

Implementacja prostej relacyjnej bazy danych przedsiębiorstwa zarządzającego nieruchomościami za pomocą strukturalnego języka SQL

Gomółka Aleksandra

Szrajer Szymon

Nęcki Maks

5 lutego 2021



Bioinformatyka 2020/2021
Kraków

Spis treści

1	Cel projektu	2
2	Podstawowe informacje dotyczące projektu	2
2.1	Główne założenia	3
2.2	Możliwości bazy danych	3
2.3	Ograniczenia wynikające z założeń projektu	4
3	Diagramy bazy danych	5
3.1	Diagram ER bazy danych	5
3.2	Diagram relacji bazy danych	7
4	Tabele bazy danych	8
5	Elementy bazy danych	12
5.1	Indeksy	13
5.2	Procedury składowane	13
5.3	Funkcje definiowane przez użytkownika	17
5.3.1	Funkcje tabelaryczne	17
5.3.2	Funkcje skalarne	19
5.4	Widoki	20
5.5	Wyzwalacze	23
5.6	Inne elementy bazy danych	25
6	Strategia pielęgnacji i konserwacji bazy danych	27
7	Typowe zapytania	28

1 Cel projektu

Celem pracy było stworzenie relacyjnej bazy danych, która symulowałaby podstawowe działanie systemów baz zarządzających danymi zbieranymi przez przedsiębiorstwa oraz biura nieruchomości. Nasza baza danych umożliwiała przeprowadzanie wszystkich najważniejszych operacji wykonywanych na tego typu obiektach, w tym rozdział metod wykonywanych, chociażby ze względu na typ powiązania jednostki z systemem zarządzającym bazą (rozdziela administratora, zarządcę bazy danych od klienta lub osoby o niższym stopniu uprawnień) lub ze względu na rodzaj komendy którą wykonujemy (odróżnia czy chcemy uzyskać informacje z określonej tabeli czy może chcemy zmodyfikować wrażliwe dane w którejś z tabel).

2 Podstawowe informacje dotyczące projektu

Tak jak w przypadku większości projektów naukowych zajmujących się bazami danych tak też i w naszym przypadku proces tworzenia bazy można podzielić na kilka następujących po sobie części. Podstawową kwestią którą należało rozpatrzyć było zastanowienie się w jaki sposób poprawnie odwzorować wycinek rzeczywistości odpowiadający przedsiębiorstwu nieruchomości tak aby był on jak najbardziej zgodny z naszą bazą. Paradoksalnie pomimo, że ten etap pracy nie miał wiele wspólnego z faktycznym oprogramowywaniem bazy zajął on nam najwięcej czasu albowiem to od tego etapu zależy dalszy kształt i tempo działania systemu zarządzającego bazą. Dopiero jak udało się nam wyklarować jasno opisany plan relacji oraz związków pomiędzy relacjami jakie mogą zachodzić w naszej bazie to wtedy zajęliśmy się bardziej praktyczną częścią projektu czyli implementacją planu w wybranym systemie zarządzania bazą danych. Zdecydowaliśmy się na wybór produktu bazodanowego firmy Microsoft czyli resztę projektu pisaliśmy w programie Microsoft SQL

Server, głównie za pomocą języka Transact-SQL. Taką a nie inną decyzję argumentujemy głównie tym, że jest to przez nas najbardziej znane środowisko do zarządzania bazami danych więc naturalnie najprościej było nam w nim tworzyć wszelkie dodatkowe komponenty bazy.

2.1 Główne założenia

Bardziej wnikliwie możliwości i ograniczenia bazy jak i jej dodatkowe zalety wynikające z sposobu stworzenia bazy zostaną poruszone w osobnych podrozdziałach pracy.

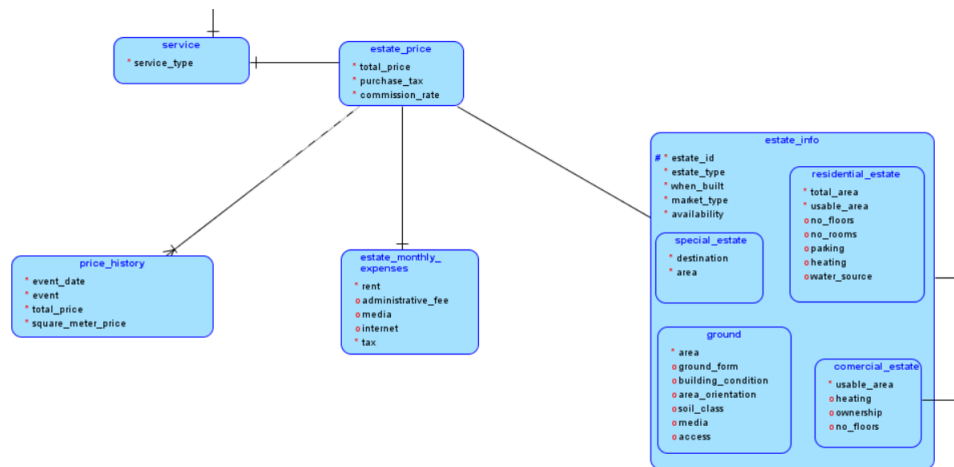
2.2 Możliwości bazy danych

Przestrzeń zagadnienia w stworzonej bazie danych stanowił dość skomplikowany i nieuporządkowany element, należało więc ją jak najbardziej uprościć tworząc relacyjny model danych. Przede wszystkim bez względu na strukturę bazy chcieliśmy zagwarantować aby model miał możliwość udzielenia odpowiedzi zwrotnej na jakiekolwiek sensowne zapytanie w ramach przestrzeni zagadnienia. Oznacza to że osoba wchodząca w interakcje z bazą (bez względu na to czy jest ona klientem czy pracownikiem biura nieruchomości) mogła w prosty sposób sprawdzić pewne interesujące ją informacje. To właśnie sprawdzenie/wyszukanie w bazie stanowi obok przeprowadzenia transakcji jeden z fundamentalnych typów działań wykonywanych na bazie. Przechodząc bardziej do konkretów dla utworzonej bazy danych oznacza to, że klient powinien mieć sposobność na wybór nieruchomości do kupna w zależności od jego preferencji. Klient (bo tak dalej będziemy nazywać przykładową osobę, która chce kupić nieruchomość z naszej bazy danych) w prosty sposób może wyselekcjonować nieruchomość w zależności od jej położenia (zarówno oznacza to położenie geograficzne tzn. w jakim państwie, mieście, dzielnicy jak i położenie względem pożądanego obiektu np. odległości od dworca kolejowego, centrum biznesowego, szkół itd.), ceny (w zależności od zakresu jaki interesuje klienta zostaną wyszukane odpowiednie nieruchomości), rynku nieruchomości (czy jest on wtórny czy jest on pierwotny) czy wyposażenia nieruchomości (jak bardzo gotowa jest do wykorzystania czy może raczej jest w stanie pustym). Skoro mówimy już o klientach i nieruchomościach warto wspomnieć o tym w jaki sposób dane dla tych kategorii są przechowywane w bazie. Dane poufne klientów przechowywane są w tabelach do których wgląd ma jedynie administrator bazy. Oprócz podstawowych informacji takich jak imię, nazwisko, pochodzenie czy wiek klienta zbierane są także bardziej szczegółowe informacje, których gromadzenie służy do lepszego dostosowywania wyszukiwanych przez klienta nieruchomości, stanowią też informacje dla agentów nieruchomości w jaki sposób poprawić prywatne konsultacje z takim klientem. Przedstawienie nieruchomości jako obiektów na których przeprowadza się takie operacje jak kupno, sprzedaż czy wyszukanie w przedstawianej bazie danych stanowiły duży problem natury technicznej. Istnieje wiele podziałów nieruchomości ze względu na pewien wybrany, określony czynnik (np. ze względu na operacje tzn. nieruchomość na sprzedaż, nieruchomość pod wynajem, ze względu na typ budynku tzn. dom wolnostojący jednorodzinny, blok, kamienica, bungalow, ze względu na typ nieruchomości tzn. nieruchomości mieszkalne, nieruchomości dla firm, grunty itd.). Rozwiązane zostało to w taki sposób, że podział nieruchomości odbył się ze względu na jeden z czterech (arbitralnie przez nas wyróżnionych) głównych typów nieruchomości. I tak dla nieruchomości mieszkalnych, nieruchomości komercyjnych, gruntów, nieruchomości specjalnych zostały stworzone osobne tabele, a informacje wspólne dla każdego z typów nieruchomości (np. dostępność, typ rynku, kiedy wybudowano) zostały zawarte w nadrzędnej tabelce po której wyżej wymienione tabele z wyróżnionymi nieruchomościami dziedziczą atrybuty. Cała metoda składowania danych dotyczących nieruchomości została w taki szczegółowy sposób opisana w pracy dlatego że ma ona kluczowy wpływ na strukturę i długość zapytań jakie są wykonywane na tym obiekcie (jakim są nieruchomości w bazie). Oprócz możliwości związanych z wyszukiwaniem nieruchomości baza posiada wgląd w pracowników zatrudnianych przez przedsiębiorstwo nieruchomości. Można sprawdzać datę zatrudnienia danego pracownika, jego stopień na tle hierarchii firmy, jego zarobki oraz (jeśli jest agentem nieruchomości) jakim klientem aktualnie się zajmuje. Oprócz tego jest możliwość aby agentów zwalniać z firmy albo promować na lepsze stanowiska. Naturalnie opisywana baza poza obsługą transakcji typu: kupno, sprzedaż, wynajem dopuszcza także tworzenie dla każdego indywidualnego klienta kalkulacji kredytowych (działa to na podobnej zasadzie jak kalkulatory kredytów dostępne w internecie - czyli pozwala na obliczenie wysokości miesięcznej raty spłaty, ilość rat, przewidywany czas

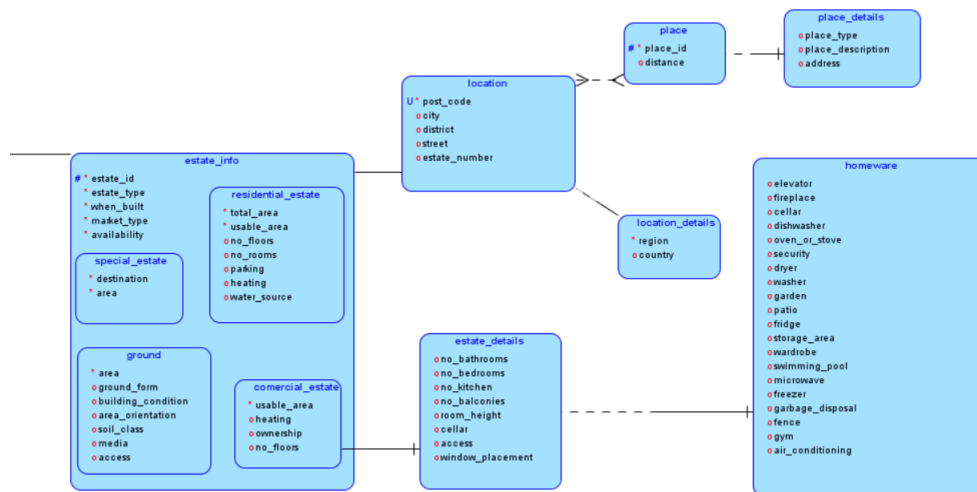
spłaty i prowizje z odsetkami dla banku udzielającego kredyt). Baza ma możliwość wykonywania na niej też wybranych, zaimplementowanych operacji statystycznych w celu zauważenia trendów na rynku nieruchomości oraz hipotetycznym określaniu dochodowości z wybranej grupy mieszkań. Tak więc można obliczyć współczynnik korelacji Pearsona dla dwóch wybranych zmiennych jakościowych opisujących w skali liczbowej wybrane nieruchomości. Co to oznacza? W skrócie możemy badać np. czy lokalizacja (albo jakakolwiek inna cecha zawarta w tabeli parameters) wpływa na cenę mieszkania (znów można wybrać jakąkolwiek inną cechę z tabeli parameters). Czyli jeszcze bardziej ogólnie mówiąc możemy testować hipotezy czy jakiś czynnik ma wpływ na inny czynnik w bazie danych. Jeśli istnieje pozytywna korelacja liniowa na jej podstawie możemy wyliczyć także linie regresji albo linie dopasowania, która jeśli stworzylibyśmy wykres (niestety w innym programie niż SQL Server) regresji pozwalałaby na szybkie i jasne określenie jak mocny występuje trend dla danej nieruchomości.

2.3 Ograniczenia wynikające z założeń projektu

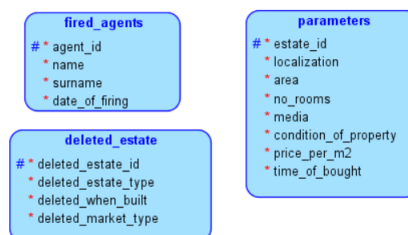
Za każdymi możliwościami i dodatkowo wykonanymi funkcjami w bazie idą pewne ograniczenia, restrykcje nałożone na skrypt bazy i uszczuplenia w jej sposobach działania. Przede wszystkim pomimo starań w jak najlepszym podziale nieruchomości na ogólne typy, nadal w bazie istnieje niebezpieczeństwo, że klient będzie chciał kupić nieruchomość, którą ciężko przyporządkować do jakiejś z czterech tabel na szczegółowe nieruchomości (albo wręcz będzie takie przyporządkowanie niemożliwe!). Ograniczeniem z kolei w wykonywaniu transakcji jest fakt że nie istnieje doszczegółowiony, jasno określony sposób w jaki sposób ma dokonać się przelanie środków do bazy nieruchomości (w końcu klient mógłby taką transakcję chcieć odłożyć w czasie albo mógłby chcieć zapłacić inną walutą, albo czekiem lub wekslem - wszystkie z wymienionych problemów są w bazie marginalizowane). Uszczupleniem w możliwości wyszukania nieruchomości mieszkalnych jest także ściśle wymieniona lista wyposażenia jakie może się znajdować w mieszkaniu. Po pierwsze przyjmuje ona wartości zero-jedynkowe to znaczy albo jakieś wyposażenie jest albo go nie ma. Nie mówi nam to nic jednak o ilości wyposażenia w mieszkaniu ani o ich stanie (klient mógłby błędnie kupić np. wille z jednym starym basenem myśląc, że kupi nieruchomość z trzema dużymi, nowymi basenami). Pewnie problem ten zostałby dość sprawnie rozwiązany jeśli albo mielibyśmy kolejną tabelę ze szczegółami nieruchomości lub tak jak rozwiązują to współczesne strony przedsiębiorstw nieruchomości - składowalibyśmy w bazie danych zdjęcia nieruchomości w które zainteresowany klient miałby wgląd. Drugim problemem z wyposażeniem jest fakt, że być może nadal brakuje jakiegoś elementu, który dla danego klienta stanowi kluczową wartość którą kieruje się kupując mieszkanie (nawiasem mówiąc ten problem występuje też w innych tabelach np. dotyczących obiektów położonych blisko wybranej nieruchomości). Kolejnym przykrym ograniczeniem jest fakt, że wiele obiektów programowalnych czyli np. niektóre procedury z listy stworzonych procedur składowych w naszej bazie danych - mogą działać tylko jednorazowo (są to albo procedury aktualizujące albo wpisujące dane do tabel lub funkcje tworzące nowe tabele). Analiza statystyczna bazy jest sztywna to znaczy chcąc wyliczyć trend w sprzedaży nieruchomości w zależności od wybranego parametru innego niż lokalizacja i cena - należałoby modyfikować kod w języku SQL istniejących zapytań. W bazie danych występują trudności z wpisywaniem nowych rekordów do tabel (sic!), które posiadają wiele więzów klucza obcego dlatego przy tworzeniu przykładowego zbioru rekordów ograniczyliśmy się do stosunkowo małej ilości danych, średnio wyliczając każda tabela zawiera po 10 wierszów atrybutów.



Część diagramu ER zajmująca się ogólnymi informacjami na temat nieruchomości takimi jak cena nieruchomości zmieniająca się w czasie, dostępność nieruchomości. Ciekawym elementem tej części diagramu jest dziedziczenie atrybutów w jednej z encji.

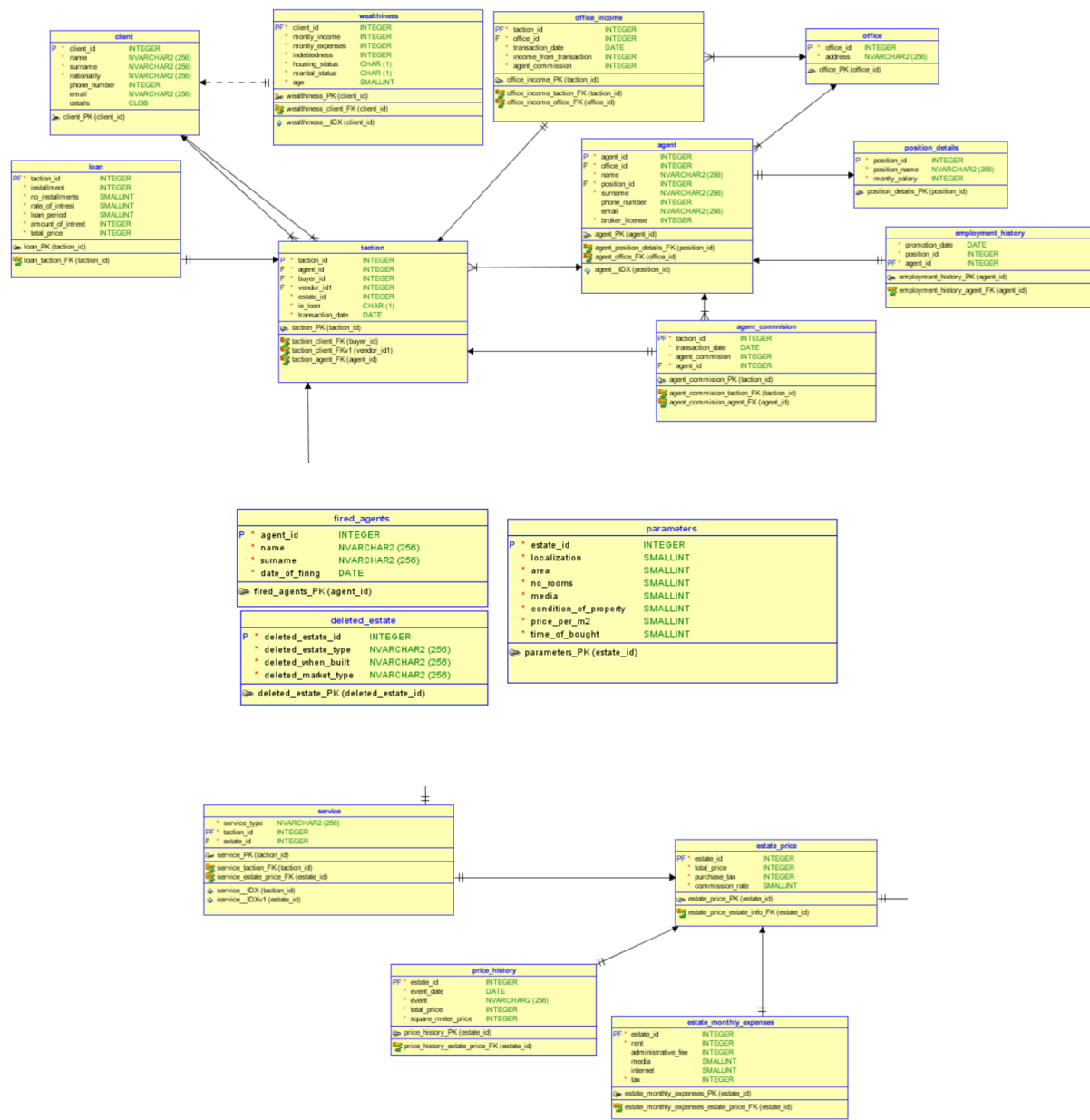


Encje przedstawiające zależności pomiędzy nieruchomościami a lokalizacją i wyposażeniem nieruchomości.



Encje nie powiązane żadnymi związkami z innymi encjami, stanowią magazyn dla danych na których będziemy już w zaimplementowanej bazie danych przeprowadzać operacje analityczne.

3.2 Diagram relacji bazy danych



4 Tabele bazy danych

Tabela przechowująca dane o klientach.


client		
P *	client_id	INTEGER
*	name	NVARCHAR2 (256)
*	surname	NVARCHAR2 (256)
*	nationality	NVARCHAR2 (256)
	phone_number	INTEGER
	email	NVARCHAR2 (256)
	details	CLOB
 client_PK (client_id)		

Tabela przechowująca dane o zamożności klientów.




wealthiness		
PF *	client_id	INTEGER
*	montly_income	INTEGER
*	montly_expenses	INTEGER
*	indebtedness	INTEGER
*	housing_status	CHAR (1)
*	marital_status	CHAR (1)
*	age	SMALLINT
 wealthiness_PK (client_id)		
 wealthiness_client_FK (client_id)		
 wealthiness__IDX (client_id)		

Tabela przechowująca dane o oddziałach biura nieruchomości.


office		
P *	office_id	INTEGER
*	address	NVARCHAR2 (256)
 office_PK (office_id)		

Tabela przechowująca dane o agentach nieruchomości. Dany agent podporządkowany jest do biura na którym zajmuje odpowiednią pozycję.

agent		
P	* agent_id	INTEGER
F	* office_id	INTEGER
	* name	NVARCHAR2 (256)
F	* position_id	INTEGER
	* surname	NVARCHAR2 (256)
	phone_number	INTEGER
	email	NVARCHAR2 (256)
	* broker_license	INTEGER
agent_PK (agent_id)		
agent_position_details_FK (position_id)		
agent_office_FK (office_id)		
agent_IDX (position_id)		

Tabele dotyczące nieruchomości. Tabele special_estate, residential_estate, comercial_estate i ground dziedziczą z tabeli estate_info.

estate_info			comercial_estate		
P	* estate_id	INTEGER	PF	* estate_id	INTEGER
	* estate_type	NVARCHAR2 (256)		* usable_area	INTEGER
	* when_built	DATE		heating	NVARCHAR2 (256)
	* market_type	NVARCHAR2 (256)		ownership	NVARCHAR2 (256)
	* availability	CHAR (1)		no_floors	SMALLINT
estate_info_PK (estate_id)			comercial_estate_PK (estate_id)		
			comercial_estate_UN (estate_id)		
			comercial_estate_estate_info_FK (estate_id)		
special_estate			ground		
PF	* estate_id	INTEGER	PF	* estate_id	INTEGER
	* destination	NVARCHAR2 (256)		* area	INTEGER
	* area	INTEGER		ground_form	NVARCHAR2 (256)
special_estate_PK (estate_id)				building_condition	NVARCHAR2 (256)
special_estate_PKv1 (estate_id)				area_orientation	NVARCHAR2 (256)
special_estate_estate_info_FK (estate_id)				soil_class	NVARCHAR2 (256)
				media	NVARCHAR2 (256)
				access	NVARCHAR2 (256)
residential_estate			ground_PK (estate_id)		
PF	* estate_id	INTEGER	ground_PKv1 (estate_id)		
	* total_area	SMALLINT	ground_estate_info_FK (estate_id)		
	* usable_area	SMALLINT			
	no_floors	SMALLINT			
	no_rooms	SMALLINT			
	parking	NVARCHAR2 (256)			
	heating	NVARCHAR2 (256)			
	water_source	NVARCHAR2 (256)			
residential_estate_PK (estate_id)					
residential_estate_PKv1 (estate_id)					
residential_estate_estate_info_FK (estate_id)					

Tabela taction zawiera dane o transakcji. Znajdują się w niej klucze obce tabeli client w postaci buyer_id i vendor_id oraz klucz obcy tabeli agent w postaci agent_id.

taction		
P	* taction_id	INTEGER
F	* agent_id	INTEGER
F	* buyer_id	INTEGER
F	* vendor_id1	INTEGER
	* estate_id	INTEGER
	* is_loan	CHAR (1)
	* transaction_date	DATE
taction_PK (taction_id)		
taction_client_FK (buyer_id)		
taction_client_FKv1 (vendor_id1)		
taction_agent_FK (agent_id)		

Tabela loan, w której rekordy tworzą się po sprecyzowaniu atrybutu is_loan w tabeli taction.

loan		
PF *	taction_id	INTEGER
*	installment	INTEGER
*	no_installments	SMALLINT
*	rate_of_intrest	SMALLINT
*	loan_period	SMALLINT
*	amount_of_intrest	INTEGER
*	total_price	INTEGER
loan_PK (taction_id)		
loan_taction_FK (taction_id)		

Tabela office_income, do której trafiają dochody biura z każdej transakcji.

office_income		
PF *	taction_id	INTEGER
F *	office_id	INTEGER
*	transaction_date	DATE
*	income_from_transaction	INTEGER
*	agent_commission	INTEGER
office_income_PK (taction_id)		
office_income_taction_FK (taction_id)		
office_income_office_FK (office_id)		

Tabela agent_commission, do której trafiają przychody agentów z każdej transakcji.

agent_commission		
PF *	taction_id	INTEGER
*	transaction_date	DATE
*	agent_commission	INTEGER
F *	agent_id	INTEGER
agent_commission_PK (taction_id)		
agent_commission_taction_FK (taction_id)		
agent_commission_agent_FK (agent_id)		

Tabela przechowująca dane dotyczące stanowisk możliwych do objęcia przez agenta.

position_details		
P *	position_id	INTEGER
*	position_name	NVARCHAR2 (256)
*	montly_salary	INTEGER
position_details_PK (position_id)		

Tabela dotycząca historii zatrudnienia agentów.

employment_history		
P *	promotion_date	DATE
*	position_id	INTEGER
PF *	agent_id	INTEGER
employment_history_PK (agent_id, promotion_date)		
employment_history_agent_FK (agent_id)		

Tabele location oraz location_details stanowiące obszerny zbiór danych o lokalizacjach.

location		
PF*	estate_id	INTEGER
U*	post_code	NVARCHAR2 (256)
	city	NVARCHAR2 (256)
	district	NVARCHAR2 (256)
	street	NVARCHAR2 (256)
	estate_number	NVARCHAR2 (256)
location_PK (estate_id)		
location_post_code_UN (post_code)		
location_estate_info_FK (estate_id)		
location__IDX (estate_id)		
location_details		
PF*	post_code	NVARCHAR2 (256)
*	region	NVARCHAR2 (256)
	country	NVARCHAR2 (256)
location_details_PK (post_code)		
location_details_location_FK (post_code)		
location_details__IDX (post_code)		

Tabele Relation_19 oraz place_details stanowiące informacje na temat obiektów w pobliżu danej lokalizacji.

Relation_19			place_details		
PF*	estate_id	INTEGER	P*	place_id	INTEGER
PF*	place_id	INTEGER		place_type	NVARCHAR2 (256)
				place_description	CLOB
				address	NVARCHAR2 (256)
Relation_19_PK (estate_id, place_id)			place_details_PK (place_id)		
Relation_19_location_FK (estate_id)					
Relation_19_place_details_FK (place_id)					

Tabele estate_details i homeware zawierające szczegółowe informacje o nieruchomościach mieszkalnych.

estate_details			homeware		
PF*	estate_id	INTEGER	PF*	estate_id	INTEGER
	no_bathrooms	SMALLINT		elevator	CHAR (1)
	no_bedrooms	SMALLINT		fireplace	CHAR (1)
	no_kitchen	SMALLINT		cellar	CHAR (1)
	no_balconies	SMALLINT		dishwasher	CHAR (1)
	room_height	SMALLINT		oven_or_stove	CHAR (1)
	cellar	CHAR (1)		security	CHAR (1)
	access	NVARCHAR2 (256)		dryer	CHAR (1)
	window_placement	NVARCHAR2 (256)		washer	CHAR (1)
estate_details_PK (estate_id)				garden	CHAR (1)
estate_details_comercial_estate_FK (estate_id)				patio	CHAR (1)
estate_details__IDX (estate_id)				fridge	CHAR (1)
				storage_area	CHAR (1)
				wardrobe	CHAR (1)
				swimming_pool	CHAR (1)
				microwave	CHAR (1)
				freezer	CHAR (1)
				garbage_disposal	CHAR (1)
				fence	CHAR (1)
				gym	CHAR (1)
				air_conditioning	CHAR (1)
			homeware_PK (estate_id)		
			homeware_estate_details_FK (estate_id)		

Tabela estate_price przechowująca dane o obecnym koszcie nieruchomości.

estate_price	
PF *	estate_id INTEGER
*	total_price INTEGER
*	purchase_tax INTEGER
*	commission_rate SMALLINT
estate_price_PK (estate_id)	
estate_price_estate_info_FK (estate_id)	

Tabela price_history zawierająca historię cen nieruchomości i estate_monthly_expenses dotycząca miesięcznych kosztów utrzymania.

price_history		estate_monthly_expenses	
PF *	estate_id INTEGER	PF *	estate_id INTEGER
*	event_date DATE	*	rent INTEGER
*	event NVARCHAR2 (256)		administrative_fee INTEGER
*	total_price INTEGER		media SMALLINT
*	square_meter_price INTEGER		internet SMALLINT
		*	tax INTEGER
price_history_PK (estate_id)		estate_monthly_expenses_PK (estate_id)	
price_history_estate_price_FK (estate_id)		estate_monthly_expenses_estate_price_FK (estate_id)	

Tabela service stanowiąca połączenie estate_price z taction.

service	
*	service_type NVARCHAR2 (256)
PF *	taction_id INTEGER
F *	estate_id INTEGER
service_PK (taction_id)	
service_taction_FK (taction_id)	
service_estate_price_FK (estate_id)	
service__IDX (taction_id)	
service__IDXv1 (estate_id)	

Tabele fired_agents, deleted_estate oraz parameters stanowiące magazyn danych dla niektórych, często używanych procedur i triggerów.

fired_agents		parameters	
P *	agent_id INTEGER	P *	estate_id INTEGER
*	name NVARCHAR2 (256)	*	localization SMALLINT
*	surname NVARCHAR2 (256)	*	area SMALLINT
*	date_of_firing DATE	*	no_rooms SMALLINT
fired_agents_PK (agent_id)		*	media SMALLINT
		*	condition_of_property SMALLINT
		*	price_per_m2 SMALLINT
		*	time_of_bought SMALLINT
deleted_estate		parameters_PK (estate_id)	
P *	deleted_estate_id INTEGER		
*	deleted_estate_type NVARCHAR2 (256)		
*	deleted_when_built NVARCHAR2 (256)		
*	deleted_market_type NVARCHAR2 (256)		
deleted_estate_PK (deleted_estate_id)			

5 Elementy bazy danych

Przedstawiana baza danych oprócz ścisłej struktury tabel i więzów kluczy obcych występujących pomiędzy tabelami posiada swoją programowalność czyli zbiór procedur składowych, funkcji definiowanych przez

użytkownika, widoków, nietypowych wyrażeń SELECT wyodrębnionych ze względu na ich zadanie i wyzwalaczy. Właściwie to co zostało wymienione powyżej stanowi wybrany przez nas obiektów, które w kolejnych częściach pracy postaramy się dokładniej wytłumaczyć (jaka jest ich budowa, funkcja oraz użyteczność w stosunku do całej bazy danych). Operując czystymi liczbami można stwierdzić, że baza danych posiada 11 procedur, 7 indeksów, 5 funkcji definiowanych przez użytkownika, 6 wyzwalaczy, 8 widoków i jedną strukturę typu CTE. Wszystkie z nich zapewniają bazie poprawne i szybkie działanie oraz możliwość wykonywania dodatkowych operacji, które normalnie byłyby niemożliwe do wykonania.

5.1 Indeksy

Wszystkie indeksy stworzone na tablicach w bazie danych mają typ nonclustered i są stworzone na polach o wartościach unikalnych (warunek nałożony przez słowo kluczowe UNIQUE) dzięki czemu oprócz tego, że zmniejszamy ilość wykorzystywanej pamięci dysku twardego podczas wykonywania działania przeszukania danej tablicy (gdyby wartości mogły się powtarzać to w systemie Microsofta do każdego powtarzającego się rekordu indeksu dodawane są cztery najty charakteryzujące dany element) zapobiegamy możliwości fragmentaryzacji indeksów (pola na których tworzyliśmy unclustered indexes w większości nie będą podlegać modyfikacji). Ogólna zasada z którą tworzyliśmy indeksy nieklastrowe polegała na staraniu się zidentyfikowaniu zapytań do odpowiednich tabel, które będą często formułowane i poprawienia ich wydajności przez nałożenie na nie indeksu. Przykładowym indeksem może być indeks na tabeli lokalizacja:

```
1 CREATE UNIQUE NONCLUSTERED INDEX
2     location__IDX ON location
3     (
4         estate_info_estate_id
5     )
6 GO
```

Sposób nazywania indeksów typu nonclustered zaczerpnęliśmy z wykładów. Indeks stworzony jest tak jak już wspominaliśmy na tabeli lokalizacja a jego identyfikatorem (czyli kolumną dzięki której przyspiesza wyszukiwanie pożądaney wartości z tabeli) jest pole: identyfikator nieruchomości.

5.2 Procedury składowane

Procedura how many available

```
1 GO
2 CREATE PROCEDURE [how many available]
3     @City nvarchar(256)
4 AS
5 BEGIN
6     SELECT COUNT(m.estate_id) [liczba dostępnych nieruchomości mieszkalnych],
7           COUNT(g.estate_id) [liczba dostępnych gruntów],
8           COUNT(k.estate_id) [liczba dostępnych nieruchomości komercyjnych],
9           COUNT(s.estate_id) [liczba dostępnych nieruchomości specjalnych]
10    FROM [location] l join estate_info i on l.estate_id=i.estate_id
11   join residential_estate m on i.estate_id=m.estate_id
12   join ground g on i.estate_id=g.estate_id
13   join comercial_estate k on i.estate_id=k.estate_id
14   join special_estate s on i.estate_id=s.estate_id
15 WHERE city = @City and m.estate_id not in (select estate_id from service)
16    and g.estate_id not in (select estate_id from service)
17    and k.estate_id not in (select estate_id from service)
18    and s.estate_id not in (select estate_id from service)
19
20 END
```

Procedura ta jako argument przyjmuje miasto dla którego chcemy wyszukać nieruchomości na których w bazie danych możemy wykonać operacje (tzn. są one dostępne na sprzedaż albo do wynajęcia).

Procedura automfill

```
1 GO
2 create proc automfill
3 AS
4 declare @col1 int, @col2 int, @col3 int, @col4 int, @col5 int,
5 @petla INT, @num int, @cena int, @czas int
6 set @petla=0
7 Set @num = 1
8 while @petla < 10
9 BEGIN
10 SET @col1=1+rand()*5
11 SET @col2=1+rand()*5
12 SET @col3=1+rand()*5
13 SET @col4=1+rand()*5
14 SET @col5=1+rand()*5
15 SET @cena=FLOOR(RAND()*(12000-6000+1))+6000
16 set @czas=FLOOR(RAND()*(10-0+1))+0
17 Insert into parameters(estate_id, localization, area, no_rooms, media, condition_of_property
, price_per_m2, time_of_bought)
18 values(@num, @col1, @col2, @col3, @col4, @col5, @cena, @czas)
19 Set @petla=@petla+1
20 Set @num=@num+1
21 END
22 GO
```

Procedura ta pozwala na automatyczne wypełnienie 10 rekordami tabeli parameters. Identyfikator nieruchomości zostanie wypełniony liczbami od 1 do 10 (w kolejności). Następne 5 pól zostanie wypełnione losowymi liczbami z zakresu od 1 do 6. Kolejne pole przyjmie wartości całkowite z zakresu od 6000 do 12000 włącznie. Ostatnie pole zostanie wypełnione losowymi wartościami całkowitymi w zakresie od 0 do 10.

Procedura loans

```
1 CREATE PROCEDURE [dbo].[loans]
2 @loan_time tinyint
3 AS
4 BEGIN
5
6 SELECT total_price, @loan_time, 1 AS [payment frequency], @loan_time/1 AS [no payments],
total_price/(@loan_time/1) FROM estate_price
7 UNION
8 SELECT total_price, @loan_time, 2 AS [payment frequency], @loan_time/2 AS [no payments],
total_price/(@loan_time/2) FROM estate_price
9 UNION
10 SELECT total_price, @loan_time, 3 AS [payment frequency], @loan_time/3 AS [no payments],
total_price/(@loan_time/3) FROM estate_price
11 UNION
12 SELECT total_price, @loan_time, 6 AS [payment frequency], @loan_time/6 AS [no payments],
total_price/(@loan_time/6) FROM estate_price
13 UNION
14 SELECT total_price, @loan_time, 12 AS [payment frequency], @loan_time/12 AS [no payments],
total_price/(@loan_time/12) FROM estate_price
15 UNION
16 SELECT total_price, @loan_time, 24 AS [payment frequency], @loan_time/24 AS [no payments],
total_price/(@loan_time/24) FROM estate_price
17
18 END
```

Procedura proponująca klientowi różne warianty spłacania kredytu na podstawie jego preferencji co do długości okresu na który chciałby wziąć kredyt.

Procedura credibility

```
1 CREATE PROCEDURE [dbo].[credibility]
2   @client int
3 AS
4 BEGIN
5
6     SET NOCOUNT ON;
7
8
9     SELECT client_id ,
10
11     CASE WHEN montly_income - montly_expenses <= 500 THEN 1
12          WHEN montly_income - montly_expenses <= 2000 AND montly_income - montly_expenses >
13             500 THEN 2
14          WHEN montly_income - montly_expenses <= 5000 AND montly_income - montly_expenses >
15             2000 THEN 3
16          WHEN montly_income - montly_expenses <= 10000 AND montly_income - montly_expenses >
17             5000 THEN 4
18          WHEN montly_income - montly_expenses <= 50000 AND montly_income - montly_expenses >
19             10000 THEN 5
20          ELSE 6
21     END AS income ,
22
23     CASE WHEN (montly_income - montly_expenses ) * 6 <= indebtedness THEN 1
24          WHEN (montly_income - montly_expenses ) * 6 > indebtedness THEN 0
25     END AS debt ,
26
27     CASE WHEN age >= 50 THEN 0
28          WHEN age < 50 THEN 1
29     END AS age ,
30     housing_status ,
31     marital_status
32 FROM wealthiness
33 WHERE client_id = @client
34 END
```

Procedura określa zdolność kredytową na podstawie miesięcznych dochodów i wydatków, stanu zadłużenia klienta, stanu mieszkaniowego i materialnego klienta oraz wieku. Zwraca tabele z wartościami: 0-6, 0-1, 0-1, 0-1, 0-1. Wartości te sumują się do 10 i na ich podstawie można proponować określone nieruchomości klientowi.

Procedura top_most_expensive_by_location

```
1 CREATE PROCEDURE [dbo].[top_most_expensive_by_location]
2   @Location nvarchar(256)
3 AS
4 BEGIN
5     SET NOCOUNT ON;
```

```

6
7 SET ROWCOUNT 10
8 SELECT A.estate_id AS TenMostExpensiveLocations, total_price
9 FROM estate_price A JOIN location B ON A.estate_id = B.estate_id
10 WHERE @Location = B.city
11 ORDER BY total_price DESC

```

Procedura pokazuje dziesięć najdroższych nieruchomości w danym mieście.

Procedura office_sales

```

1 CREATE PROCEDURE [dbo].[office_sales]
2     @Beginning_Date DateTime
3 AS
4 BEGIN
5     SET NOCOUNT ON;
6     SELECT A.office_id, MONTH(@Beginning_Date) AS "Month",
7     SUM(A.income_from_transaction - agent_commission), SUM(A.income_from_transaction),
8     SUM(A.agent_commission)
9     FROM office_income A
10    WHERE MONTH(A.transaction_date) = MONTH(@Beginning_Date) AND YEAR(A.transaction_date)=YEAR(
11        @Beginning_Date)
12    GROUP BY A.office_id
13 END

```

Procedura pobiera datę, a następnie zwraca dochody biura w miesiącu przez nią wskazanym.

Procedura Enter profits for office

```

1 CREATE PROCEDURE [Enter profits for office]
2     @TransactionID int,
3     @OfficeCommission int
4 AS
5 BEGIN
6     INSERT INTO office_income
7     SELECT T1.taction_id, T3.office_id, T1.transaction_date, @OfficeCommission, (@OfficeCommis
8         sion*0.2) AS agent_commission FROM
9     (taction T1 JOIN estate_price T2
10    ON T1.estate_id = T2.estate_id)
11    JOIN agent T3 ON T3.agent_id = T1.agent_id
12    WHERE T1.taction_id = @TransactionID
13 END

```

Procedura, która po transakcji dodaje rekord w tabeli zyski biura

Procedura Enter profits for agents

```

1 CREATE PROCEDURE [Enter profits for agents]
2     @TransactionID int
3 AS
4 BEGIN
5     INSERT INTO agent_commission
6     SELECT T2.taction_id, T2.transaction_date, T3.agent_commission, T1.agent_id FROM agent T1
7     JOIN taction T2 ON T1.agent_id = T2.agent_id
8     JOIN office_income T3 ON T2.taction_id = T3.taction_id
9     WHERE T2.taction_id = @TransactionID
10    END

```

Procedura swoim działaniem po transakcji dodaje rekord w tabeli zyski agenci.

Procedura Agent_month_profits

```
1 CREATE PROCEDURE [Agent_month_profits]
2   @Transaction_Date DateTime
3 AS
4 BEGIN
5   SELECT T1.agent_id , MONTH(@Transaction_Date) , SUM(T1.agent_commission) AS Salary FROM
6     agent_commission T1
7   WHERE MONTH(T1.transaction_date) = MONTH(@Transaction_Date) AND YEAR(T1.transaction_date)=
8     YEAR(@Transaction_Date)
9   GROUP BY T1.agent_id
10 END
```

Procedura wyliczająca zyski agenta w danym miesiącu.

Procedura Estate status change

```
1 CREATE PROCEDURE [Estate status change]
2   @EstateID int
3 AS
4 BEGIN
5   UPDATE estate_info SET availability = 0
6   WHERE @EstateID = estate_id
7 END
```

Procedura, która po transakcji zmienia status nieruchomości na sprzedaną.

5.3 Funkcje definiowane przez użytkownika

5.3.1 Funkcje tabelaryczne

Funkcja Peopleinrelation

```
1 GO
2 CREATE FUNCTION People_in_relation()
3 RETURNS @people TABLE (
4   name NVARCHAR(256) ,
5   surname NVARCHAR(256) ,
6   phonenumber int ,
7   email NVARCHAR(256) ,
8   type_of_person NVARCHAR(256)
9 )
10 AS
11 BEGIN
12   INSERT INTO @people
13   SELECT
14     name ,
15     surname ,
16     phone_number ,
17     email ,
18     'Agent '
19 FROM
```

```

20         agent;
21
22     INSERT INTO @people
23     SELECT
24         name,
25         surname,
26         phone_number,
27         email,
28         'Client'
29     FROM
30         client;
31     RETURN;
32 END;

```

Jedyna funkcja definiowana przez użytkownika w naszej tabeli zwracająca wspólne informacje (tzn. imię, nazwisko, telefon i mail) na temat agenta i klienta. Dodatkowo w zależności od tego czy osoba sprawuje w bazie danych funkcje agenta czy funkcje klienta, wyświetlane jest dodatkowe pole z odpowiednim tytułem.

Funkcja Expensecounter

```

1  GO
2  CREATE FUNCTION Expense_counter (@estate int)
3      RETURNS @Expenses TABLE (
4      id_estate int,
5      total_price int,
6      rent_amount int,
7      administrative_fee int,
8      media int,
9      internet int,
10     tax int,
11     one_off_expense int,
12     monthly_expense int
13 )
14 AS
15 BEGIN
16     INSERT INTO @Expenses
17     SELECT T1.estate_id, T1.total_price, T2.rent,
18         T2.administrative_fee,
19         T2.media, T2.internet, T2.tax, T1.total_price,
20         SUM(T2.rent + T2.administrative_fee +
21             T2.media + T2.internet + T2.tax)
22     FROM estate_price AS T1 JOIN estate_monthly_expenses AS T2
23     ON T1.estate_id = T2.estate_id
24     WHERE T1.estate_id = @estate
25     GROUP BY T1.estate_id, T1.total_price, T2.rent,
26         T2.administrative_fee,
27         T2.Media, T2.Internet, T2.tax, T1.total_price
28     RETURN
29 END

```

Funkcja grupująca wszystkie potrzebne dla zainteresowanego transakcją informacje dotyczące wydatków związanych z konkretną nieruchomością (@estate jako identyfikator nieruchomości) takich jak: całkowita cena nieruchomości, czynsz, opłata administracyjna, opłata za media, opłata za internet oraz podatek. Funkcja dodatkowo podlicza i przedstawia ostateczny jednorazowy koszt oraz miesięczne opłaty, które może ponieść potencjalny klient.

5.3.2 Funkcje skalarne

Funkcja Positions of agent by text

```
1 GO
2 CREATE FUNCTION [Position of agent by text](@stat [tinyint])
3 RETURNS nvarchar(256)
4 AS
5 BEGIN
6     DECLARE @position nvarchar(256);
7
8     SET @position =
9         CASE @Stat
10             WHEN 1 THEN 'Szef Oddzialu'
11             WHEN 1 THEN 'Wiceszef Oddzialu'
12             WHEN 3 THEN 'Starszy Zarzadca'
13             WHEN 4 THEN 'Zarzadca Oddzialu'
14             WHEN 5 THEN 'Asystent Dzialu'
15             WHEN 60 THEN 'Agent Nieruchomosci'
16             WHEN 61 THEN 'Rzecznawca majatkowy'
17             WHEN 62 THEN 'Kierownik techniczny obiektu'
18             WHEN 63 THEN 'Asystent Architekta'
19             WHEN 64 THEN 'Asystent Handlowy'
20             ELSE 'Nieprawidlowy identyfikator pozycji'
21         END;
22
23     RETURN @position
24 END;
```

Funkcja, która w zależności od podanego jako argument numeru (liczby o typie całkowitym) zwraca tekstowo jego pozycje w hierarchii danego biura nieruchomości

Funkcja convert earnings

```
1 GO
2 CREATE FUNCTION [Convert_earnings]
3     (@Convert int, @Currency VARCHAR(5))
4 RETURNS DECIMAL(38,2)
5 AS
6 BEGIN
7     DECLARE @final decimal(20,2)
8     DECLARE @PLN DECIMAL(20,2)
9     DECLARE @EU DECIMAL(20,2)
10    DECLARE @GBP DECIMAL(20,2)
11    DECLARE @USA DECIMAL(20,2)
12    DECLARE @YEN DECIMAL(20,2)
13    DECLARE @CHF DECIMAL(20,2)
14    DECLARE @AUD DECIMAL(20,2)
15    DECLARE @RUB DECIMAL(20,2)
16
17    SET @final = 0.00
18    SET @PLN = 1
19    SET @EU = 4.53
20    SET @GBP = 5.11
21    SET @USA = 3.74
22    SET @YEN = 0.03
23    SET @CHF = 4.21
24    SET @AUD = 2.86
25    SET @RUB = 0.05
26
27    BEGIN
28        IF (@Currency = 'POL')
29            BEGIN
```

```

30         SET @final = @Convert / @PLN
31     END
32     ELSE IF (@Currency = 'EU')
33     BEGIN
34         SET @final = @Convert / @EU
35     END
36     ELSE IF (@Currency = 'GBP')
37     BEGIN
38         SET @final = @Convert / @GBP
39     END
40     ELSE IF (@Currency = 'USA')
41     BEGIN
42         SET @final = @convert / @USA
43     END
44     ELSE IF (@Currency = 'YEN')
45     BEGIN
46         SET @final = @convert / @YEN
47     END
48     ELSE IF (@Currency = 'CHF')
49     BEGIN
50         SET @final = @convert / @CHF
51     END
52     ELSE IF (@Currency = 'AUD')
53     BEGIN
54         SET @final = @convert / @AUD
55     END
56     ELSE IF (@Currency = 'RUB')
57     BEGIN
58         SET @final = @convert / @RUB
59     END
60 END
61 RETURN
62     @final
63 END

```

Funkcja ta konwertuje przekazaną jako argument ilość pieniędzy z polskich złotych na wybraną przez siebie walutę przekazaną jako drugi argument. Możliwymi walutami do przeliczeń są: dolary amerykańskie, funty brytyjskie, dolary australijskie, euro, japońskie jeny, franki szwajcarskie oraz ruble rosyjskie.

5.4 Widoki

Widok equipped apartments

```

1 GO
2 CREATE VIEW [equipped apartments]
3 AS
4 select W.estate_id , total_area , no_rooms ,
5 coalesce(elevator , 0)+coalesce(fireplace , 0)+coalesce(ceiling , 0)+coalesce(dishwasher , 0)+
6 coalesce(oven_or_stove , 0)+coalesce(security , 0)+coalesce(dryer , 0)+coalesce(washer , 0)+
7 coalesce(garden , 0)+coalesce(patio , 0)+coalesce(fridge , 0)+coalesce(storage_area , 0)+
8 coalesce(wardrobe , 0)+coalesce(swimming_pool , 0)+coalesce(microwave , 0)+coalesce(freezer , 0)
9 +
10 coalesce(garbage_disposal , 0)+coalesce(fence , 0)+coalesce(gym , 0)+coalesce(air_conditioning ,
11 0)
12 AS [no_of_equipped_elements]
13 from homeware w join residential_estate m on w.estate_id=m.estate_id
14 where
15 coalesce(elevator , 0)+coalesce(fireplace , 0)+coalesce(ceiling , 0)+coalesce(dishwasher , 0)+
16 coalesce(oven_or_stove , 0)+coalesce(security , 0)+coalesce(dryer , 0)+coalesce(washer , 0)+
17 coalesce(garden , 0)+coalesce(patio , 0)+coalesce(fridge , 0)+coalesce(storage_area , 0)+

```

```

16  coalesce(wardrobe, 0)+coalesce(swimming_pool, 0)+coalesce(microwave, 0)+coalesce(freezer, 0)
17  +
18  coalesce(garbage_disposal, 0)+coalesce(fence, 0)+coalesce(gym, 0)+coalesce(air_conditioning,
    0)
    > 10

```

Widok pozwalana szybkie wyszukanie nieruchomości mieszkalnych, które posiadają ponad połowe cech i atrybutów z tabeli wyposażenie (inaczej mówiąc pozwala na wyszukanie przyzwoicie zaopatrzonych mieszkań).

Widok equipped apartments

```

1  CREATE VIEW [dbo].[Employment_history] AS
2  SELECT agent_id, promotion_date, position_id
3  FROM employment_history
4  GROUP BY agent_id, promotion_date, position_id

```

Widok pokazujący uporządkowane historie zatrudnienia agentów.

Widok equipped apartments

```

1  CREATE VIEW [dbo].[Price_history] AS
2  SELECT estate_id, event_date, total_price
3  FROM price_history
4  GROUP BY estate_id, event_date, total_price

```

Widok pokazujący uporządkowane historie cen nieruchomości.

Widok Vendor and Buyer

```

1  CREATE VIEW [Vendor and Buyer]
2  AS
3  SELECT T1.client_id, T1.name, T1.surname, T1.nationality, 'Vendor' AS Relation FROM client
4  T1
5  JOIN taction T2 ON T2.vendor_id1 = T1.client_id
6  UNION
7  SELECT T3.client_id, T3.name, T3.surname, T3.nationality, 'Buyer' FROM client T3
8  JOIN taction T4 ON T4.buyer_id = T3.client_id

```

Stworzony widok przedstawia wszystkich klientów (kupujących i sprzedających) wraz z odpowiednim tytułem określającym ich funkcje. Widok ułatwia rozróżnienie klientów, ponieważ w naszej bazie kupujący i sprzedający są zgromadzeniu w jednej tabeli client"

Widok Estates Below Average Price

```

1  CREATE VIEW [Estates Below Average Price]
2  AS
3  SELECT estate_id, total_price FROM estate_price
4  WHERE (total_price < (SELECT AVG(total_price) AS Price FROM estate_price))

```

Widok przedstawia nieruchomości o cenie niższej niż średnia cena wśród wszystkich nieruchomości. Szczególnie przydatny jest ten widok dla osób poszukujących nieruchomości o niskiej cenie z uwagi np. na swoje niskie zarobki

Widok Estate Type

```
1 CREATE VIEW [Estate Type]
2 AS
3 SELECT estate_id, 'residential_estate' AS Estate_Type FROM residential_estate
4 UNION
5 SELECT estate_id, 'comercial_estate' FROM comercial_estate
6 UNION
7 SELECT estate_id, 'special_estate' FROM special_estate
8 UNION
9 SELECT estate_id, 'ground' FROM ground
```

Widok ukazuje identyfikator nieruchomości wraz z opisem rodzaju tej nieruchomości. Dotychczas w naszej bazie nieruchomości z różnych typów były rozdzielone po różnych tabelach zatem taki widok ułatwia odnalezienie danej nieruchomości o konkretnym typie.

Widok Estates Below Average Price with Type

```
1 CREATE VIEW [Estates Below Average Price with Type]
2 AS
3 SELECT T1.*, T2.total_price FROM [Estate Type] AS T1 JOIN
4 (SELECT estate_id, total_price FROM estate_price
5 WHERE (total_price < (SELECT AVG(total_price) AS Price FROM estate_price))) AS T2
6 ON T1.estate_id = T2.estate_id
```

Widok przedstawia nieruchomości o cenie poniżej średniej ceny wśród nieruchomości wraz z uwzględnieniem typu danej nieruchomości.

Widok Estates Price with Type

```
1 CREATE VIEW [Estates Price with Type]
2 AS
3 SELECT T1.*, T2.total_price FROM [Estate Type] T1 JOIN
4 (SELECT estate_id, total_price FROM estate_price) AS T2
5 ON T1.estate_id = T2.estate_id
```

Widok łączy w sobie widok [Estate Type] zawierający informacje o identyfikatorze nieruchomości oraz typie nieruchomości. Dodatkowo uwzględnia cenę danej nieruchomości.

Widok Transaction by Estate Type

```
1 CREATE VIEW [Transaction by Estate Type]
2 AS
3 SELECT T1.transaction_id, T1.vendor_id1, T1.buyer_id, T1.agent_id, T2.* FROM taction AS T1
4 JOIN [Estates Price with Type] AS T2
5 ON T1.estate_id = T2.estate_id
```

Widok łączy w sobie widok [Estates Price with Type]. Zawiera informacje o identyfikatorze nieruchomości, typie nieruchomości, uwzględnia cenę danej nieruchomości oraz dodatkowo przedstawia dane dotyczące transakcji związanej z konkretną nieruchomością oraz wypisuje pośredników biorących udział w tej transakcji.

5.5 Wyzwalacze

wyzwalacz update promotion date

```
1 GO
2 CREATE TRIGGER [update promotion date]
3 ON employment_history
4 AFTER UPDATE
5 AS
6     UPDATE employment_history
7     SET promotion_date = GETDATE()
8     WHERE agent_id IN (SELECT DISTINCT agent_id FROM Inserted)
```

Wyzwalacz typu AFTER UPDATE. Uruchamia się on gdy chcemy zaktualizować w tabeli *employmenthistory* datę awansu agenta. W takim wypadku automatycznie wpisywana jest data modyfikacja danego wiersza z tejże tabeli.

wyzwalacz we are not billionaires

```
1 GO
2 CREATE TRIGGER [we are not billionaires]
3 on office_income
4 FOR UPDATE
5 AS
6 IF EXISTS
7     (SELECT 'True'
8      FROM Inserted i
9      JOIN Deleted d
10         ON i.taction_id = d.taction_id
11         AND i.office_id = d.office_id
12         WHERE (d.agent_commission + i.agent_commission) >= 10000 OR
13              i.agent_commission >= 10000
14     )
15 BEGIN
16     RAISERROR('Nie mozna podniesc stawki agenta jesli przekracza ona 10000',10,1)
17     ROLLBACK TRANSACTION
18 END
```

Wyzwalacz typu FOR UPDATE. Uruchamia się w wypadku gdy chcemy zaktualizować (w domyśle podnieść) prowizję agenta w taki sposób, że jej nowa wartość będzie większa lub równa 10000 zł. W takim wypadku transakcja która będzie chciała zmienić prowizję zostanie wycofana i zostanie wyświetlony błąd o treści mówiącej, że nie można podnieść w taki sposób prowizji agenta.

wyzwalacz archive estate

```
1 GO
2 CREATE TRIGGER [archive estate]
3 ON estate_info
4 INSTEAD OF UPDATE
5 AS
6 BEGIN
7     UPDATE estate_info
8     SET
9         estate_id = INSERTED.estate_id,
10         estate_type = INSERTED.estate_type,
11         when_built = INSERTED.when_built,
12         market_type = INSERTED.market_type,
13         availability = INSERTED.availability
```

```

14 FROM INSERTED
15 WHERE INSERTED.estate_type = estate_info.estate_id
16    AND INSERTED.availability = 1
17 ;
18 DELETE FROM estate_info
19 FROM INSERTED
20 WHERE INSERTED.estate_id = estate_info.estate_id
21    AND INSERTED.availability = 0
22 ;
23 INSERT INTO [deleted_estate] (deleted_estate_id, deleted_estate_type, deleted_market_type,
    delted_when_built)
24 SELECT estate_id, estate_type, market_type, when_built
25 FROM INSERTED
26 WHERE availability = 0
27 ;
28 END

```

Wyzwalacz typu INSTEAD OF UPDATE. W zależności od wartości w tabeli estateinfo w kolumnie availability albo wstawia nowy rekord zawierający informacje o identyfikatorze nieruchomości, typie nieruchomości, dacie budowy nieruchomości oraz rodzaju rynku do tabeli estateinfo albo (w przypadku gdy chcemy zaktualizować rekord o wartości availability wynoszącej 0) przenosi wstawiany rekord do tabeli deletedestate, która przechowuje informacje na temat niedostępnych do wykonania transakcji nieruchomości.

wyzwalacz no delete clients

```

1 GO
2 CREATE TRIGGER [no delete clients] ON [client]
3 INSTEAD OF DELETE NOT FOR REPLICATION AS
4 BEGIN
5     IF @@rowCount = 0
6         RETURN;
7     SET NOCOUNT ON;
8
9     BEGIN
10        RAISERROR
11        (N'Klienci nie mogą zostać usunięci, ich rekordy mogą zostać jedynie
    zaktualizowane.', 10, 1);
12        IF @@TRANCOUNT > 0
13            BEGIN
14                ROLLBACK TRANSACTION;
15            END
16        END;
17 END;

```

Wyzwalacz typu INSTEAD OF DELETE. Uruchamia się w przypadku gdy chcemy usunąć z tabeli klienci jakikolwiek rekord dotyczący klienta. Taka transakcja zostanie wycofana i wyświetli się komunikat informujący, że rekordy klientów nie podlegają procesowi usuwania.

wyzwalacz after update of prices

```

1 GO
2 CREATE TRIGGER [after update of prices] on [estate_price]
3 FOR UPDATE
4 AS DECLARE @EstateID INT,
5            @totalprice int,
6            @purchasetax int,
7            @comissionrate tinyint,
8            @ActionPerformed VARCHAR(50);

```



```

9
10 SELECT @EstateID = ins.estate_id FROM INSERTED ins;
11 SELECT @totalprice = ins.total_price FROM INSERTED ins;
12 SELECT @purchasetax = ins.purchase_tax FROM INSERTED ins;
13 SELECT @comissionrate = ins.commission_rate FROM INSERTED ins;
14 IF UPDATE(total_price)
15 BEGIN
16     SET @ActionPeformed = 'Calkowita cena nieruchomosci zostala zaktualizowana'
17 END
18 IF UPDATE(purchase_tax)
19 BEGIN
20     SET @ActionPeformed = 'Podatek od nieruchomosci zostal zaktualizowany'
21 END
22 UPDATE [estate_price]
23     SET [total_price] = @totalprice ,
24         [purchase_tax] = @purchasetax ,
25         [comission_rate] = @comis sionrate
26 WHERE estate_id = @estateID;
27 PRINT 'Wykonany zostal wyzwalacz typu: AFTERUPDATE.'
28 GO

```

Wyzwalacz typu AFTER UPDATE. Gdy zaktualizujemy w tabeli estateprice tylko pole mówiące o całkowitej cenie nieruchomości wyświetli się komunikat jasno określający, że całkowita cena nieruchomości została zaktualizowana. Analogiczny komunikat zostanie wyświetlony gdy zaktualizowany zostanie podatek od nieruchomości. Bez względu na wykonaną akcję po działaniu wyzwalacza zostanie wyświetlony napis: wykonany został wyzwalacz typu: AFTER UPDATE.

wyzwalacz fire employees

```

1 CREATE TRIGGER fire_employees on agent
2 FOR DELETE
3 AS
4     declare @number INT
5     INSERT INTO [Fired Agents]
6     SELECT agent_id, name, surname, GETDATE() [Date of firing] FROM DELETED;
7     set @number=(select count(*) from deleted)
8     PRINT 'Z sukcesem pozbylismy sie '+cast(@number as nvarchar(256))+ ' pracownikow naszej firmy
9     !'
10    GO

```

Wyzwalacz typu FOR DELETE. Wyzwalacz zostanie uruchomiony w przypadku gdy chcemy usunąć rekordy z tabeli agent. W takim wypadku rekord który miał zostać usunięty, będzie przeniesiony do tabeli [Fired Agents]. Na końcu zostanie wyświetlony napis wskazujący na to ilu pracowników biura nieruchomości zostało zwolnionych.

5.6 Inne elementy bazy danych

```

1 ;WITH pearsoncorr AS
2 (SELECT localization as X, price_per_m2 as Y from parameters
3 ),
4 Agg AS
5 (
6     SELECT AVG(X) as Xavg, AVG(Y) as Yavg, Stdev(X) as deviationX, Stdev(Y) as deviationY
7     FROM pearsoncorr
8 ),
9 maths AS
10 (

```

```

10 SELECT AVG((X-Xavg)*(Y-Yavg)) as covariance ,
11 MAX(deviationX) maxdevX, MAX(deviationY) maxdevY FROM pearsoncorr CROSS JOIN Agg
12 )
13 SELECT CAST(covariance/(maxdevX*maxdevY) AS Numeric(3,2)) AS [Pearson correlation] INTO
    Pearson FROM maths

```

Przykład zastosowania CTE w bazie danych do wyliczenia współczynnika korelacji pearsona pomiędzy lokalizacją a ceną mieszkania za m2. Współczynnik korelacji jest zapisywany do tabeli Pearson. Współczynnik Pearsona jest ilorzem kowariancji i iloczynu odchyłeń standardowych. Tak też jest on wyliczany w CTE: najpierw liczymy średnie wartości obu zmiennych z tabeli oraz ich odchylenia standardowe. Następnie obliczamy kowariancję oraz maksymalne odchylenie dla każdej próbki. Za pomocą ostatniego zapytania SELECT wyliczamy ostateczną wartość współczynnika korelacji i jako że działamy na CTE, to abyśmy mogli poza nim wykonywać operacje na współczynniku - zapisujemy jego wartość do oddzielnej tabeli (nie jest uwzględniana ona ani w schemacie relacyjnym bazy ani w diagramie ER).

```

1 select
2 case
3 when [Pearson correlation]=-1 then 'korelacja pelna negatywna'
4 when [Pearson correlation]<=-0.9 and [Pearson correlation] > -1 then 'korelacja niemal pelna
    negatywna'
5 when [Pearson correlation]<=-0.7 and [Pearson correlation] > -0.9 then 'korelacja bardzo
    wysoka negatywna'
6 when [Pearson correlation]<=-0.5 and [Pearson correlation] > -0.7 then 'korelacja wysoka
    negatywna'
7 when [Pearson correlation]<=-0.3 and [Pearson correlation] > -0.5 then 'korelacja przecietna
    negatywna'
8 when [Pearson correlation]<=-0.1 and [Pearson correlation] > -0.3 then 'korelacja slaba
    negatywna'
9 when [Pearson correlation]<0 and [Pearson correlation] > -0.1 then 'korelacja nikla
    negatywna'
10 when [Pearson correlation]=0 then 'brak korelacji'
11 when [Pearson correlation]>0 and [Pearson correlation] <= 0.1 then 'korelacja nikla
    pozytywna'
12 when [Pearson correlation]>0.1 and [Pearson correlation] <= 0.3 then 'korelacja slaba
    pozytywna'
13 when [Pearson correlation]>0.3 and [Pearson correlation] <= 0.5 then 'korelacja przecietna
    pozytywna'
14 when [Pearson correlation]>0.5 and [Pearson correlation] <= 0.7 then 'korelacja wysoka
    pozytywna'
15 when [Pearson correlation]>0.7 and [Pearson correlation] <= 0.9 then 'korelacja bardzo
    wysoka pozytywna'
16 when [Pearson correlation]>0.9 and [Pearson correlation] < 1 then 'korelacja niemal pelna
    pozytywna'
17 when [Pearson correlation]=1 then 'korelacja pelna pozytywna'
18 end from Pearson

```

Na podstawie wartości wyżej wyliczonego współczynnika korelacji określamy siłę i typ korelacji pomiędzy lokalizacją a ceną za m2. W zależności od tego czy te dwie zmienne zależą od siebie wprost proporcjonalnie czy odwrotnie proporcjonalnie, korelacja będzie albo pozytywna albo negatywna.

```

1 select slope , wsp_b1 - wsp_b0 * slope intercept into [regression_line]
2 from (
3     select sum((localization - local_bar) * (price_per_m2 - price_bar)) /
4     sum((localization - local_bar) * (localization - local_bar)) slope ,

```

```

5      max(local_bar) as wsp_b0, max(price_bar) as wsp_b1
6  from (
7      select localization, avg(localization) over () as local_bar,
8      price_per_m2, avg(price_per_m2) over () as price_bar
9      from parameters) s1
10 ) s2

```

Zapytanie wylicza i wstawia do tabeli [regression line] parametry prostej regresji (także dla zmiennej lokalizacja i zmiennej cena za m2). Z tych parametrów można skonstruować następnie wykres przedstawiający rozkład punktów (czyli wartości z tabeli parameters (dla nas są to localization i price_per_m2)) względem prostej regresji.

6 Strategia pielęgnacji i konserwacji bazy danych

Przyjęcie jednej skutecznej strategii zarządzania i pielęgnowania bazy danej zawsze stanowi dość duży problem z racji tego, że (zwłaszcza w przypadku mniejszych baz) istnieje wiele efektywnych sposobów na dbanie i kontrole nad danymi w bazie. Rozwiązaniem wykorzystanym w omawianej bazie jest podział czynności doglądania bazy pomiędzy użytkownikami posiadającymi odpowiednie uprawnienia a zautomatyzowanie procesu pielęgnacji aby wykonywał się on autonomicznie o określonej porze bez względu na użytkownika zarządzającego bazą. Szczegółowy harmonogram konserwacji bazy można przeczytać z listy poniżej:

1. Pełna kopia zapasowa bazy danych (System) - codziennie o 10:00

- (a) Weryfikacja sum kontrolnych odczytywanych z dysków danych
- (b) Sprawdzenie spójności całej bazy danych
- (c) Sporządzenie kopii zapasowej bazy
- (d) Sprawdzenie poprawności kopii zapasowej bazy danych

2. Pełna kopia zapasowa bazy danych (Użytkownik) - codziennie o 10:00

- (a) Defragmentacja indeksów bazy użytkowników
- (b) Aktualizacja statystyk dotyczących użytkowników
- (c) Sprawdzanie warunku spójności bazy danych
- (d) Sporządzenie kopii zapasowej bazy

3. Pełna kopia zapasowa dziennika transakcji (Użytkownik) - codziennie co 2 godziny zaczynając o 10:00:00 i kończąc o 23:59:59

- (a) Wykonanie kopii logów dotyczących transakcji użytkowników.

4. Kopia różnicowa bazy danych (Użytkownik) - od poniedziałku do piątku o 11:00

- (a) Przeprowadzenie Differential backup na danych dotyczących użytkowników

5. Czyszczenie bazy danych - co poniedziałek o 8:00

- (a) Wykonanie polecenia sp_delete_backuphistory
- (b) Wykonanie polecenia sp_purge_jobhistory

7 Typowe zapytania

```
1 select agent_id, a.office_id, position_id, name, surname from agent a join office_income o
2 on o.office_id=a.office_id
3 where agent_commission>1000
```

```
1 select r.estate_id, total_area, city, service_type from [location] l join residential_estate
2 r
3 on l.estate_id=r.estate_id join service s on r.estate_id=s.estate_id
4 where (usable_area>100 or no_rooms>2)
and post_code not in (select post_code from location where post_code like '30-0__' )
```

```
1 select c.client_id, name, surname, avg(monthly_income), avg(monthly_expenses) from client c
2 join wealthiness w
3 on c.client_id=w.client_id
4 where (select avg(monthly_income) from wealthiness) between 10000 and 100000
group by c.client_id, name, surname
```

```
1 select estate_id, street, district, city, region, country from [location] l join
2 location_details d
3 on l.post_code=d.location_post_code and (country=N'Poland' or country=N'United Kingdom' or
4 country=N'Portugal'
5 or country=N'Spain' or country=N'France' or country=N'Switzerland' or country=N'Belgium' or
6 country=N'Netherlands'
7 or country=N'Italy' or country=N'Germany' or country=N'Austria' or country=N'Irleand' or
8 country=N'Czech Republic'
or country=N'Slovakia' or country=N'Hungary' or country=N'Greece' or country=N'Romania' or
country=N'Bulgaria'
or country=N'Moldova' or country=N'Ukraine' or country=N'Belarus' or country=N'Russia' or
country=N'Finland'
or country=N'Estonia' or country=N'Latvia' or country=N'Lithuania' or country=N'Sweden' or
country=N'Norway'
or country=N'Denmark' or country=N'Serbia' or country=N'Croatia' or country=N'Slovenia' or
country=N'Albania')
```

```
1 select s.estate_id, area from special_estate s join estate_info e
2 on s.estate_id=e.estate_id
3 where availability=1 and when_built>'2001-01-01' and (destination=N'School' or destination=
4 University')
```

```
1 select estate_id, event_date, total_price from price_history
2 where event=N'Change in the exchange rate of foreign currencies against PLN'
```