|  |
| --- |
| **C:\Users\patry\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LOGO HADES.PNG** |

**Zakład Pogrzebowy – Hades Travel**

Patrycja Karbownik, Michał Iwanek, 4I4

Dokumentacja Bazy Danych

WSTĘP

Przedsiębiorstwo Usługowe Zakład Pogrzebowy „Hades Travel”, później nazywane Zakładem, jest ogólnopolską siecią zakładów świadczących usługi pogrzebowe. Oddziały mieszczą się w Warszawie, Wrocławiu i Gdańsku. W najbliższej przyszłości planowane jest otworzenie kolejnych oddziałów w innych miastach Polski.

Oddziały otwarte są od poniedziałku do niedzieli, codziennie w tych samych godzinach.

W każdym z miast jest kilka cmentarzy, a jeden oddział może obsługiwać pogrzeby na kilku cmentarzach. Jednak w danym momencie na danym cmentarzu możliwy jest tylko jeden pochówek – związane jest to głównie z dostępnością kaplicy cmentarnej.

Praca w Zakładzie jest z pewnością wyczerpująca, dlatego nie obciążamy pracowników zbyt dużą ilością obowiązków. W trosce o ich dobro każdy pracownik będzie pracował w tylko jednym oddziale na jednym stanowisku. Jedynie dyrektor i jego asystentka nie mają przypisanego stałego oddziału, gdyż dyrektor zarządza całym przedsiębiorstwem, a asystentka pomaga mu w tym.

W zakładzie panuje ściśle określona hierarchia: każdy kierownik odpowiada za swoje działania przed dyrektorem, z kolei bezpośrednimi podwładnymi każdego kierownika są grabarze, dyspozytorzy

i kierowcy pracujący w danym oddziale.

Oddziałów, pracowników oraz przede wszystkim klientów ciągle przybywa. Potrzebne jest miejsce, gdzie wszystkie dane zostaną zgromadzone. Właśnie dlatego stworzyliśmy bazę danych wspomagającą nasze przedsiębiorstwo.

STRUKTURA BAZY DANYCH

Baza danych Zakładu zbudowana jest z jedenastu tabel powiązanych ze sobą relacjami różnych typów (1:1, 1:N, M:N), wśród których tylko niektóre są obligatoryjne.

Część tabel posiada klucze głównie złożone z kilku atrybutów, kilka z jednego atrybutu,  
a inne ze sztucznie wygenerowanego atrybutu typu *integer*.

Model związków encji i model relacyjny znajdują się na końcu dokumentacji.

WIDOKI

W bazie znajduje się widok *DATA\_EMPLOYEE*, który zawiera dane pracowników – PESEL, nazwisko, imię, nazwę stanowiska oraz adres i numer telefonu. Dzięki temu widokowi, gdy jest potrzeba przejrzenia danych pracowników, użytkownik nie musi wywoływać zapytania – ma gotową tabelę, gdzie w jednym miejscu w przejrzysty sposób zamieszczone są wszystkie najważniejsze informacje o danej osobie.

TRIGGERY

Z czasem w bazie danych przybywa rekordów. Rekordy w tabelach, w których kluczem głównym jest sztucznie generowany atrybut, są numerowane kolejno liczbami naturalnymi. Wykorzystywane są do tego triggery *address\_ID\_addr\_trg*, *customer\_ID\_addr\_trg*, *employee\_ID\_addr\_trg* oraz sekwencje *address\_ID\_addr\_seq*, *customer\_ID\_customer\_seq*, *employee\_ID\_employee\_seq*.

Trigger *salary\_or\_bonus\_trg* upraszcza administratorowi wprowadzanie dodatkowych rekordów w tabeli *salary\_or\_bonus*. W przypadku wprowadzania kolejnych pensji nie jest konieczne uzupełnianie atrybutu *min\_funerals* i *max\_ funerals*, które wykorzystywane są tylko dla premii.

Pochówek może mieć jeden z trzech statusów – *scheduled* - zaplanowany (realizowany), *finished* - zakończony (zrealizowany) oraz *cancelled* – odwołany. Gdy klient odwoła pogrzeb  
(może się to zdarzyć z różnych powodów), wszystkie akcesoria (trumna oraz wieniec, jeśli został zamówiony) wracają do magazynu. W tym celu został stworzony trigger *burial\_cancelled\_trg*.

Przed dodaniem zamówienia powinno zostać sprawdzone czy wybrane przez klienta trumna  
i wieniec znajdują się w magazynie. Jeśli z różnych powodów nie zostanie to sprawdzone, a wybrane zostaną artykuły niedostępne, z pomocą przychodzi nam trigger *burial\_check\_trg*, który informuje dyspozytora o niedozwolonym zamówieniu.

W przypadku widoków zazwyczaj nie możemy po prostu zaktualizować danych w nich zawartych, gdyż pochodzą one z kilku tablic. Aby umożliwić choć częściową modyfikację danych  
w widoku *DATA\_EMPOYEE* dodany został trigger *view\_update\_trg*, który jest uruchamiany  
w przypadku próby aktualizacji widoku i działa zamiast polecenia *UPDATE*. Trigger umożliwia użytkownikowi bazy danych aktualizację peselu, imienia, nazwiska, nr\_telefonu, czy wreszcie stanowiska danego pracownika. Funkcjonalność jest realizowana przez szereg poleceń *UPDATE* uruchamianych w ciele trigger’a. Jeśli stanowisko, które próbujemy przypisać pracownikowi  
nie istnieje, to zostanie wypisany stosowny komunikat o błędzie.

PROCEDURY I FUNKCJE

W wyniku różnych ustaw lub pomysłu dyrektora, może zajść potrzeba zmiany nazw oddziałów Zakładu. W razie takiej potrzeby, administrator bazy danych może użyć procedury *change\_name\_branch*. Niestety przy projektowaniu bazy danych nie zauważyliśmy, że może to być problematyczne. Nazwa oddziału jest kluczem głównym tabeli *BRANCH*, w związku z czym musimy ją zmieniać w kilku tabelach (tej oraz innych, gdzie jest ona kluczem obcym). Aby było to możliwe, konieczne jest chwilowe wyłączenie ograniczeń między tabelami. Lepszym rozwiązaniem byłoby wprowadzenie atrybutu ze sztucznie generowanym ID.

Procedura *update\_finished* została utworzona z myślą o uaktualnieniu statusu wszystkich pochówków, które do tej pory były zaplanowane. Dzięki temu, dyspozytor na bieżąco musi tylko dbać o to, by zmieniać ręcznie status każdego odwołanego pochówku.

Funkcje *demand\_for\_coffins* oraz *availability\_of\_coffins* przekazują informacje o trumnach  
w magazynie – jak duże jest ich zapotrzebowanie (na podstawie ilości zamówień) oraz czy trzeba wyprodukować kolejne (gdyż może kończyć się ich zapas).

OPTYMALIZACJA WYDAJNOŚCI

W celu zbadania wpływu indeksów na czas trwania zapytań trzy kluczowe tabele zostały wybrane do przeprowadzenia testów: Address, Person i Burial, na których przeprowadzono szereg eksperymentów z i bez indeksu.

W przypadku adresów stworzono indeks bitmapowy „address\_index2” na kolumnach  
(City, Street, Flat\_number) oraz wykonano zapytanie:

*SELECT \**

*FROM ADDRESS*

*WHERE CITY = 'Gdansk' AND STREET = 'Polna' AND FLAT\_NUMBER <10) ;*

Jak się okazało, zapytanie było na tyle proste, że przyrost czasu wykonania w zakresie  
od 100-100000 rekordów był marginalny. Jednak nawet tutaj widać pozytywny wpływ indeksów, gdyż czas wykonania zapytań przy użyciu „address\_index2” jest średnio 2x krótszy, niż w przypadku bez niego.

W przypadku osób stworzono raz kolejny indeks bitmapowy „people\_index”  
na kolumnach(Surname, Name, Phone\_Number) oraz wykonano zapytanie:

*SELECT \**

*FROM PERSON*

*WHERE SURNAME = 'Lis' AND NAME = 'Natasza'*

*ORDER BY PHONE\_NUMBER;*

W zakresie od 100-100000 rekordów widać już wyraźny wzrost czasów wykonania  
dla sytuacji bez indeksu. Tymczasem czas wykonania zapytania z indeksem jest praktycznie stały  
i dla 100000 rekordów jest już 6x mniejszy. Wyniki zgadzają się z teorią, według której indeksy bitmapowe są bardzo skuteczne dla kolumn z małą ilością różnych wartości.

Dla tabeli zawierającej pochówki stworzony został zwykły, oparty o B-Tree indeks „burial\_index” na kolumnach(ID\_customer, Burial\_price) oraz wykonano zapytanie:

*SELECT \**

*FROM BURIAL*

*WHERE ID\_customer < 24000 AND ID\_customer > 23000 AND BURIAL\_PRICE < 10000 AND BURIAL\_PRICE > 9000*

*ORDER BY ID\_customer;*

Wraz ze wzrostem liczby rekordów widać tutaj znaczny wzrost czasów wykonania zapytania bez użycia indeksu. Czasy wykonania wersji z użyciem „burial\_index” są niemal stałe i wahają się głównie ze względu na różne czasy opóźnień w bazie. Dla 102400 rekordów czas ten jest już niemal 7x mniejszy. Możemy zauważyć, że indeksy oparte na B-Tree są bardzo skuteczne dla wartości

w kolumnach, które są niemal unikalne lub przynajmniej bardzo różnorodne, a tak właśnie było zarówno w przypadku ID\_customer, jak i BURIAL\_PRICE. Obserwacja jest oczywiście zgodna z teorią.

Dla każdej tabeli oprócz średniego czasu zapytania dla danej liczby rekordów obliczone zostało także odchylenie standardowe. W przypadku pomiarów o dużym odchyleniu standardowym możemy domniemywać większą niedokładność danego pomiaru. Umieszczenie odchylenia standardowego na wykresie zbytnio zaciemniłoby obraz, dlatego postanowiliśmy go nie umieszczać.

We wszystkich przypadkach pomocniczo został wykorzystany interface JDBC umożliwiający sprawne wykonanie dużej ilości zapytań do bazy danych. Użyto tu obiektów typu PreparedStatement, które przyśpieszają zapytania przesyłane do bazy, a także utrudniają atak typu SQL injection.

Explain plan

Wyniki zwracane w Explain Planie różnią się w zależności od ilości danych i od tego czy nałożyło się index na atrybuty, czy nie.

Aby dojść do takiego wniosku, należało wykonać szereg eksperymentów.

*EXPLAIN PLAN*

*SET statement\_id = 'il\_osob\_na\_ulicy' for*

*SELECT A.Street, COUNT(P.PESEL)*

*FROM ADDRESS A JOIN PERSON P ON P.ID\_addr = A.ID\_addr*

*WHERE A.City = 'Warszawa' AND P.Sex = 'k' AND SUBSTR(P.Surname, 1, 1) = 'K' AND P.ID\_addr > 500 AND P.ID\_addr < 900*

*GROUP BY A.Street*

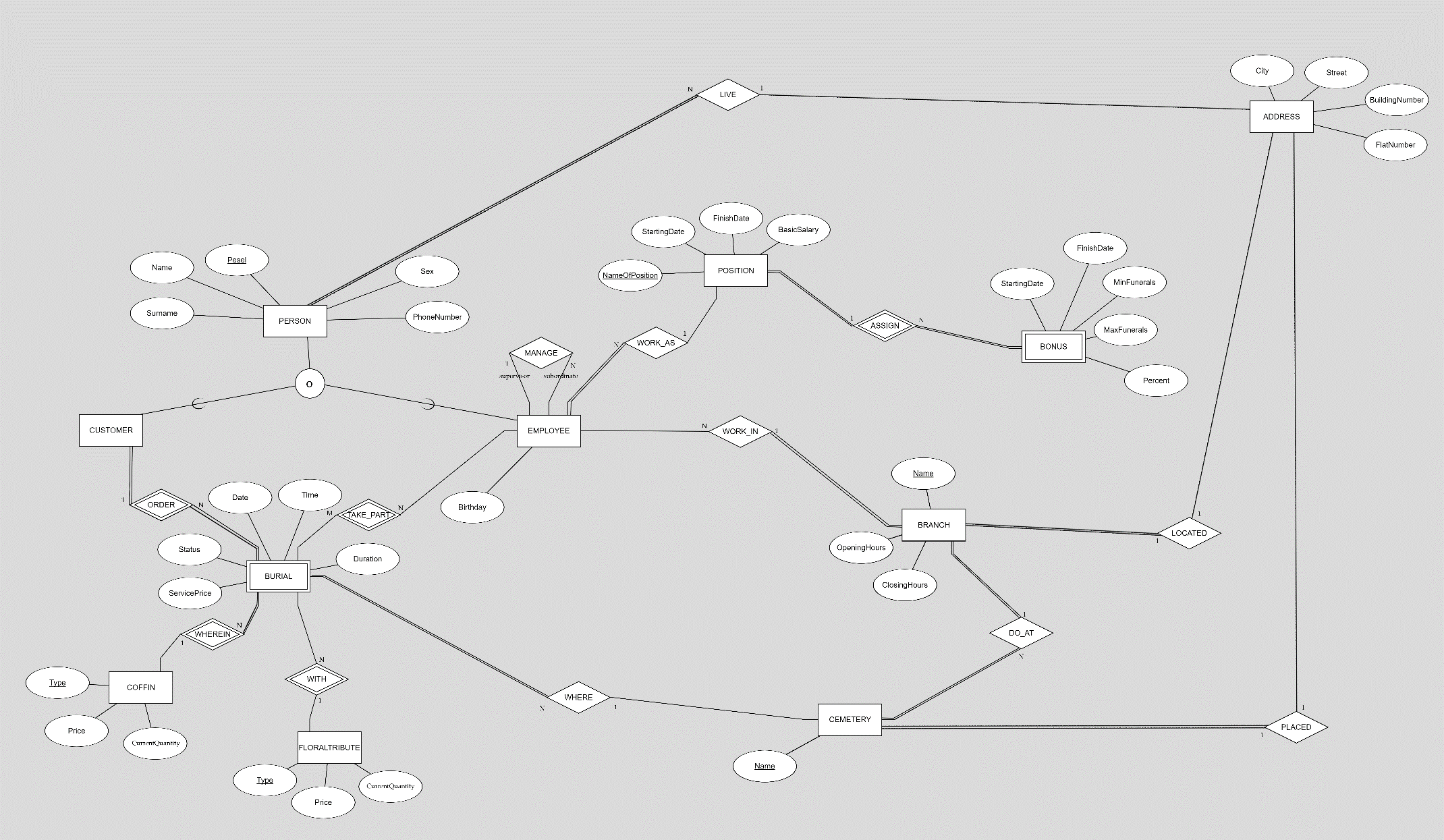
*ORDER BY A.Street;*

Nałożenie indeksu ma wpływ na wyniki zwracane przez EXPLAIN. Powyżej widzimy,  
że koszty (szacunkowa liczba odczytów I/O niezbędna w celu wykonania zapytania) mniejsze

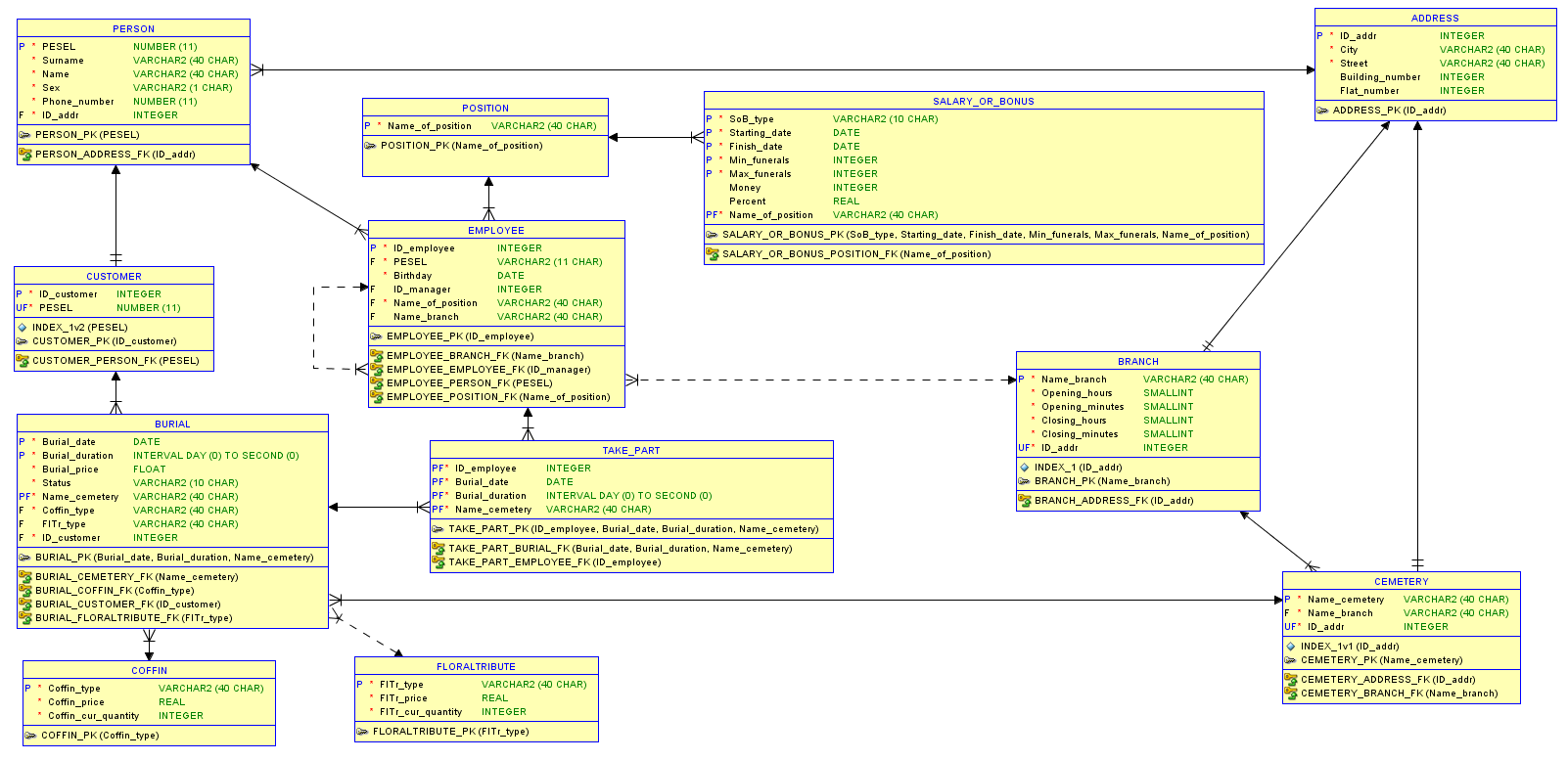
są gdy indeks zostanie nałożony.

W przypadku, gdy nie nałożymy indeksu koszt znacznie rośnie wraz z wzrostem ilości danych w tabelach. Natomiast z nałożonym indeksem koszt prawie nie ulega zmianom.

W zależności od obecności indeksu różnią się też operacje wykonywane w poszczególnych etapach realizacji zapytania.



Model związków encji



Model relacyjny