podstawie informacji w widoku polaczenia_pomocnicze.

- Przystanki_linie – tabela łącząca tabele przystanki oraz tabelę Linie, unikając w ten sposób połączenia wiele-do-wielu:

ID_Linii - pole typu INTEGER, klucz główny tabeli. Klucz obcy z tabeli Linie, łączący je relacją jeden-do-wielu. Jest częścią klucza głownego tej tabeli.

ID_Przystanku – Pole typu INTEGER, klucz obcy z tabeli Przystanki. Tabele te połączone są relacją 1-do-wiele. Wchodzi w skład klucza głownego tej tabeli.

Kolejnosc – pole typu INTEGER, przechowuje informację o tym, który w kolejności jest dany przystanek na linii określonej w ID_linii. Wchodzi w skład klucza głównego tej tabelii.

- Przystanki – przechowująca informację o poszczególnych przystankach:

ID_Przystanku - pole typu INTEGER, klucz główny tabeli. Automaycznie wyznaczany za pomocą sekwencji.

Nazwa – Pole typu VARCHAR, przechowuje nazwe danego przystanku.

Aglomeracyjny – pole typu BOOLEAN, przechowuje informację o czy dany przystanek jest aglomeracyjny.

- Polaczenia_przystanków – przechowuje informacje o połączeniu między dwoma przystankami:

Od - pole typu INTEGER, klucz obcy z tabeli Przystanki. Przechowuje id_przystanku od którego zaczyna się połączenie. Wchodzi w skład klucza głównego tabeli.

Do – Pole typu INTEGER, klucz obcy z tabeli Przystanki. Przechowuje id_przystanku, na którym kończy się połączenie. Wchodzi w skład klucza głownego tabeli.

Czas_podróży – pole typu INTEGER, przechowuje informacje o czasie jaki potrzebuje pojazd na przejazd między dwoma przystankami. Jednostka czasu to minuta.

ID_typu – pole typu INTEGER, przechowuje informacje o czasie jaki potrzebuje pojazd na przejazd między dwoma przystankami. Jednostka czasu to minuta. Klucz obcy z tabeli słownikowej

Typy_połączeń

- Typy_polaczen – tabela słownikowa przechowywująca informację o typach połączeń:

ID_Typu - pole typu INTEGER, klucz główny tabeli.. Automatycznie ustawiany na podstawie sekwencji.

Typ – Pole typu VARCHAR, przechowuje informację o nazwie danego typu połączenia.

3. Widoki

W celu bardziej czytelnego dostępu do danych, stworzyłem widoki niektórych tabel, które składają się z dużej ilości kluczy, które są trudno rozszyfrowywalne przez niewtajemniczego użytkownika. W tym celu powstały cztery widoki:

1. CREATE VIEW polaczenia_przystankow_czytelne AS
SELECT p.nazwa AS od, p2.nazwa AS do, pp.czas_podrozy, tp.typ
FROM przystanki p, przystanki p2, polaczenia_przystankow pp, typy_polaczen tp
WHERE p.id_przystanku=pp.od
AND p2.id_przystanku=pp.do_1
AND tp.id_typu=pp.id_typu
ORDER BY od;

Ten widok umożliwia zobaczenie tabeli polaczenia_przystanków w dużo bardziej czytelnej formie. Kolumny Od oraz do, przechowuwujące id_przystanku w postaci liczby całkowitej, zostały zamienione przez nazwy przystanków odpowiadające tym wartością ID. Podobnie kolumna id_typu została zamieniona na nazwe typu z tabeli typy_polaczen. Posortowane według nazwy przystanku.

2. CREATE VIEW pojazdy_czytelne AS
select p.id_pojazdu, m.typ, m.producent AS marka, m.nazwa_modelu AS model, z.adres
from pojazdy p, zajezdnie z, modele m
WHERE z.id_zajezdni=p.id_zajezdni
AND m.id_model=p.id_model
ORDER BY id_pojazdu;

Ten widok pokazuje tabelę pojazdy, ale zamiast id_modelu wyświetla odpowiednie informacje z tabeli Modele, odpowiadające tej wartości id. Zamiast ID-zajezdni, wyświetla się adres tej zajezdni. Posortowane według id_pojazdu

3. CREATE OR REPLACE VIEW kierowcy_czytelne AS
Select k.id_pracownika, p.imie, p.nazwisko, poj.typ, k.PJ_autobus, k.PJ_tramwaj, p.pensja
FROM pracownicy p, kierowcy k, pojazdy_czytelne poj
WHERE k.id_pracownika=p.id_pracownika
AND poj.id_pojazdu=k.id_pojazdu
ORDER BY id_pracownika;

Ten widok ułatwia sprawdzanie typu pojazdu dla kierowcy. Wywietla wszystkie wartości z tabeli Kierowca, zastępując kolumne ID_pojazdu typem. Posortowane według ID_pracownika.

4. CREATE VIEW przystanki_linie_czytelne AS
SELECT pl.id_linii, p.nazwa, pl.kolejnosc
FROM przystanki_linie pl, przystanki p
WHERE pl.id_przystanku=p.id_przystanku
ORDER BY id_linii, kolejnosc;

Ten widok umożliwia zobaczenie kolejności przystanków dla każdej linii, gdzie id_przystanku zastąpione jest jego nazwą. Dodatkowo jest posortowane według linii oraz kolejności, co sprawia że

łatwo można odczytać przystanki w kolejności ich występowania na trasie linii.

Dodatkowo stworzyłem jeszcze dwa widoki, ułatwiające liczenie czasu podróży na całej linii

5. CREATE VIEW polaczenia_pomocnicze AS
SELECT pl.id_linii, pl.id_przystanku, AS od, pl2.id_przystanku AS do, pl.kolejnosc
from przystanki_linie pl, przystanki_linie pl2
WHERE pl.id_linii = pl2.id_linii
AND pl2.kolejnosc=pl.kolejnosc+1;

6. CREATE VIEW do_liczenia_podrozy AS
SELECT pl.id_linii, pl.od, pl.do, pp.czas_podrozy, pl.kolejnosc
FROM polaczenia_przystankow pp, polaczenia_pomocnicze pl
WHERE (pl.od=pp.od AND pl.do=pp.do_1)
OR (pl.od=pp.do_1 AND pl.do=pp.od);

Pokazują one połączenia między przystankami w kolejności występowania tych przystanków na trasie linii. Dzięki temu moge łatwo dodać wartośći czasu podróży i na bieżąco aktualizować odpowiednie pole w tabelii Linie.

4. Wyzwalacze

W celu jeszcze lepszej kontroli wprowadzanych danych, zdecydowałem się utworzyć kilka wyzwalaczy, kotrolujących wprowadzane dane. Kod tych wyzwalaczy zamieszczony jest w pliku wyzwalacze.sql załączonym w projekcie. Krótki ich opis zamieszczony jest poniżej.

1. Zajezdnie:

Ten wyzwalacz służy sprawdzaniu czy dodanie do tabeli Pojazdy kolejnego pojazdu nie sprawi, że zajezdnia będzie przepełniona. W razie gdy dodanie pojazdy do zajezdni przeproczyłoby wartość kolumny pojemność dla tej zajezdni, zostanie rzucony wyjątek i rekord nie zostanie dodany.

2. Przelozony:

Ten wyzwalacz sprawdza, przy każdym dodawaniu pracownika do tabeli Pracownicy, czy stanowisko pracownika, którego id_pracownika wpisano do pozycji Przełożony to kierownik. Jeśli nie, nie ma możliwości dodania takiego rekordu do tabeli i rzucany jest wyjątek.

3. Dziury w liniach:

Ten wyzwalacz uniemożliwia usuwanie oraz uaktualnianie kolejności przystanków w linii w sposób, który sprawił by, że w kolejności powstałaby „dziura”. Sprawdza też, czy nowo dodawany przystanek do linii, dodawany jest na koniec. Jeżeli nie, rzuca wyjatek i uniemożliwia dodanie rekordu.

4. Uzupelnij_czas:

Ten wyzwalacz na bieżąco uaktualnia czas podróży linią, kiedy dodajemy do niej kolejny przystanek. Dzięki niemu, nie ma potrzeby ręcznego wstawiania danych do kolumny czas_podrozy w liniach. Korzysta w tym celu z pomocniczych widoków opisanych powyżej oraz z funkcji Licz_czas_przejazdu opisanej w podpunkcie „Funkcje”.

5. sprawdz_polaczenia:

Ten wyzwalacz sprawdza czy przy dodawaniu do tabeli przystanki_linie istnieje w bazie informacja o połączeniu tych dwóch przystanków w bazie polaczenia_przystanków. Korzysta w tym celu z widoków pomocniczych i funkcji licz_czas_przejazdu, opisanej w podpunkcie „Funkcje”.

6. maksymalny_czas_pracy

Ta funkcja upewnia się, że nie wstawiamy do tabeli Grafik zmiany, która sprawiłaby, że kierowca pracowałby dłużej niż osiem godzin dziennie. Nie jest to bezpieczne dla pasażerów, ani dla budżetu przedsiębiorstwa.

7. Auto_kierowcy:

Ten wyzwalacz sprawdza przy każdym dodawaniu pracownika do tabeli pracownicy czy stanowisko nowo dodanego pracownika to „Kierowca". Jeśli tak, automatycznie tworzy wpis do tabeli Kierowcy z ID_pracownika tego pracownika.

8. uaktualnianie_pensji

Ten trigger automatycznie wypełnia pole „pensja" w tabeli pracownicy za każdym razem gdy pojawi się nowy lub zmieni się aktualny wpis w tabeli Grafik. Pensja każdego pracownika zależy od ilości przejechanych przez niego godzin.

5. Funkcje

Kolejnym bardzo ważnym elementem mojego projektu były funkcje pozwalające na automatyczne przydzielenie danych, których wpisywanie ręczne byłoby pracą na pełny etet, na który nasza firma nie może sobie pozwolić. Dzięki nim pola takie jak Grafik oraz id_pojazdu w tabeli Kierowcy wypełniane są automatycznie i poprawnie(tzn. np. kierowca bez prawa jazdy na autobus nie dostanie autobusu, tylko tramwaj, jeśli ma uprawnienia na autobus). Wszystkie funkcje zawarte są w pliku functons.sql.

1. Licz_czas_przejazdu:

Funkcja wspomniana już wcześniej. Wyznacza czas przejazdu daną linią na podstawie czasu przejazdu pomiędzy wszystkimi przystankami w tej linii. Korzystają z niej dwa wyzwalacze i bardzo ułatwia wprowadzanie do systemu zmian, poprzez automatyczne uzupełnianie danych.

2. Liczenie_liczby_kursów

Funkcja wyznaczająca ile kursów musi wykonać przekazana w argumencie linia w przekazanym w dniu argumencie, aby zaspokoić zapotrzebowanie wyznaczone w tabeli Linie.

3. Przydziel_pojazdy

Ta funkcja korzysta z kursorów w celu wyznaczenia optymalnego rozłożenia pojazdów dla kierowców. Dzięki niej nie ma potrzeby mozolnego przypisywania pojazdu każdemu pracownikowi i do tego zapewnia bezbłędne działanie.

4. Generuj_grafik

Pieszczotliwie nazywana „arcydziełem kierownika", ta funkcja wyznacza grafik na podany jako argument dzień w optymalny sposób, to znaczy korzystając z możliwie najmniejszej liczby kierowców dostępnych w bazie danych. Przydziela pracownikom jak najdłuższe zmiany, zawsze jednak mniejsze niż osiem godzin. Ponownie, zaoszczędza bardzo dużo mozolnego wpisywania danych i do tego w bardzo optymalny sposób. W połączeniu z wyzwalaczem uaktualnienie pensji, właściwie sprawia, że kierownik cały czas ma wolne.

IV. Projekt funkcjonalny

1. Interfejs użytkownika

Zdecydowałem się napisać cały interfejs użytkownika w Javie. Chciałem stworzyć aplikację która umożliwiłaby mi dostęp do najważniejszych funkcji bazy danych, w tym lepszego sposobu na oglądanie danych. Zdecydowałem się skorzystać z aplikacji SceneBuilder do wizualnego budowania aplikacji, a do logicznej części użyłem programu Netbeans. Pomimo tego że jest to oficjalny program od Oracle, wydawców Javy, to dalej nie jest dostosowany do wersji Javy 9.

Prawdopodobnie z tego powodu, wyświetlanie tabeli Pracownicy (i tylko tej jednej tabeli) powoduje czasami rzucanie wyjątków. Niestety nie byłem w stanie znaleźć przyczyny tego problemu i wyświetlanie tej jednej tabeli w aplikacji czasami kończy się błędem.

Program otrzymuje połączenie z bazą danych poprzez stworzenie tunelu na taurusa a następnie zalogowanie się do bazy danych u5franczuk na Pascalu, automatycznie ustawiając search_path na schemat mpk. Z tego powodu uruchamianie aplikacji może trwać nawet do kilku minut, w zależności od jakości połączenia internetowego z serwisem Pascal.

Projekt składa się z czterech klas:

MPK_PSQL – klasa główna inicjująca aplikację

FXMLDocumentController – Klasa zajmująca się logiką aplikacji. To w niej występuje obsługa wszystkich wydarzeń które mogą zajść podczas obsługi aplikacji przez użytkownika. W tej klasie występują też wszystkie metody do pobierania i wysyłania danych do bazy danych.

PascalConnection – Ta klasa, korzystając z biblioteki JSCH tworzy tunel do sieci wydziałowej Pascal oraz połączenie do bazy danych w tym serwisie.

Polaczenie – klasa przechowywująca zmienną typu Connection, z której korzysta każda metoda klasy FXMLDocumentController, która łączy się z serwerem.

Interfejs użytkownika składa się z paneli i jest podzielony przez nie na część prawą i lewą. W zależności od opcji wybranej przez użytkownika, różne AnchorPane'y przyjmują wartość pola isVisible true. Dzięki czemu na jednej jest możliwe zarówno wyświetlanie jak i dodawanie danych.

Panel po lewej stronie służy do wyświetlania informacji o wykonywanych poleceniach oraz do wpisywania opcji wyświetlania, natomiast prawy panel służy do wyświetlania bądź wstawiania informacji do bazy danych. W górnej części aplikacji znajduje się pasek menu, z czterema pozycjami. Pierwsza z nich, „Tabele", służy do wyświetlania wszyskich tabel znajdujących się w bazie danych. Po wybraniu opcji „wyświetlającej", czyli Tabele albo Widoki, pojawia się możliwość dodania własnej kwerendy na koniec wyrażenia „select * from [table]". Będzie ona wykorzystana przy następnej próbie wyświetlenia tabeli. Jeśli pole to jest puste, program korzysta z domyślnych kwerend GROUP BY. Po kolei:

Pracownicy – brak klauzuli, zawsze powoduje błędy, prawdopodobnie związane z ww. błędem Javy. Nie można też dodawać własnych kwerend. Ta opcja wyświetlenia istnieje w tabeli tylko z nadzieją że może ten błąd w JavieFX kiedyś zostanie naprawiony i wtedy wszystko będzie działać poprawnie.

Kierowcy – id_pracownika.
Pojazdy - id_pojazdu
Zajezdnie - id_zajezdni
Modele - id_model
Grafik – id_linii, czas_rozpoczecia, data
Linie - id_linii
Przystanki_linie - id_przystanku
Przystanki - id_przystanku
Polaczenia_przystankow - od
Typy_polaczen – id_typu

Wyświetlanie tych danych odbywa się w TableView, specjalnej klasie JavyFX do wyświetlania tabel. Pobieranie danych z serwera i wstawianie ich do tabeli odbywa się dynamicznie za pomocą metody stworzonej przezemnie o nazwie buildData(String statement) zawartej w klasie FXMLDocumentController. Dane automatycznie przedstawione są w przyjazny dla użytkownika sposób. Przy każdym nowym pobraniu danych, zarówno dane na ekranie jak i w tabeli w pamięci programu są kasowane.

Kolejna pozycja menu to „Widoki". Jak nazwa sama na to wskazuje, opcje w tym rozwijanym menu pozwalają na wyświetlenie wszystkich widoków zawartych w bazie danych. Wyświetlanie danych z widoków odbywa się dokładnie tak samo jak wyświetlanie danych z tabel. Opcje w menu odnoszą się do następujących widoków, opisanych dokładnie w punkcie III:

Przystanki w liniach: przystanki_linie_czytelne,
Kierowcy i ich pojazdy: kierowcy_czytelne,
Pojazdy informacje: pojazdy_czytelne,

Do liczenia czasu podróży: do_liczenia_podrozy,
Połączenia przystankow: polaczenia_przystankow_czytelne,
Dodatkowe informacje o przystankach: polaczenia_pomocnicze.

Następne menu rozwijane to „Dodawanie danych”. Służy oczywiście wstawianiu danych do tabeli. Istnieje możliwość wstawiania danych do wszystkich tabel, za wyjątkiem „Grafik” oraz „Kierowcy”. Dodawanie do tabelii Kierowcy nie ma sensu, gdyż za każdym razem gdy dodany zostaje do tabeli Pracownicy rekord z polem „Stanowisko” o wartości „Kierowca”, rekord zostaje tam dodany automatycznie. Dodawanie do tabeli Grafik obywa się korzystając z funkcji Generuj_grafik(DATE). Można to także zrobić w zakładce „Dodawanie”, po podaniu daty jaka ma zostać przekazana jako argument funkcji.

Żeby dodać dane, należy wpisać wartości w pola TextField a następnie nacisnąć przycisk insert znajdujący się obok tego wiersza danych. Po lewej stronie ekranu pojawi się informacja o powodzeniu naszego wyrażenia bądź też co poszło nie tak(komunikat wyjątku z PSQL'a) w przypadku niepowodzenia. Dane(szczególnie DATE i TIME) muszą być wprowadzane w sposób opisany przy tych polach. Poprawność wprowadzonego stringa jest sprawdzona za pomocą wyrażenia regularnego.

Ostatnim rozwijanym menu jest zakładka „Inne”. Znajdują się w niej dwie pozycje, przydziel pojazdy, które wykonuje funkcje „przydziel_pojazdy” opisaną w punkcie III.4, oraz Wykonaj. Jest to zakładka umożliwiająca wykonywanie dowolnego polecenia DELETE lub UPDATE, wpisywanego jak w konsoli PSQL'a. Nie jest to najbezpieczniejsze rozwiązanie, ktokolwiek z dostępem do aplikacji może nawet usunąć cały schemat. Zakładamy jednak że do tabeli ma dostęp tylko admin lubi kierownik, w którego jak najlepszym interesie jest nieusuwanie całego projektu. Poza tym, nazwanie kogoś „;DROP TABLE pracownicy CASCADE” prawdopodobnie też mogłoby namieszać w bazie danych, a to już było umożliwione wcześniejszymi opcjami.

W tym oknie jest możliwość wybrania czy chcemy wykonać polecenie UPDATE czy DELETE, a następnie na której tabeli chcielibyśmy je wykonać. Ułatwia to prace, bo nie trzeba całego polecenia wpisywać do konsoli.

V. Dokumentacja

1. Wprowadzanie danych.

Dane wprowadzałem do tabeli na dwa sposoby. Informacje o rzeczywistych obiektach jak np. przystanki wprowadzałem ręcznie. Są to dane w tabelach: „Przystanki”, „Zajezdnie”, „Modele”, „Przystanki_linie”, „Polaczenia_przystankow”, „Typy_polaczen” oraz „Linie”. Dane te zawarte są w pliku pop_info.sql. Pozostałe dane, takie jak Pracownicy, Kierowcy czy Pojazdy generowałem za pomocą programu napisanego w javie, których kod załączam razem z projektem w plikach generate*.java. Pracowników dla przykładu generowałem z bazy stworzonej przez siebie składającej się z około piećdziesięciu imion oraz siedemdziesięciu pięciu nazwisk. Próbowałem jak najbardziej uniknąć powtórzeń imiona z nazwiskiem. Dane do grafiku wprowadzalem za pomocą napisanej przez siebie funkcji

2. Testowanie.

Aplikacja spełnia wszystkie założenia z punku pierwszego, z drobnym potknięciem w postaci niemożliwości poprawnego wyświetlania tabeli „Pracownicy” związanego z błędem niezależnym od napisanego przezemnie kodu. Sprawdziłem też działanie aplikacji na Linuxie na sieci wydziałowej Taurus. Program działał tak samo dobrze jak na moim systemie Windows 10.

3. Dokumentacja techniczna.

Zgodnie z zaleceniami prowadzącego przedmiotu, nie załączam szczegółowej dokumentacji technicznej. Starałem się opisać działanie aplikacji jak najlepiej to możliwe w tym pdf'ie, mam

nadzieje że jest to zrobione w sposób zadowalający.

4. Uruchamianie

Aplikacje można uruchomić zwyczajnie z konsoli poleceniem java -jar MPK_PSQL.jar. Moża też uruchomić skrypt, który wpisze to polecenie za ciebie. Plik run.bat uruchamia go dla windowsów, a run.sh dla linuxów. W razie problemów z działaniem na maszynie zdalnej, można uruchomić też aplikacje na taurusie. W moim folderze domowym: /stud2015/5franczuk/ w folderze test znajdują się wszystkie te pliki które można znaleźć w folderze Aplikacja, co pozwala na uruchomienie z taurusa, na którym przetestowałem już działanie tej aplikacji.

5. Źródła.

www.stackoverflow.com – zbyt dużo tematów z rozwiązaniami problemów niż można by tu zmieścic.

www.youtube.com – masa tutoriali do SceneBuildera.

I na koniec najważniejsze źródło, bez którego nie miał bym szans nawet tknąć tego projektu. To źródło dało mi nie tylko wiedzę, ale także motywację, która w moim przypadku zwykle jest dużo bardziej cennym zasobem:

http://aurora.ftj.agh.edu.pl/~zimnoch/index.php