Poradnik wdrożeniowca

Gniazda rozszerzeń w WAPRO Mag

Obowiązuje od wersji 8.30.0

Spis treści

1. Wstęp – czym są gniazda rozszerzeń?

Niniejszy dokument zawiera opis i przykłady zastosowania gniazd rozszerzeń dostępnych w programie WAPRO Mag.

Czym są gniazda rozszerzeń? Najkrócej można zdefiniować gniazda rozszerzeń jako miejsca w programie, które umożliwiają automatyczne wywoływanie modułów rozszerzających standardową funkcjonalność programu. Nazwy gniazd rozszerzeń odzwierciedlają zdarzenia jakie zachodzą podczas pracy z programem. Mamy więc gniazdo po zatwierdzeniu dokumentu handlowego, przed dodaniem pozycji dokumentu itd. Program realizując standardowe operacje, gdy dochodzi w kolejności wykonania do gniazda rozszerzeń, rozpoczyna analizę tego co znajduje się w gnieździe i wykonuje moduły rozszerzające, które podpięliśmy do tego gniazda. Mechanizm ten umożliwia automatyczne wywoływanie własnych procedur SQL, modułów wykonywalnych, generowanie komunikatów, warunkowe wywoływanie funkcji itp.

Funkcjonalność gniazd rozszerzeń przypomina dostępne w systemie Operacje dodatkowe, różnica polega na tym, iż automat wywołujący procedury robi to bez potrzeby ingerencji użytkownika.

Za pomocą instrukcji w gnieździe możemy również sterować zachowaniem się programu. Możemy zadać pytanie użytkownikowi i wykonać moduł rozszerzeń w zależności od jego decyzji. Można również sprawdzić wartość określonych parametrów i w zależności od wyniku sprawdzenia podjąć lub przerwać dalsze wykonywanie sekwencji zdarzeń w programie. Oczywiście można również zablokować wykonanie standardowych funkcji programu lub też ominąć ich wykonanie w sposób niewidoczny dla użytkownika tzn. zrealizować własną procedurę postępowania zamiast wbudowanej w program.

W dalszej części dokumentu opisane zostały przykładowe zastosowania jakie można zrealizować za pomocą gniazd rozszerzeń. Możliwości wykorzystania gniazd rozszerzeń ograniczone są jedynie naszą wyobraźnią. Można wzbogacić program WAPRO Mag o elementy:

- serwisu maszyn i urządzeń;
- wypożyczalni;
- prowadzenia i rozliczania projektów;

.... i wiele innych zastosowań wynikających z potrzeb użytkowników.

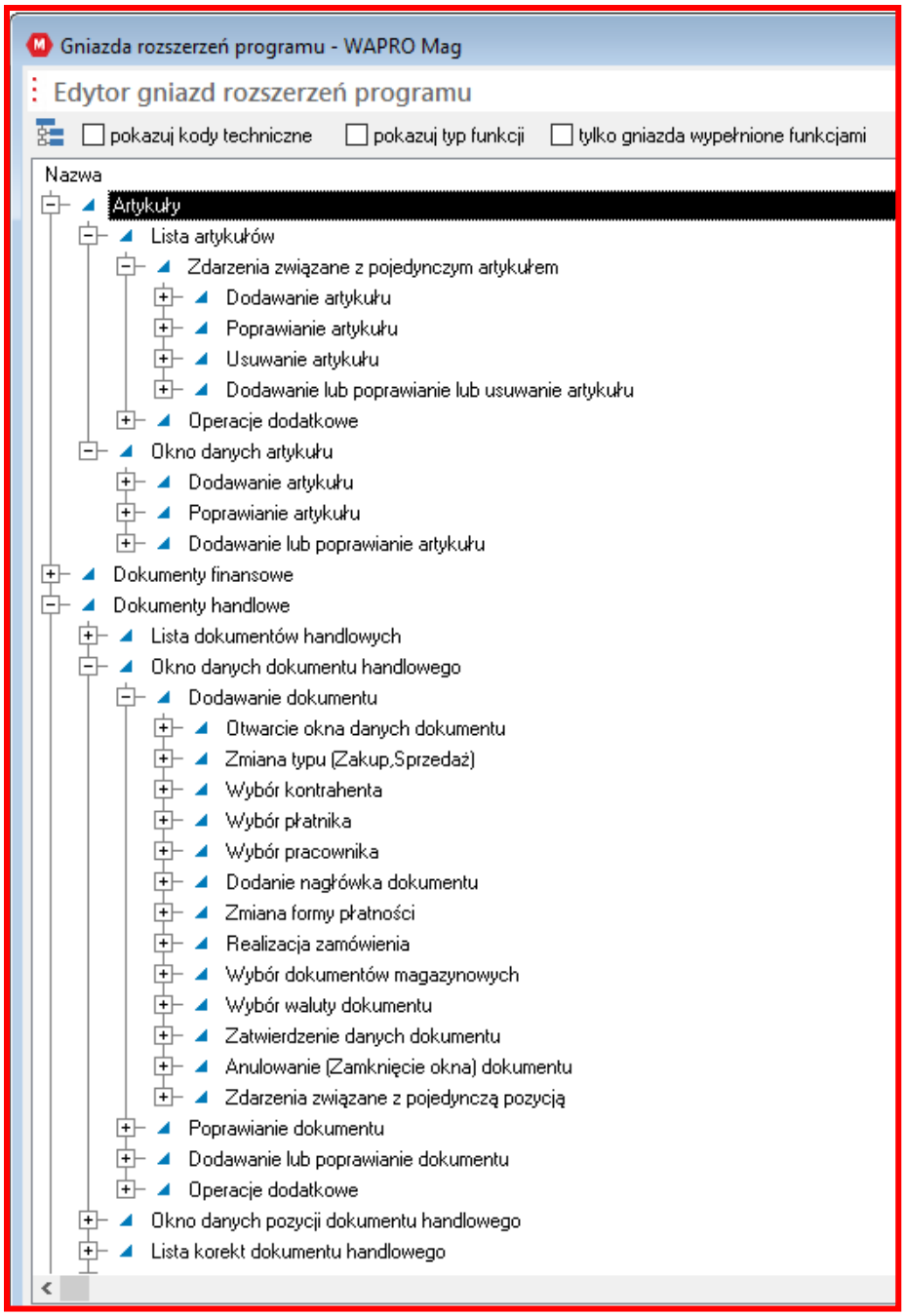
2. Edytor gniazd rozszerzeń - informacje podstawowe

Gniazda rozszerzeń dostępne są na większości podstawowych ekranów programu tj. m.in. w:

- kartotece artykułów i kontrahentów;
- listach i formularzach dokumentów handlowych;
- listach i formularzach dokumentów magazynowych;
- listach i formularzach zamówień;
- listach i formularz zleceń produkcyjnych;
- listach i formularzach rozrachunków;
- listach i formularzach dokumentów finansowych;

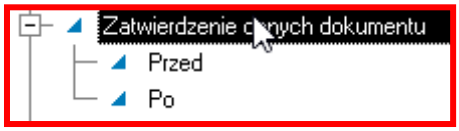
Pełna lista gniazd rozszerzeń oraz możliwość podpięcia wywołania własnych dodatkowych modułów jest dostępna poprzez Edytor Gniazd Rozszerzeń uruchamiany skrótem klawiszowym CTRL+SHIFT+F12 na poszczególnych ekranach programu (np. w oknie dodawania dokumentu handlowego). Edytor dostępny jest również w menu głównym programu „Administrator > Definicje > Gniazda rozszerzeń programu".

Gniazda uporządkowane są w strukturze drzewa. Drzewiasta hierarchia gniazd ułatwia nawigację i umożliwia szybkie wyszukanie miejsca gdzie znajduje się potrzebne gniazdo. Na najwyższym poziomie jest SYSTEM, do którego przypięte są pozostałe elementy drzewa. I tak kolejno, na pierwszym niższym poziomie znajdują się nazwy modułów funkcjonalnych (Artykuły, Kontrahenci, ... , Dokumenty handlowe itd.). Następny poziom tworzą listy danych (np. Lista dokumentów handlowych), okna danych (np. artykułu, dokumentu) oraz w przypadku dokumentów – okna danych pozycji dokumentu. Kolejny poziom tworzą już konkretne zdarzenia np. dodawanie dokumentu oraz zdarzenia szczegółowe (np. otwarcie okna danych dokumentu, wybór kontrahenta ,zatwierdzenie dokumentu) (Rys 1) .



Rys 1. Hierarchia gniazd rozszerzeń

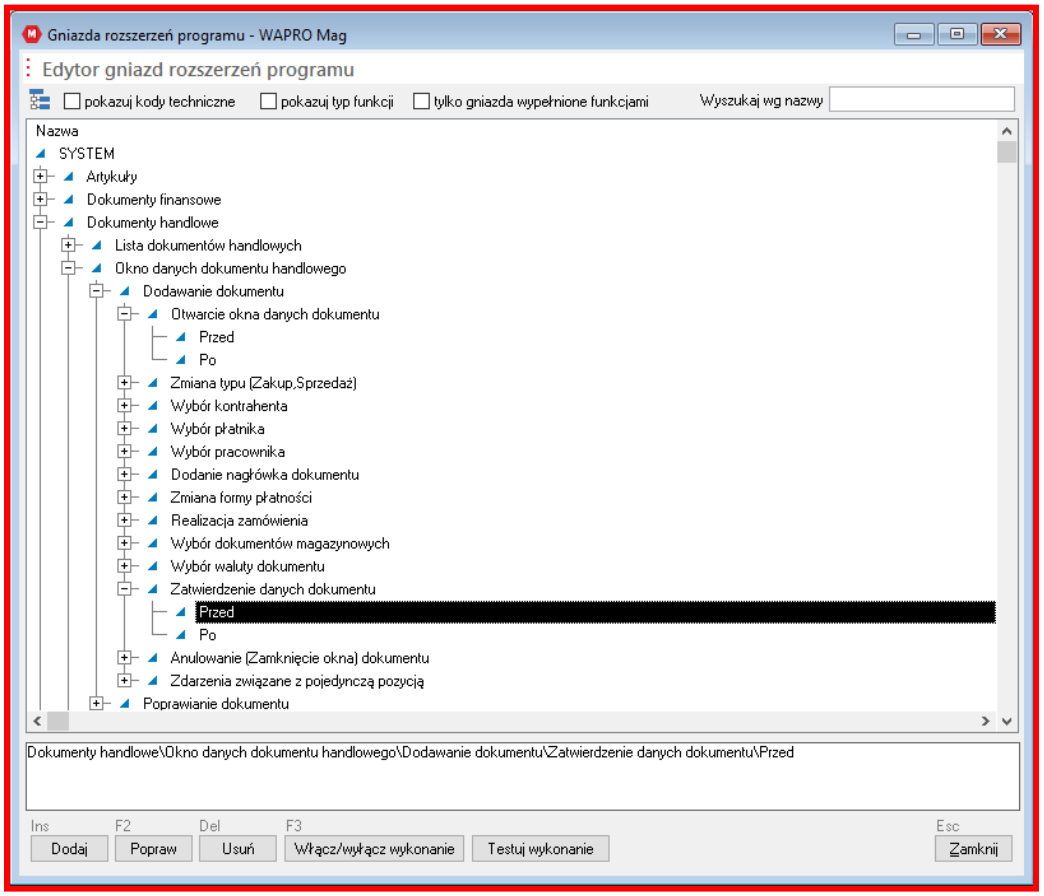
Na najniższym, ostatnim poziomie znajdują się zdarzenia typu „przed" i „po" np. przed zatwierdzeniem danych dokumentu i po zatwierdzeniu danych dokumentu (Rys 2)



Rys 2. Hierarchia gniazd rozszerzeń – gniazda przed i po

Tylko w gniazdach najniższego poziomu (czyli typu „przed" lub „po") można umieszczać moduły rozszerzeń (własne funkcje). Gniazdo jest identyfikowane poprzez jego adres - adres gniazda. Adres jest

zapisany w formie tekstowej. W adresie wymieniane są po kolei poziomy od najwyższego do najniższego, które prowadzą do wybranego miejsca w strukturze drzewa np. „*Dokumenty handlowe \ Okno danych dokumentu handlowego \ Dodawanie dokumentu \ Zatwierdzenie danych dokumentu \ Przed*" (Rys 3)



Rys 3. Edytor gniazd rozszerzeń – w części dolnej widać adres gniazda

Edytor w dolnej części okna zawsze podpowiada adres gniazda ułatwiając orientację w strukturze.

Dodawanie funkcji i operacji w gniazdach odbywa się standardowo poprzez przyciski Dodaj, Popraw, Usuń. Nie ma żadnych ograniczeń co do ilości funkcji dodanych gnieździe.

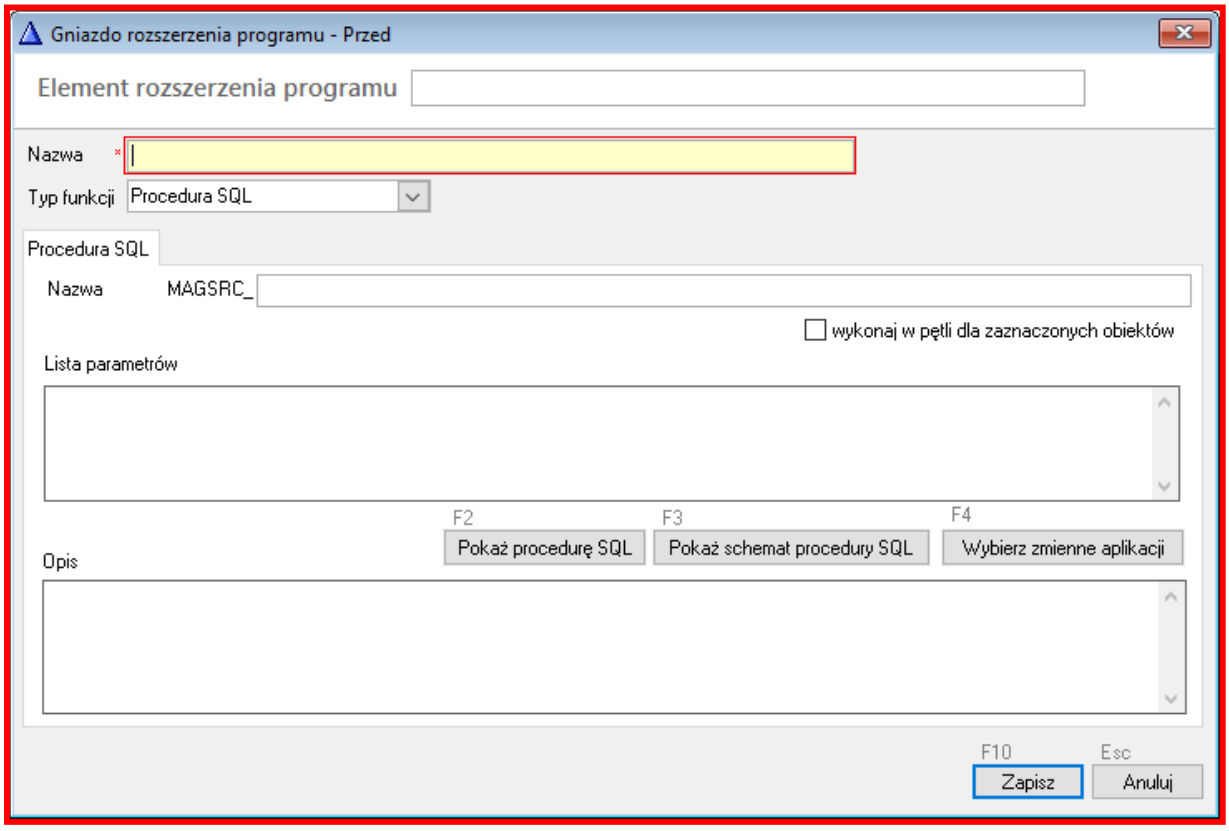
3. Dostępne rodzaje funkcji w gniazdach rozszerzeń

W gniazdach rozszerzeń można stosować 9 typów funkcji:

- *Procedura SQL*
- *Skrypt VBS*
- *Moduł wykonywalny*
- *Komunikat* - wywołanie okna komunikatu
- *Pytanie* - wywołanie okna zapytania
- *Koniec* - funkcja przerywająca (kończąca) działanie gniazda rozszerzeń
- *Tabela dodatkowa* - wywołanie tabeli dodatkowej
- *Formularz tabeli* - wywołanie formularza tabeli dodatkowej
- *IF (warunek) THEN ELSE* – instrukcja warunkowego wykonania

Funkcje w zależności od typu przyjmują różne parametry natomiast elementami wspólnymi dla wszystkich funkcji są (Rys 4):

- nazwa funkcji – obowiązkowa do wypełnienia nazwa własna definiowanej funkcji;
- opis - w którym możemy zawrzeć własny, dodatkowy komentarz do działań jakie realizuje funkcja;



Rys 4. Formularz edycji funkcji w gnieździe rozszerzeń

Zarówno nazwa jak i opis widoczne są w Edytorze gniazd rozszerzeń bez konieczności otwierania formularza. Wygląd formularza zmienia się w zależności od typu funkcji, którą wybiera się z listy dostępnych typów.

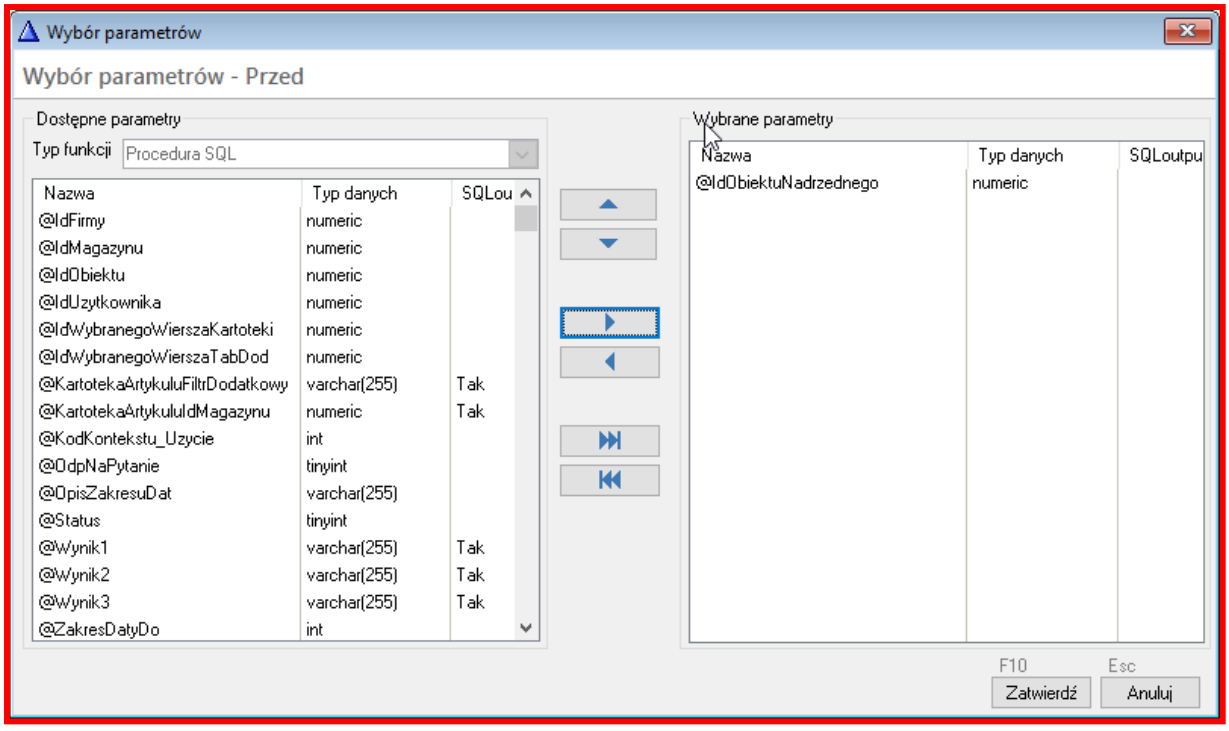
W kolejnych rozdziałach omówione zostaną parametry możliwe do przekazania w poszczególnych typach funkcji.

3.1 Procedura SQL

Procedura SQL (Rys 4) jest podstawową funkcją używaną w gniazdach. Aby możliwe było uruchomienie przez gniazdo procedury SQL spełnione muszą być dwa warunki:

1) procedura musi istnieć w bazie danych i musi posiadać parametry podawane w takiej kolejności i takich typów jakie zdefiniowano w polu „Lista parametrów";
2) nazwa procedury musi być w ustalonym formacie tzn. zaczynać się od ciągu znaków „MAGSRC_".

W polu „Lista parametrów" parametry procedury podaje się zgodnie ze składnią SQL tzn. oddziela przecinkami. Przy parametrach wyjściowych należy pamiętać o słowie kluczowym OUTPUT. Nie ma potrzeby wpisywania parametrów ręcznie – można skorzystać z podpowiedzi dostępnej pod przyciskiem „Wybierz zmienne aplikacji" (F4) , której uruchomienie skutkuje pojawieniem się okna wyboru parametrów (Rys 5).

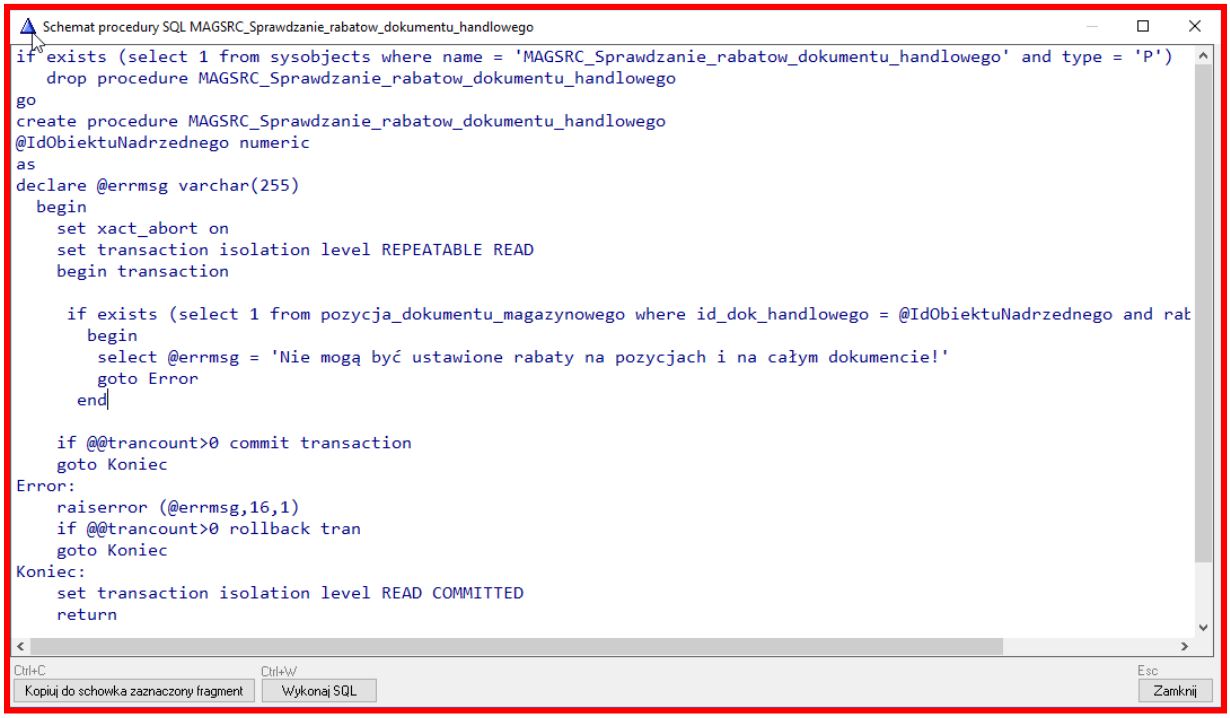


Rys 5. Wybór parametrów dla procedury SQL

Każda zmienna widoczna w oknie wyboru na liście „Dostępne parametry" może być użyta jako parametr funkcji. Obok nazwy zmiennej uwidoczniony jest jej typ oraz w kolumnie „SQL output" wskaźnik czy zmienna jest wejściowo-wyściowa w gnieździe tzn. czy możliwa jest jej zmiana wartości w procedurze oraz przekazanie tej wartości dalej do gniazda. Tylko za pomocą procedury SQL można zmienić wartość zmiennej pod warunkiem, że jest ona typu „SQL output". Zmienne z listy „Dostępne parametry" przenosimy przyciskami pomiędzy listami do wykazu „Wybranych parametrów". Ważna jest kolejność wybranych parametrów (którą możemy w dowolnym momencie zmienić) ponieważ zmienne będą przekazywane do procedury w takiej kolejności jak widać na liście „Wybrane parametry". Skompletowaną listę potrzebnych parametrów zatwierdzamy F10 co generuje w polu „Lista parametrów" formularza edycji funkcji zgodną ze składnią SQL listę zmiennych i ich typów oddzielonych przecinkami.

Po zdefiniowaniu listy parametrów przez gniazdo procedury SQL możemy podejrzeć sugerowany

schemat tworzenia takich procedur, który został opracowany przy założeniu, że procedury będą zmieniały dane w bazie (stąd ujęcie kodu procedury w transakcję) (Rys 6).



Rys 6. Przykład zmodyfikowanego schematu procedury SQL

Klawiszem F3 uruchamiamy edytor zawierający schemat procedury. Schemat jest od razu dostosowany do definicji funkcji tzn. zawiera nazwę i parametry procedury SQL, które zostały wcześniej zdefiniowane (pole „nazwa" i „lista parametrów"). Schemat można dowolnie poprawić w edytorze i umieścić w bazie danych przyciskiem „Wykonaj SQL". Zaleca się jednak aby schemat skopiować do schowka i dalej poddać wypełnieniu kodem, który procedura ma realizować w zewnętrznym edytorze SQL (np. SQL Management Studio).

W większości wypadków procedura SQL wywoływana jest przez gniazdo jednokrotnie dla zdefiniowanej listy parametrów. Jeśli chcemy wywołać ją ponowne musimy jeszcze raz zdefiniować w gnieździe jej wywołanie. Wyjątkiem są sytuacje gdy chcemy oprogramować wykonanie wielokrotne procedury dla zaznaczonych obiektów w WAPRO Mag. W takim przypadku należy zaznaczyć opcję „wykonaj w pętli dla zaznaczonych obiektów" jak również przekazać jako jeden z parametrów zmienną @IdObiektu, która będzie się zmieniać w kolejnych iteracjach. Trzeba pamiętać, że kod obsługi musimy napisać dla pojedynczego @IdObiektu a nie dla grupy zaznaczonych.

Należy pamiętać o specyficznym sposobie obsługi błędów sygnalizowanych w procedurze SQL (błędów wykonania SQL). Gniazda automatycznie po odebraniu sygnalizacji błędu wyświetlają komunikat o błędzie (zwracany z SQL funkcją raiserror) i przerywają dalsze wykonanie kodu zarówno umieszonego w gnieździe rozszerzeń jak i standardowego kodu programu, którego wykonanie nastąpiłoby normalnie po zakończeniu kodu gniazda (więcej nt. obsługi błędów w rozdziale 6.1).

3.2 Skrypt VBS

Wykonanie skryptu VBS jest możliwe tylko jeśli w środowisku systemu operacyjnego dostępny jest procesor skryptów VBS czyli program wscript.exe. Standardowo w systemach XP i Vista procesor skryptów VBS powinien być dostępny.

W polu „Ścieżka do skryptu" należy wskazać lokalizację pliku skryptu VBS wraz z jego nazwą widoczną dla wszystkich stacji roboczych WAPRO Mag. W szczególnym przypadku może to być katalog instalacyjny programu (nie zalecane!).

W polu „Parametr 2" można przekazać wartości zmiennych dostępnych w gnieździe. Wartości będą oddzielone przecinkami i ujęte w cudzysłów zgodnie z zasadami przekazywania parametrów do programów w środowisku systemu operacyjnego.

Skrypt powinien również odbierać wartość parametrów połączenia do bazy danych (tzw. Connection String) przekazywaną jako parametr 1. Parametry połączenia podawane są w kolejności: nazwa serwera, nazwa bazy danych, użytkownik serwera, hasło użytkownika serwera oddzielone przecinkami i oczywiście ujęte w cudzysłowy.

Wartości zmienne oddzielone przecinkami najłatwiej w skrypcie VBS odebrać funkcją Split, która zwraca jednowymiarową tablicę wartości tekstowych z łańcucha znakowego, w którym wartości oddzielone są separatorami np. przecinkami.

3.3 Moduł wykonywalny

Moduły rozszerzeń muszą mieć postać wykonywalnych programów tzn. typu EXE lub BAT.

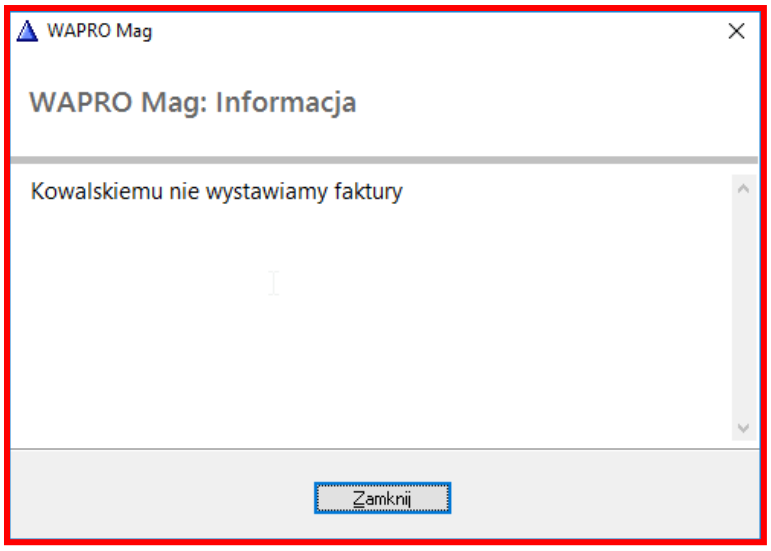
W polu „Ścieżka do modułu" należy wskazać lokalizację pliku modułu rozszerzeń wraz z jego nazwą widoczną dla wszystkich stacji roboczych WAPRO Mag. W szczególnym przypadku może to być katalog instalacyjny programu (nie zalecane!).

W polu „Parametr 2" można przekazać wartości zmiennych dostępnych w gnieździe. Wartości będą oddzielone przecinkami i ujęte w cudzysłów zgodnie z zasadami przekazywania parametrów do programów w środowisku systemu operacyjnego.

Moduł powinien również odbierać wartość parametrów połączenia do bazy danych (tzw. Connection String) przekazywaną jako parametr 1. Parametry połączenia podawane są w kolejności: nazwa serwera, nazwa bazy danych, użytkownik serwera, hasło użytkownika serwera oddzielone przecinkami i oczywiście ujęte w cudzysłowy.

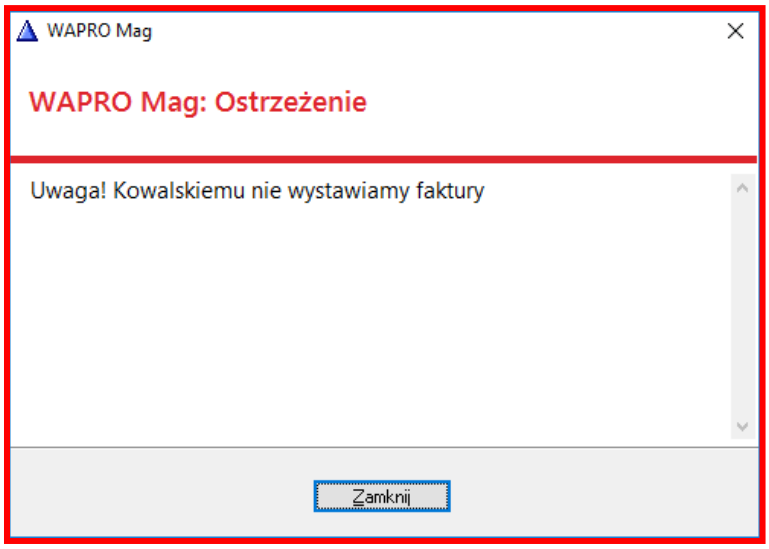
3.4 Komunikat

Komunikat wyświetla na ekranie okno dialogowe z przyciskiem OK. W polu „Treść komunikatu" można wstawić dowolny ciąg tekstowy lub wartość zmiennych wybranych przyciskiem „Wybierz zmienne aplikacji". Umieszczenie w gnieździe funkcji typu „Komunikat" i uruchomienie wykonania powoduje wstrzymanie dalszego przetwarzania kodu do czasu zamknięcia okna komunikatu. Komunikat domyślnie jest oknem informacyjnym (Rys. 7):



Rys 7. Okno komunikatu które działa jako Informacja

W przypadku użycia w treści komunikatu wykrzyknika „!" okno staje się ostrzeżeniem. (Rys. 8)



Rys 8. Okno komunikatu które działa jako ostrzeżenie.

3.5 Pytanie

Funkcja typu Pytanie wyświetla na ekranie okno dialogowe z przyciskami TAK i NIE. Podobnie jak w przypadku funkcji „Komunikat" możemy w treści pytania umieścić wartości dostępnych zmiennych w gnieździe, w którym uruchomione zostanie pytanie. Naciśnięcie przez użytkownika przycisku TAK w odpowiedzi na pytanie ustawia zmienną @OdpNaPytanie dostępną w gniazdach na wartość 1 zaś wybór przycisku NIE powoduje przypisanie wartości 0. Sprawdzenie wartości zmiennej i podjęcie dalszych działań umożliwia funkcja typu „IF (warunek) THEN ELSE" (instrukcja warunkowego wykonania – rozdział 3.9)

Funkcja typu Pytanie jest funkcją interakcji z użytkownikiem toteż jej wykonanie w gnieździe wstrzymuje wykonanie dalszej części kodu do czasu zamknięcia okna pytania.

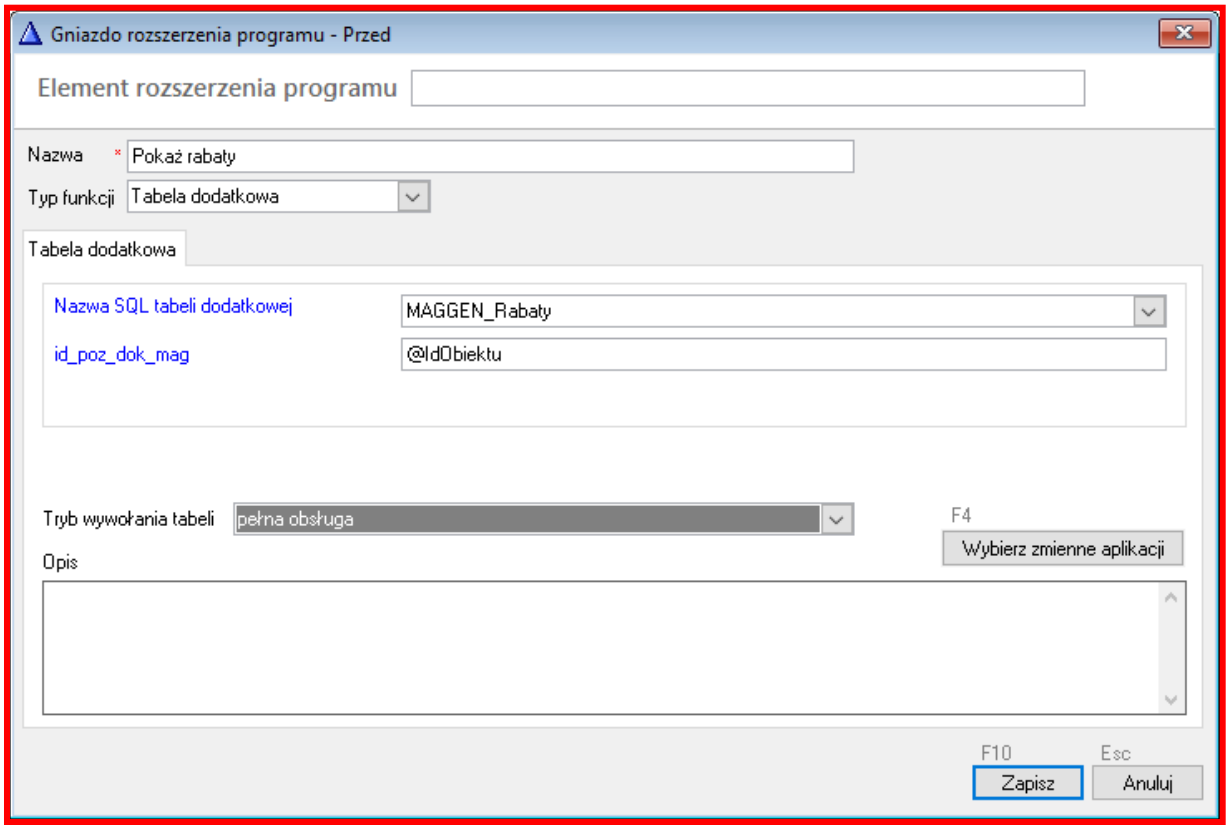
3.6 Koniec

Funkcja Koniec przerywa bezwarunkowo wykonanie kodu gniazda rozszerzeń (nie kończy działania aplikacji). W zależności od parametru statusu końca program po wyjściu z gniazda może realizować standardową funkcjonalność (status = 1 OK) lub nie wykonywać standardowych funkcji i powrócić do miejsca, które aktywowało wykonanie gniazda (status = 0 Przerwij).

3.7 Tabela dodatkowa

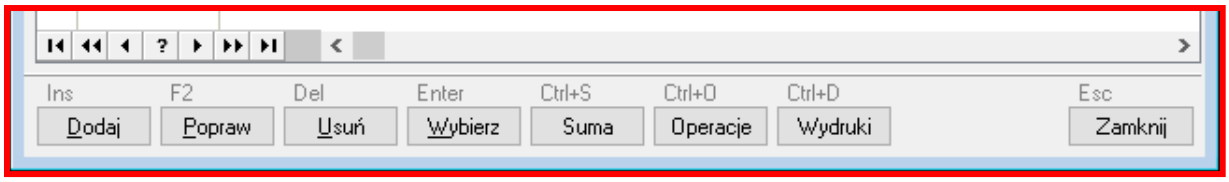
Funkcja typu Tabela dodatkowa umożliwia wywołanie w kodzie gniazda okna listy wierszy tabeli dodatkowej zdefiniowanej wcześniej w programie. W polu „Nazwa SQL tabeli dodatkowej" należy wybrać z listy nazwę serwerową tabeli . Następnie trzeba podać wartość identyfikatora pola zależnego od kontekstu tabeli, w którym została zdefiniowana (czyli klucza obcego tabeli). Przykładowo jeśli tabela została zdefiniowana w kontekście firmy należy podać wartość identyfikatora id_firmy. Jeśli natomiast w kontekście pozycji dokumentu magazynowego, wtedy należy podać wartość identyfikatora pozycji id_poz_dok_mag (Rys 9), dla którego będą pokazywane zapisy w oknie listy wierszy tabeli. Na postawie kontekstu w jakim została zdefiniowana tabela, program sam sugeruje nazwę identyfikatora, który jest wymagany. Wartość identyfikatora można przekazać jako wartość stałą (liczbę) lub wartość zmiennej dostępnej w gnieździe. W drugim przypadku nie ma znaczenia typ zmiennej tzn. czy zmienna jest tekstowa czy liczbowa – ważne aby zawierała wartość numeryczną, którą można skonwertować na wartość potrzebnego identyfikatora.

Wykonanie w gnieździe funkcji typu Tabela dodatkowa spowoduje wywołanie okna listy wierszy zdefiniowanej tabeli, zaś program będzie oczekiwał z dalszym wykonaniem kodu gniazda do czasu zamknięcia tego okna.



Rys 9. Formularz definicji funkcji typu Tabela dodatkowa

W trakcie definiowania możemy także ustalić działanie klawiszy funkcyjnych dla tabeli :



Rys 10. Przykładowe klawisze funkcyjne w tabeli dodatkowej.

i mamy do wyboru opcje

a) Pełna obsługa – wszystkie klawisze są aktywne

b) Tylko podgląd danych bez wyboru wiesza – aktywny tylko klawisz „Zamknij"

c) Tylko wybór wiersza bez aktywnych przycisków zmiany danych – tylko aktywny klawisz „Wybierz" pozostałe klawisze nie są aktywne.

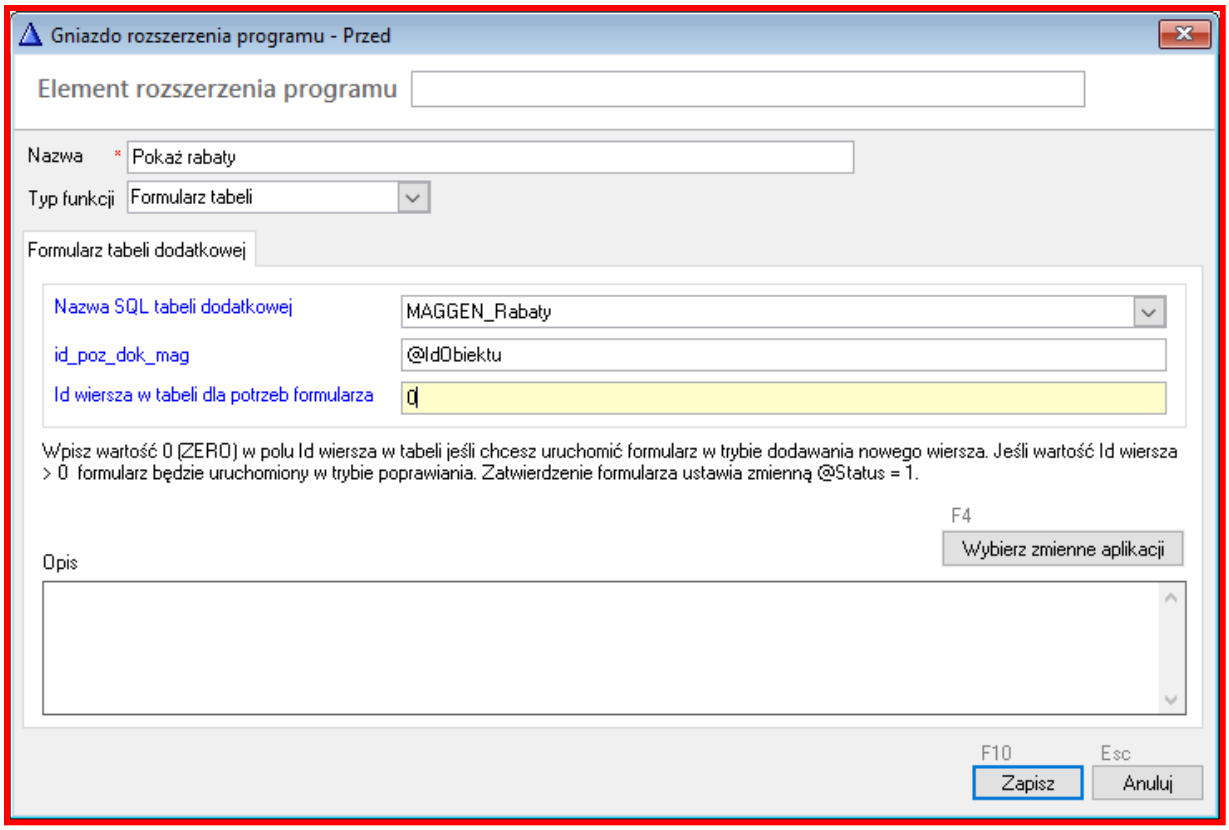
d) Nieaktywny przycisk wyboru wiersza – nieaktywny klawisz „Wybierz", pozostałe klawisze są aktywne.

3.8 Formularz tabeli

Funkcja typu Formularz tabeli (Rys 11) umożliwia wywołanie w kodzie gniazda okna formularza edycji wiersza tabeli dodatkowej. Pierwsze dwa parametry tzn. „Nazwa SQL tabeli dodatkowej" i wartość identyfikatora pola zależnego od kontekstu tabeli (klucza obcego tabeli) definiuje się identycznie jak przy funkcji typu Tabela dodatkowa. Trzecim parametrem jest „Id wiersza tabeli dla potrzeb formularza", w którym podajemy wartość kolumny IDENTYFIKATOR generowanej automatycznie podczas definiowania tabeli dodatkowej.

Formularz może być wywołany w trybie dodawania lub poprawiania. Wstawienie wartości 0 (lub zmiennej zwracającej wartość zero) spowoduje, że formularz będzie uruchomiony w trybie dodawania wiersza do tabeli dodatkowej. Wstawienie wartości większej niż 0 spowoduje uruchomienie formularza w trybie poprawiania wiersza o identyfikatorze wiersza równym podanej wartości.

Zatwierdzenie formularza skutkuje ustawieniem wartości zmiennej @Status dostępnej w gnieździe na 1 zaś anulowanie – ustawieniem wartości 0. Można więc za pomocą warunku @Status > 0 w instrukcji warunkowego wykonania kodu sprawdzić czy użytkownik zatwierdził czy anulował formularz i uzależnić od tego sposób dalszego wykonania kodu gniazda.



Rys 11. Formularz definicji funkcji typu Formularz tabeli

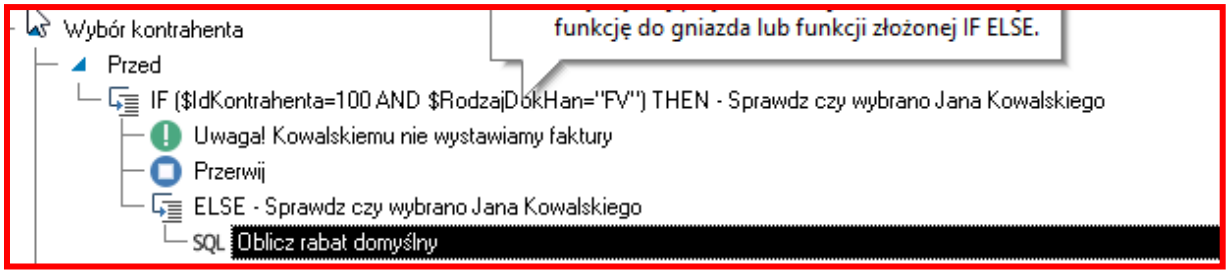
3.9 IF (warunek) THEN ELSE

Instrukcja warunkowego wykonania kodu gniazda IF (warunek) THEN ELSE umożliwia sterowanie wykonaniem funkcji zdefiniowanych w gnieździe w zależności od spełniania lub nie warunku podanego jako jej parametr. Warunek musi zwracać logiczną wartość PRAWDA lub FAŁSZ. Warunek, którego sprawdzenie zwraca wartość PRAWDA, powoduje przeniesienie sterowania do miejsca oznaczonego w edytorze gniazd rozszerzeń THEN i wykonanie sekwencji funkcji zaś zwrócenie wartości FAŁSZ (nie spełnienie warunku) powoduje, że wykonane zostaną funkcje po ELSE zaś te, które zostały umieszczone po THEN zostaną pominięte.

Instrukcja warunkowego wykonania jest funkcją złożoną tzn. jej umieszczenie w gnieździe powoduje powstanie dwóch gałęzi drzewa. Do gałęzi zawierającej w nazwie THEN dołączamy funkcje, których wykonanie będzie poprawne w przypadku spełnienia warunku zaś do gałęzi zawierającej w nazwie ELSE dołączyć należy funkcje, które powinny wykonać się gdy warunek nie jest spełniony.

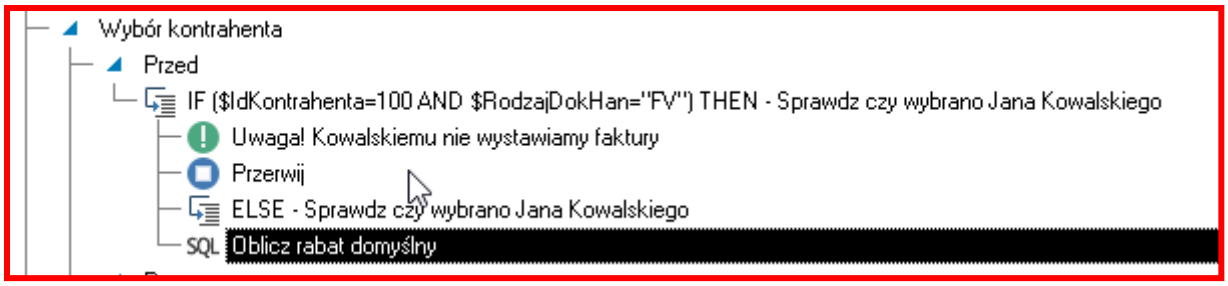
Nie ma znaczenie poziom, na którym funkcje po ELSE są dołączone tzn.

wykonanie funkcji pokazanych na Rys 12:



Rys 12. Instrukcja warunkowego wykonania – funkcja podpięta do ELSE

spowoduje efekt identyczny jak wykonanie funkcji pokazanych na Rys 13.



Rys 13. Instrukcja warunkowego wykonania – funkcja na tym samym poziomie co ELSE

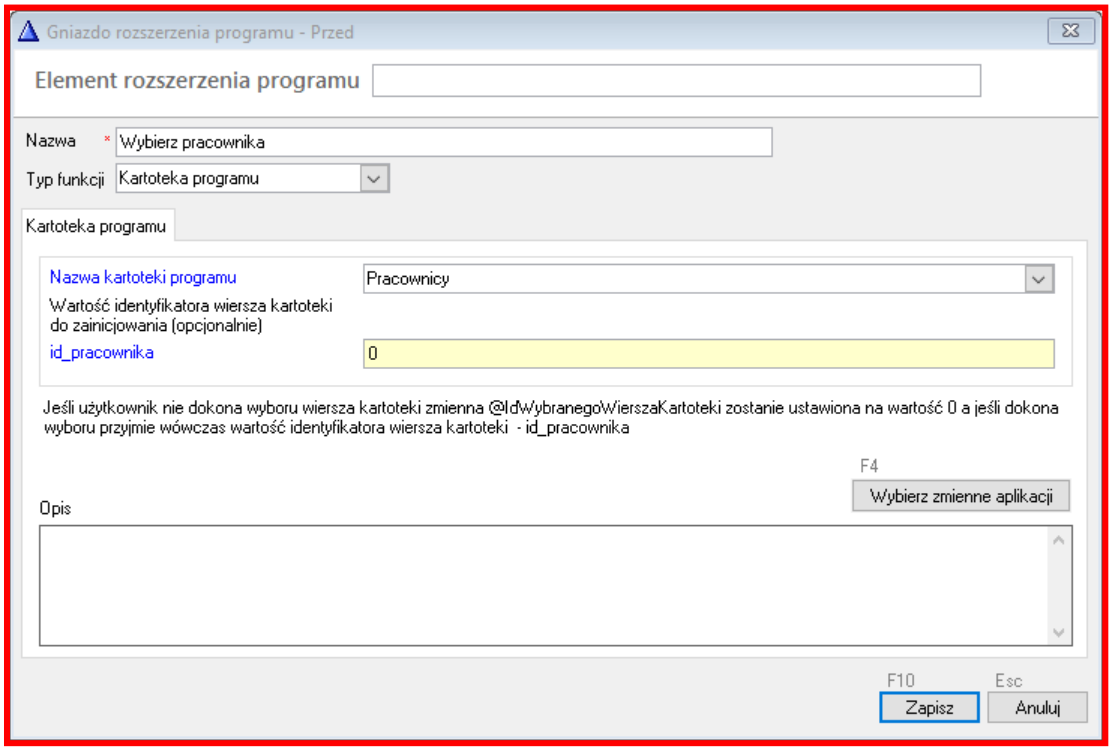
Warunek buduje się w oparciu o zmienne. Wartości zmiennych można porównywać ze stałymi lub innymi zmiennymi za pomocą standardowych operatorów >,<,= itp.

Poszczególne elementy w warunku można łączyć operatorami logicznymi OR, AND. Stałe tekstowe należy ująć w cudzysłów.

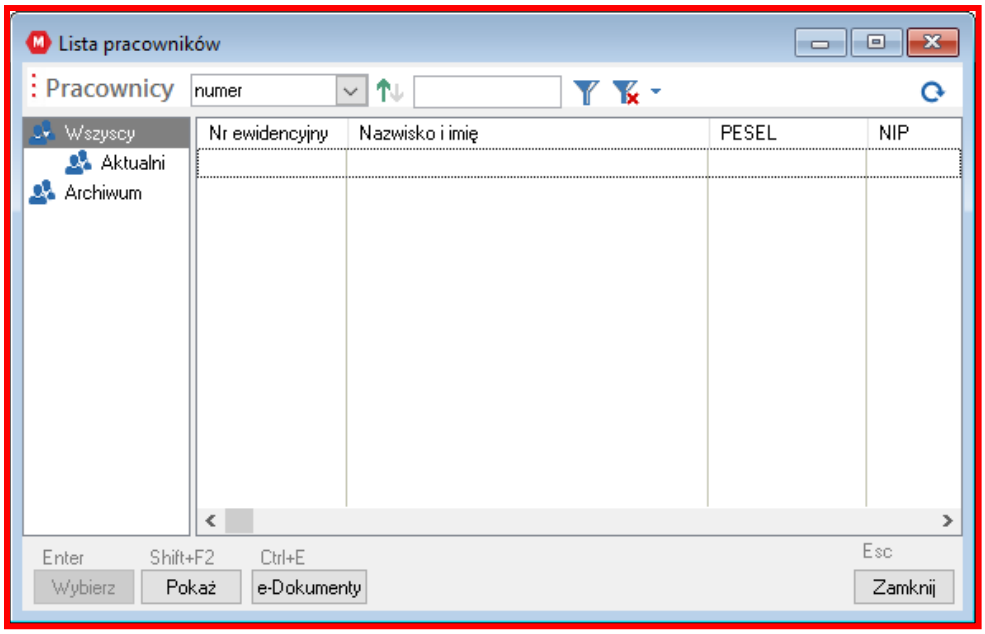
3.10 Kartoteka programu.

Instrukcja ta pozwala wywołać jedną z dostępnych kartotek programu. Każda z kartotek wygląda identyczne jak okna pracujące standardowo w aplikacji z tym że w oknie tym nie działają klawisze edycyjne – dostępny jest tylko klawisz „Wybierz" oraz „Zamknij" (Rys 15) . Po naciśnięciu klawisza „Wybierz" uzupełniana jest zmienna @IdWybranegoWierszaKartoteki na podstawie identyfikatora rekordu kartoteki. Naciśnięcie klawisza zamknij ustawia w zmiennej wartość 0.

Wybierając w gnieździe kartotekę pracowników zmienna przyjmie informacje id_pracownika.



Rys 14. Przykład definicji kartoteki pracowników.



Rys 15. Przykład wywołanej kartoteki

Aktualnie mamy dostępne kartoteki:

a) Artykuły
b) Kontrahenci
c) pracownicy
d) Magazyny
e) Kategorie artykułu

4. Zmienne (parametry funkcji) gniazd rozszerzeń

Dla potrzeb parametrów przekazywanych do funkcji można wykorzystać 2 rodzaje zmiennych:

- zmienne systemowe - których nazwa zaczyna się od znaku @;
- zmienne kontekstowe – których nazwa zaczyna się od znaku $;

Sposób wykorzystania zmiennych w funkcjach jest identyczny i nie zależy od rodzaju zmiennych. To co odróżnia ww. rodzaje zmiennych to fakt, że zmienne systemowe występują zawsze w każdym gnieździe programu natomiast zmienne kontekstowe dostępne są tylko w tych gniazdach, w których mogą być sensownie wykorzystane.

Przykładowo zmienna $Brutto_Netto przechowująca informacje o tym czy dokument liczony jest od brutto czy od netto pojawia się w gnieździe o adresie *Dokumenty handlowe\Okno danych dokumentu handlowego\Dodawanie lub poprawianie dokumentu\Zatwierdzenie danych dokumentu\Przed*. Nie znajdziemy jej natomiast w gniazdach związanych z kartoteką artykułów.

Listę dostępnych w danym gnieździe zmiennych znajdziemy pod klawiszem F4 („Wybierz zmienne aplikacji") na formularzu definicji funkcji (np. Rys 11). Okno „Wybór parametrów" (Rys 20) zawiera listę „Dostępne parametry", która prezentuje zmienne możliwe do wykorzystania w danym gnieździe i dla danego typu funkcji. Zmienne przechowują informacje o wartościach danych dostępnych w miejscu programu, w którym definiujemy funkcję. Generalnie zmienne umożliwiają tylko odczyt wartości. W przypadku funkcji typu Procedura SQL możliwa jest jednak zmiana wartości niektórych z nich. O tym czy

można zmienić wartość zmiennej w procedurze SQL decyduje kolumna „SQL Output" listy „Dostępnych zmiennych". Zapis „Tak" w ww. kolumnie oznacza, że zmienna będzie przekazywana do procedury składowanej ze słowem OUTPUT co w języku SQL oznacza, że zmiana jej wartości wewnątrz procedury spowoduje przekazanie tej zmiany do gniazda. Zmienne kontekstowe powiązane są z danymi w formularzach toteż zmiana ich wartości skutkuje zmianą danych w programie np. zmiana wartości zmiennej $CenaNettoPLN w funkcji podpiętej w gnieździe o adresie *Dokumenty handlowe\Okno danych pozycji dokumentu handlowego\Dodawanie pozycji\Otwarcie okna danych pozycji\Przed* spowoduje, że cena netto pokazana w formularzu po otwarciu okna danych pozycji dokumentu będzie taka jak wartość zmiennej $CenaNettoPLN a nie domyślna dla artykułu jakby wynikało ze standardowego zachowania programu.

Z kolei wpisanie w formularzu nowej wartości ceny netto spowoduje przepisanie tej ceny do zmiennej $CenaNettoPLN więc odczyt jej wartości w gnieździe *Dokumenty handlowe\Okno danych pozycji dokumentu handlowego\Dodawanie pozycji\Zatwierdzenie danych pozycji\Przed* pokaże cenę taką jak na formularzu.

Osobną rolę w gniazdach spełniają zmienne systemowe. Ich pełna lista wraz z objaśnieniem znajduje się w poniższej tabeli.

Nazwa zmiennej	Typ wartości (SQL)	Znaczenie
@TekstBledu	varchar(255)	przechowuj tekst błędu ostatnio wykonanej funkcji np. w przypadku procedury SQL będzie to tekst zwracany funkcją *raiseerror*
@BylBlad	tinyint	zawiera wartość 1 jeśli funkcja zakończyła się błędem i 0 jeśli jej wykonanie przebiegło poprawnie
@IdFirmy	numeric	id_firmy w kontekście której pracuje program
@IdMagazynu	numeric	id_magazynu w kontekście którego pracuje program
@IdObiektu	numeric	w zależności od umiejscowienia gniazda jest to identyfikator dokumentu (zapisu w kartotece) lub pozycji dokumentu – analogicznie jak w raportach Crystal Reports oraz operacjach dodatkowych
@IdUzytkownika	numeric	id_użytkownika aktualnie pracującego w WAPRO Mag (wykonującego funkcję w gnieździe)
@ZakresDatyOd	int	data w formacie WAPRO Mag początku ustawionego zakresu dat w programie
@ZakresDatyDo	int	data w formacie WAPRO Mag końca ustawionego zakresu dat w programie
@OpisZakresuDat	varchar(255)	opis zakresu dat, który w programie jest widoczny na dolnej belce statusowej

@KodKontekstu_Uzycie	int	wartość pola UZYCIE tabeli ZAZNACZONE ustawiana przez program automatycznie przy zaznaczaniu elementów list w zależności od miejsca listy w programie
@Wynik1	varchar(255)	zmienna dla programisty gniazd – do dowolnego wykorzystania w funkcjach
@Wynik2	varchar(255)	zmienna dla programisty gniazd – do dowolnego wykorzystania w funkcjach
@Wynik3	varchar(255)	zmienna dla programisty – do dowolnego wykorzystania w funkcjach
@OdpNaPytanie	tinyint	przechowuje stan ostatniej odpowiedzi na pytanie – wartość 1 oznacza, że wybrano odpowiedź TAK na oknie dialogowym
@IdObiektuNadrzednego	numeric	analogicznie jak @IdObiektu z tym, że wszędzie tam gdzie @IdObiektu będzie równe id pozycji dokumentu to @IdObiektuNadrzednego będzie równe identyfikatorowi dokumentu. Dla dokumentów @IdObiektu = @IdObiektuNadrzednego lub @IdObiektuNadrzednego = 0
@Status	tinyint	przechowuje wartość 1 jeśli ostatnio uruchamiany w gnieździe formularz tabeli dodatkowej został zatwierdzony i wartość 0 jeśli została anulowany
@IdWybranegoWierszaKartoteki	numeric	Zmienna przechowuje identyfikator rekordu dla rekordu z wywołanej kartoteki programu
@IdWybranegoWierszaTabDod	numeric	Zmienna przechowuje identyfikator rekordu dla rekordu z wywołanej tabeli dodatkowej
@KarotekaArtykuluFiltrDodatkowy	varchar(255)	Zmienna ta pozwala dodać dodatkowy filtr do wywołanej kartoteki programu. Pozwala ona rozszerzyć warunek where dla wyświetlanej kartoteki
@KartotekaArtykuluIdMagazynu	numeric	Zmienna, która pozwala zmienić identyfikator magazynu (z bieżącego dla zalogowanego użytkownika na inny) przed wywołaniem kartoteki programu

Zmienne systemowe można więc podzielić na 2 podgrupy:

1) zmienne przechowujące aktualne ustawienia środowiska WAPRO Mag *(@IdFirmy, @IdMagazynu, @IdObiektu, @IdUzytkownika, @ZakresDatyOd, @ZakresDatyDo, @OpisZakresuDat, @KodKontekstu_Uzycie, @IdObiektuNadrzednego*);
2) zmienne przechowujące dane zwracane przez określone typy funkcji lub obsługujące sygnalizację błędu *(@TekstBledu, @BylBlad, @Wynik1, @Wynik2, @Wynik3, @OdpNaPytanie, @Status*).

Na szczególną uwagę zasługują zmienne @TekstBledu i @BylBlad co wynika ze sposobu obsługi błędów w gniazdach rozszerzeń. Standardowo po odebraniu sygnalizacji błędu wygenerowanego przez funkcję wyświetlany jest komunikat o błędzie (zwracany np. z procedury SQL funkcją raiserror) a następnie przerywane jest dalsze wykonanie kodu zarówno umieszonego w gnieździe rozszerzeń jak i kodu programu, którego wykonanie nastąpiłoby normalnie po zakończeniu wykonania wszystkich funkcji w gnieździe. Zmienna @BylBlad ustawiana jest na wartość 1 a komunikat błędu zapisany w zmiennej @TekstBledu. Jeśli jednak następna funkcja w kolejności do wykonania użyje jako parametrów jednej z ww. zmiennych (np. w funkcji IF @BylBlad>0 THEN ELSE lub Komunikat wyświetlający @TekstBledu) wówczas program przyjmuje, że obsługę błędu przejmuje programista gniazda i wykonanie dalszego kodu gniazda nie jest przerywane a także nie jest wyświetlany standardowy komunikat błędu (patrz Przykład 2). Możemy więc całkowicie przechwytywać błędy generowane przez funkcje umieszczone w gniazdach i np. rejestrować je w tabeli dodatkowej zamiast wyświetlać. Można również oprogramować różne działania w zależności od poprawnego lub niepoprawnego zakończenia funkcji.

Należy pamiętać, że wartości wszystkich zmiennych dostępne są jedynie w obrębie gniazda. W innym gnieździe, nawet jeśli występują te same zmienne, ich wartości albo wynikają z formularzy (w przypadku zmiennych kontekstowych) albo z ustawień aplikacji (w przypadku zmiennych systemowych). Zmienne przewidziane dla programistów (np. @Wynik1) oraz zmienne obsługi błędów i statusów funkcji (@Status, @OdpNaPytanie, @BylBlad) są zainicjowane wartościami domyślnymi (0 w przypadku wartości numerycznych i pusty ciąg znaków w przypadku wartości tekstowych). Nie można więc ustawić wartości jakiejś zmiennej np. @Wynik1 w jednym gnieździe i odebrać tą wartość w innym gnieździe. Jeśli chcemy przekazać jakieś parametry między funkcjami uruchamianymi w różnych gniazdach najlepiej zapisać wartości tych parametrów w bazie danych np. w tabeli dodatkowej.

Na szczególną uwagę zasługują zmienne kontekstowe $CzyDodano, $CzyPoprawiono, $CzyUsunieto , które występują w gniazdach typu „Po" na listach dokumentów, artykułów czy kontrahentów. Pokazują one status wykonania operacji, która miała miejsce po wykonaniu określonego działania na liście. Przykładowo w gnieździe o adresie *Dokumenty handlowe\Lista dokumentów handlowych\Zdarzenia związane z pojedynczym dokumentem\Poprawianie dokumentu\Po* zmienna $CzyPoprawiono ustawiona na 1 oznacza, że dokument, którego identyfikator znajduje się w zmiennej @IdObiektu, został poprawiony. Wartość 0 będzie oznaczała, że zmiany zostały wycofane i dokument nie był zmodyfikowany. Analogicznie w przypadku dodawania dokumentów zmienna $CzyDodano = 1 będzie oznaczała, że dokument został dodany a jego identyfikator będzie zawarty w zmiennej @IdObiektu. Z kolei jeśli dokument nie został dodany ($CzyDodano = 0) zmienna @IdObiektu nie ulegnie zmianie tzn. będzie zawierała identyfikator dokumentu podświetlonego na liście dokumentów.

Gniazda rozszerzeń dokonują automatycznie konwersji typów zmiennych. Nie ma więc potrzeby stosować konwersji typów aby wyświetlać wartości numeryczne w funkcjach typu komunikat czy pytanie. Należy jednak pamiętać, że takich konwersji trzeba dokonywać korzystając ze zmiennych @Wynik1, @Wynik2, @Wynik3, które są typu tekstowego w przypadku gdy podstawiamy wartości numeryczne do tych zmiennych w procedurach SQL. Ma to szczególnie znaczenie gdy projektowane rozwiązania mają być uniwersalne i stosowane na różnych wersjach MS SQL Server.

Ważne:

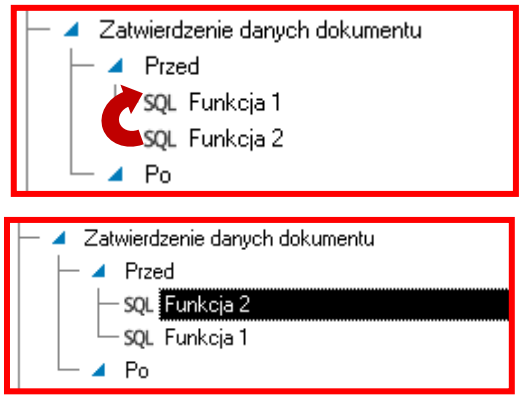
Podczas pracy z czystą bazą danych lub z programem po aktualizacji, zanim rozpoczniemy pracę z gniazdami i zaczniemy je wypełniać, zalecane jest wykonanie dowolnych działań w oknach, z którymi zamierzamy pracować. Następnie należy wylogować się z programu i ponownie zalogować.

Wynika to z zastosowanego mechanizmu przypisywania zmiennych do gniazd. Informacje te nie są tworzone i zapisywane podczas procesu instalacji (aktualizacji) bazy programu. Dane te zapisywane są sukcesywnie dopiero podczas pracy programem ponieważ przy liczbie około 700 gniazd rozszerzeń ilość wierszy w tabeli przechowującej przypisania zmiennych do gniazd sięga kilkunastu tysięcy. Zapisywanie tych informacji w trakcie instalacji (aktualizacji) znacznie wydłużyłoby jej czas.

Wobec tego dopiero otwieranie kolejnych okien powoduje sprawdzenie czy zmienne są już zarejestrowane w gniazdach dostępnych na tych oknach. Jeśli zmienna nie figuruje na liście zmiennych zarejestrowanych dla gniazda to jest dodawana do bazy danych i do listy rejestrowej zmiennych w programie. Lista ta (ograniczona kontekstem gniazda) prezentowana jest pod klawiszem „Wybierz zmienne”.

5. Edytor gniazd rozszerzeń – praca z funkcjami.

Każda kolejna funkcja dodawana w gnieździe jest umieszczana po poprzedniej. Aby zmienić ich kolejność wystarczy trzymając wciśnięty lewy przycisk myszy przesunąć daną funkcję na miejsce gdzie chcemy ją umieścić. Jeśli chcemy zamienić „Funkcje 2" miejscami z „Funkcją 1" (Rys 16) to musimy przytrzymując lewy przycisk myszy wskazać „Funkcję 1".



Rys 16. Nawigacja myszą w edytorze – zmiana kolejności funkcji

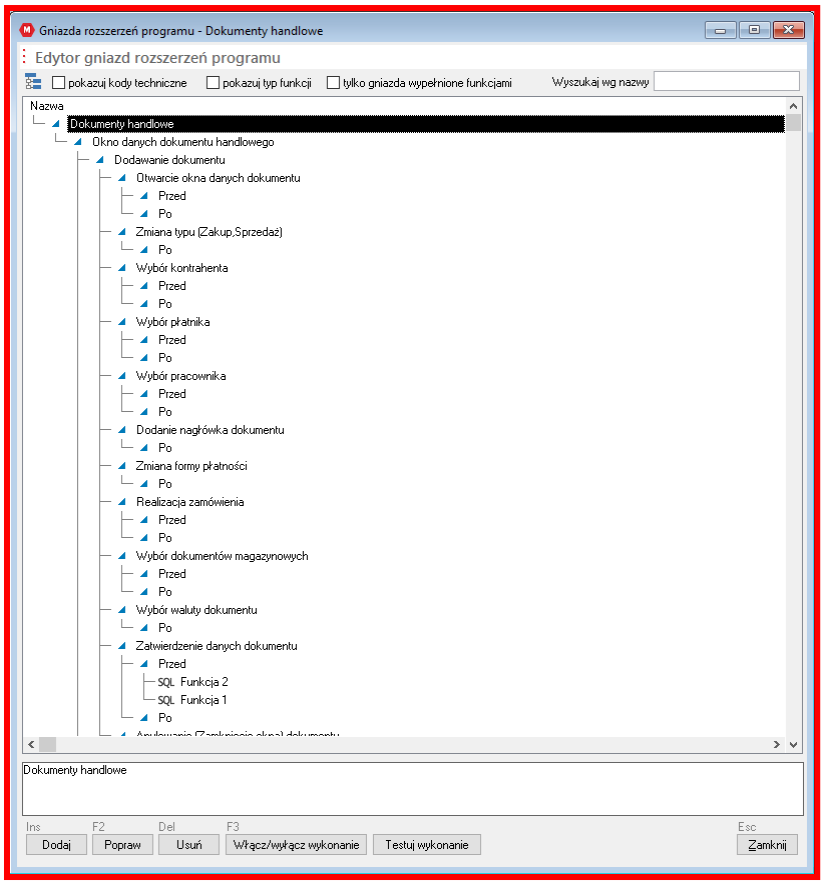
W programie można również skopiować funkcję z jednego gniazda do drugiego. Aby wykonać kopię funkcji wystarczy trzymając wciśnięty lewy przycisk myszy oraz klawisz SHIFT przemieścić kursor myszy z miejsca gdzie znajduje się funkcja do nowego gniazda czyli w miejsce gdzie chcemy ją umieścić (Rys 16). Należy przy tym pamiętać, że w nowym gnieździe nie wszystkie zmienne wykorzystywane jako parametry w kopiowanej funkcji mogą być dostępne. Najlepiej upewnić się, że wszystkie zmienne są dostępne w nowym miejscu poprzez edycję parametrów funkcji oraz wykonanie jej testu działania. W analogiczny sposób można skopiować zawartość całego gniazda rozszerzeń (czyli grupy funkcji) do nowego gniazda z tym, że kursorem myszy należy zaznaczyć gniazdo skąd chcemy skopiować funkcje a następnie trzymając lewy przycisk myszy oraz klawisz SHIFT wskazać gniazdo docelowe.

Z uwagi na fakt, że ilość gniazd rozszerzeń jest bardzo duża wygodniej nawigować pomiędzy gniazdami na poszczególnych ekranach. Przykładowo chcąc dodać rozszerzenia w gnieździe po zatwierdzeniu dokumentu należy otworzyć okno danych dokumentu i nacisnąć CTRL+SHIFT+F12. Uruchomi się Edytor pokazując tylko gniazda dostępne dla okna danych dokumentu handlowego (Rys 17).

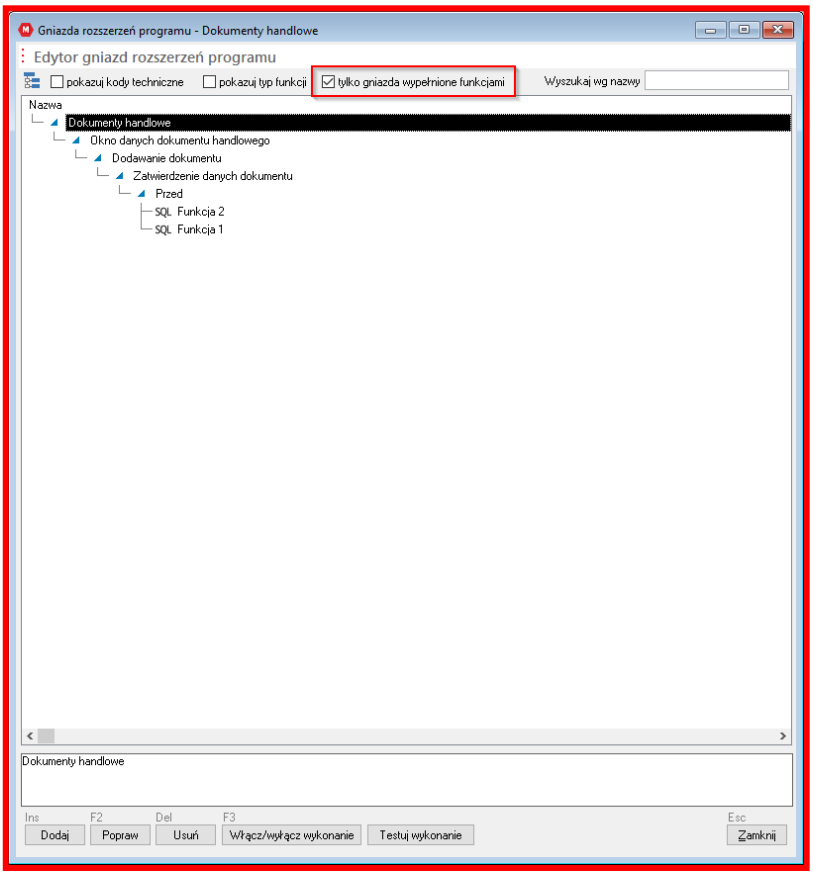
Strukturę drzewa gniazd rozszerzeń można również ograniczyć poprzez zastosowanie filtra „tylko gniazda wypełnione funkcjami" (Ctrl+F). Wówczas ukryte zostaną wszystkie puste gniazda tj. nie zawierające podpiętych funkcji (Rys 18).

Na ekranie edytora dostępne są jeszcze filtry wpływające na zawartość wyświetlanej informacji.

Filtr „pokazuj typ funkcji" przełącza wyświetlanie na liście informację o typie funkcji. Każda funkcja reprezentowana jest osobną ikoną (za wyjątkiem tabeli dodatkowej i formularza tabeli, które mają taka samą ikonę). Dodatkowy opis przydaje się szczególnie na początku pracy z edytorem gniazd ułatwiając zorientowanie się w strukturze podpiętych funkcji. Jedynie dla instrukcji złożonej IF (warunek) THEN ELSE typ funkcji wyświetlany jest zawsze niezależnie od ustawienia ww. filtra.



Rys 17. Gniazda rozszerzeń dostępne na oknie danych dokumentu handlowego



Rys 18. Włączony filtr „tylko gniazda wypełnione funkcjami”

Filtr „pokazuj kody techniczne” przełącza wyświetlanie na liście informację o kodzie technicznym gniazda lub funkcji. Kody techniczne gniazd jednoznacznie identyfikują gniazdo w strukturze. Są więc technicznym odpowiednikiem adresu gniazda. Z kolei dla funkcji kody generowane są automatycznie dla zapewnienia unikalności klucza w tabeli przechowującej dane gniazd i funkcji. Kody techniczne wykorzystywane są przede wszystkim przy tworzeniu skryptów SQL (Rozdział 10). oraz ewentualnie do szybkiego odszukania gniazda w strukturze.

Ustawienia filtrów „pokazuj kody techniczne”, „pokazuj typ funkcji” oraz „tylko gniazda wypełnione funkcjami” są zapamiętywane dla każdego użytkownika.

Dla potrzeb pracy z funkcjami na etapie projektowania zawartości gniazd rozszerzeń dodano przyciski „Włącz/wyłącz wykonanie” oraz „Testuj wykonanie”.

Przycisk „Włącz/wyłącz wykonanie” powoduje wyłączenie (lub włączenie) możliwości wykonania funkcji (gniazda). Zakres działania tego przycisku zależy od miejsca (pozycji) kursora w edytorze gniazd rozszerzeń i dotyczy wszystkich elementów podrzędnych drzewa. Funkcje lub gniazda, których wykonanie zostało wyłączone oznaczone są kolorem czerwonym. Ustawienie pozycji kursora na gałęzi SYSTEM i wyłączenie wykonania spowoduje więc wyłączenie wykonywania kodu wszystkich gniazd w programie.

Przycisk „Testuj wykonanie” umożliwia sprawdzenie działania wykonania sekwencji funkcji (lub pojedynczej funkcji). Wszystkie zmienne wykorzystane w kodzie gniazda są ustawione na wartości domyślne toteż docelowe działanie kodu gniazda może być jednak inne niż obserwowane w wyniku uruchomienia testowania. Szczególnie dotyczy to zachowania się gniazda w wyniku sprawdzenia warunku w instrukcji IF (warunek) THEN ELSE.

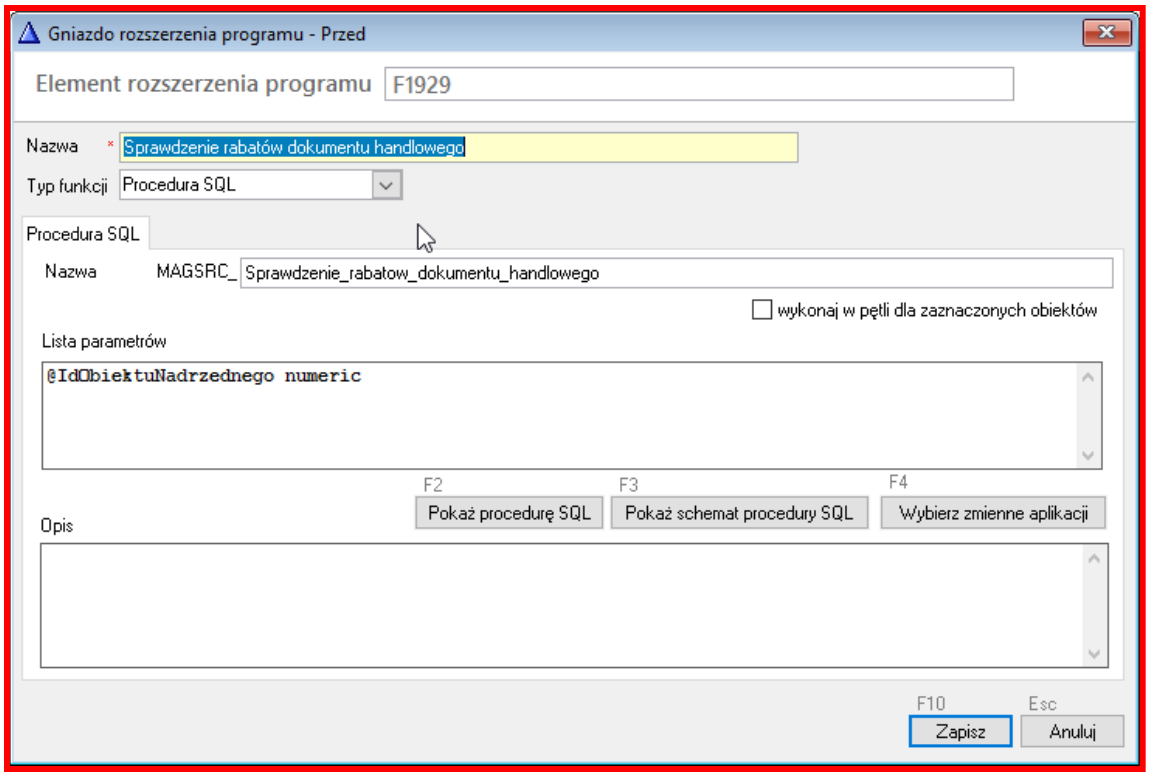
6. Przykłady zastosowań gniazd rozszerzeń.

6.1 Przykład 1

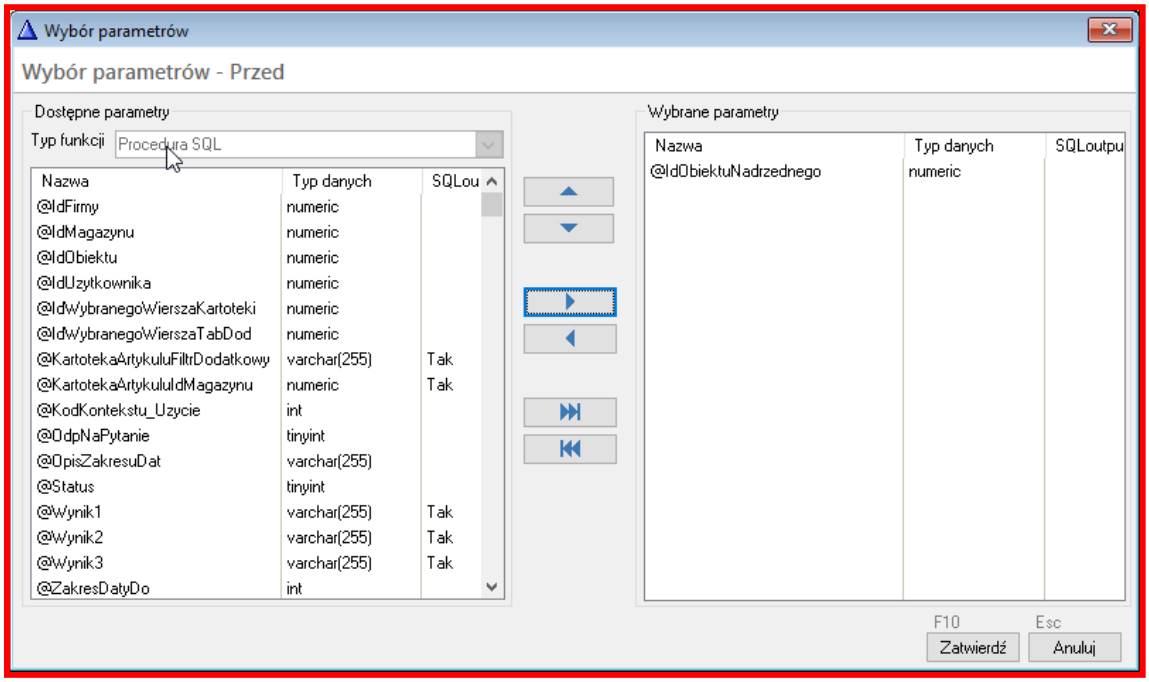
Chcielibyśmy kontrolować czy w dokumencie handlowym są ustawione rabat dla pozycji i całego dokumentu. Jeśli rabaty są ustawione dla jakiejkolwiek pozycji - dokument nie może być zatwierdzony i musi pojawić się stosowne ostrzeżenie.

Aby oprogramować ww. zadanie należy dodać przed zatwierdzaniem dokumentu w gnieździe o adresie *Dokumenty handlowe\Okno danych dokumentu handlowego\Dodawanie lub poprawianie dokumentu\Zatwierdzenie danych dokumentu\Przed* funkcję typu „Procedura SQL (3.1)” jak pokazano na (Rys 19)

Przekazujemy IdObiektuNadrzednego, które w tym miejscu programu przyjmie wartość id_dokumentu handlowego. Z listy dostępnych zmiennych uruchomionej klawiszem F4 („Wybierz zmienne aplikacji”) wybieramy IdObiektuNadrzednego i przesuwamy do listy „Wybrane parametry” (Rys 20) a następnie zatwierdzamy tak przygotowaną listę parametrów.



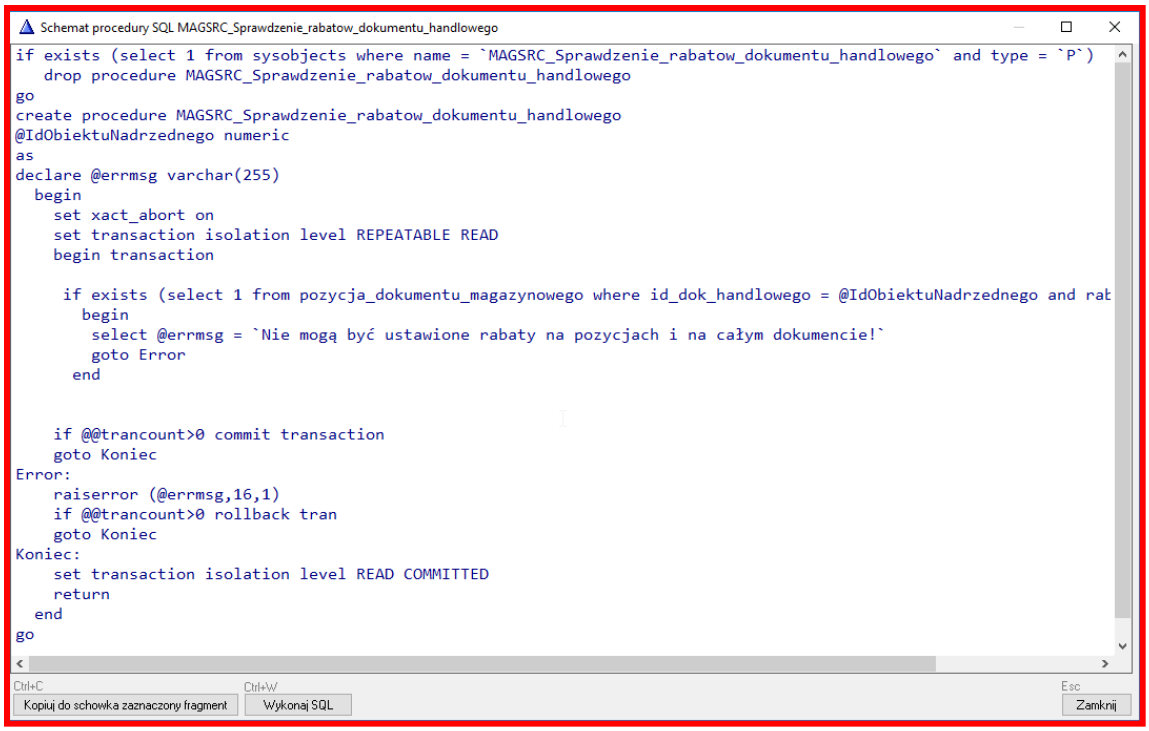
Rys 19. Dodawanie funkcji typu Procedura SQL w gnieździe



Rys 20. Wybór parametrów dla funkcji

Następnie przyciskiem F3 (Rys 21) uruchamiamy edytor schematu procedury SQL i w proponowanym standardowo schemacie zmieniamy kod SQL tak aby doprowadzić do poniższej postaci (kursywą zaznaczono fragment, który trzeba dopisać do schematu):

```
if exists (select 1 from sysobjects where name = 'MAGSRC_Sprawdzanie_rabatow_dokumentu_handlowego'
and type = 'P')
  drop procedure MAGSRC_Sprawdzanie_rabatow_dokumentu_handlowego
go
create procedure MAGSRC_Sprawdzanie_rabatow_dokumentu_handlowego
@IdObiektuNadrzednego numeric
as
declare @errmsg varchar(255)
 begin
  set xact_abort on
  set transaction isolation level REPEATABLE READ
  begin transaction

   if exists (select 1 from pozycja_dokumentu_magazynowego where id_dok_handlowego =
@IdObiektuNadrzednego and rabat2<>0 and rabat<>0)
    begin
     select @errmsg = 'Nie mogą być ustawione rabaty na pozycjach i na całym dokumencie!'
     goto Error
    end

  if @@trancount>0 commit transaction
  goto Koniec
Error:
  raiserror (@errmsg,16,1)
  if @@trancount>0 rollback tran
  goto Koniec
Koniec:
  set transaction isolation level READ COMMITTED
  return
 end
go
```



Rys 21. Schemat procedury SQL proponowany automatycznie po zmianach z przykładu 1

Aby umieścić kod procedury SQL w postaci takiej jak na Rys 21 należy wykonać skrypt SQL (Ctrl+W) lub skopiować kod z edytora do schowka (Ctrl+C) a następnie np. za pomocą SQL Management Studio uruchomić wykonanie wklejonego ze schowka skryptu.

Na zakończenie pozostaje zatwierdzić formularz dodawania funkcji (Rys 19) przyciskiem F10 i zamknąć edytor gniazd rozszerzeń. Następnie można od razu (bez konieczności przelogowania się) sprawdzić czy kontrola rabatów działa poprzez wystawienie dokumentu, który nie spełnia przyjętych założeń tzn. zawiera rabat zarówno na pozycji jak i na całym dokumencie. Ostrzeżenie przy zatwierdzaniu dokumentu „Nie mogą być ustawione rabaty na pozycjach i na całym dokumencie!" to informacja, że zadanie zrealizowaliśmy poprawnie.

Istotą całego rozwiązania jest fragment kodu:

```
  if exists (select 1 from pozycja_dokumentu_magazynowego where id_dok_handlowego =
@IdObiektuNadrzednego and rabat2<>0 and rabat<>0)
    begin
     select @errmsg = 'Nie mogą być ustawione rabaty na pozycjach i na całym dokumencie!'
     goto Error
    end
```

w którym następuje sprawdzenie czy istnieją pozycje z wypełnionymi rabatami dla pozycji (kolumna rabat) i nagłówka (kolumna rabat2) w tabeli pozycja_dokumentu_magazynowego. Jeśli ww. warunek jest spełniony ustawiany jest stosowny komunikat i sygnalizowany błąd.

Warto podkreślić, że dzięki specyfice obsługi błędów w gniazdach rozszerzeń raportowanych z procedur SQL (patrz Rozdział 4) nie musimy (chociaż możemy) odbierać błędów i oprogramowywać ich wyświetlania. Gniazdo robi to automatycznie tzn. po odebraniu sygnalizacji błędu wyświetla komunikat o błędzie (zwracany z SQL funkcją raiserror) i przerywa dalsze wykonanie kodu zarówno umieszonego w gnieździe rozszerzeń jak i standardowego kodu programu, którego wykonanie nastąpiłoby normalnie po zakończeniu kodu gniazda.

W niektórych przypadkach powyższe zachowanie w obsłudze błędów może być jednak pewną przeszkodą np. w sytuacji ustawiania podwójnych rabatów program mógłby wyświetlać stosowny

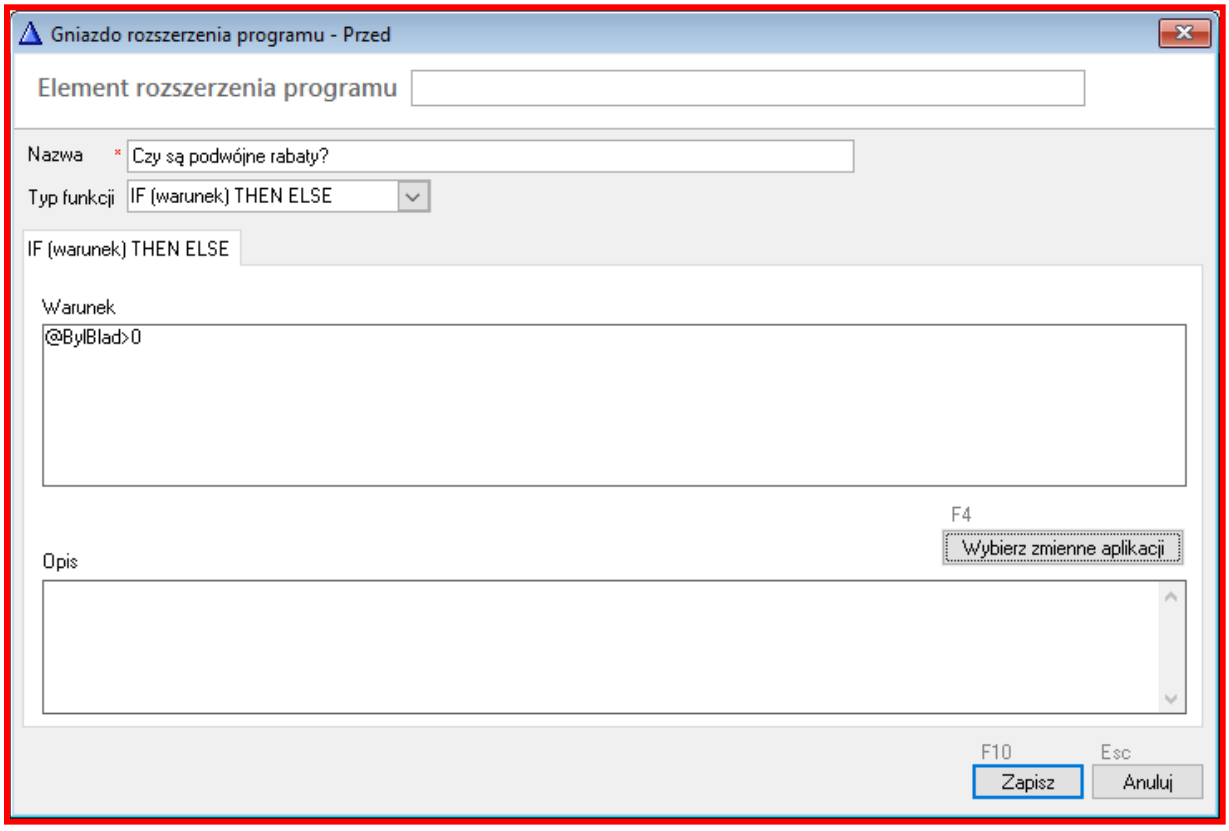
komunikat jednocześnie pozwalając zdecydować czy zatwierdzić dokument czy też nie. Jak zrealizować takie zachowanie programu pokazuje Przykład 2.

6.2 Przykład 2

Tym razem kontrolę czy w dokumencie handlowym są ustawione rabaty dla pozycji i całego dokumentu chcemy nieco rozbudować w stosunku do przykładu z pkt. 6.1. Jeśli rabaty są ustawione dla jakiejkolwiek pozycji program powinien wyświetlić ostrzeżenie o tym fakcie i zadać pytanie czy zatwierdzić dokument.

Chcemy przechwycić komunikat generowany z procedury SQL i wyświetlić pytanie czy zatwierdzić dokument. Aby ominąć standardową obsługę błędu najprościej wykorzystać zmienne @BylBlad oraz @TekstBledu. Zmienna @BylBlad ustawiana jest na wartość 1 gdy wystąpi błąd zaś komunikat błędu zapisywany jest w zmiennej @TekstBledu. Jeśli następna funkcja w kolejności do wykonania po funkcji, która wygenerowała błąd (w naszym przypadku jest to procedura SQL) użyje jako parametrów jednej z ww. zmiennych (np. w funkcji IF @BylBlad>0 THEN ELSE lub Komunikat wyświetlający @TekstBledu) wówczas program przyjmuje, że obsługę błędu przejmuje programista gniazda i wykonanie dalszego kodu gniazda nie jest przerywane a także nie jest wyświetlany standardowy komunikat błędu.

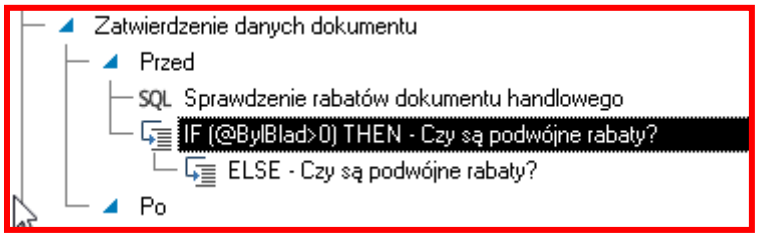
W pierwszej kolejności musimy sprawdzić w warunku funkcji IF THEN ELSE czy zmienna błąd @BylBlad > 0 (lub równa 1) co oznacza, że pojawił się błąd. W tym celu ustawiamy kursor w edytorze gniazd rozszerzeń na funkcji „Sprawdzenie rabatów dokumentu handlowego" i naciskamy przycisk "Dodaj" przechodząc do formularza edycji funkcji. Wybieramy typ funkcji IF (warunek) THEN ELSE (Rys 22).



Rys 22. Definiowanie funkcji IF (warunek) THEN ELSE

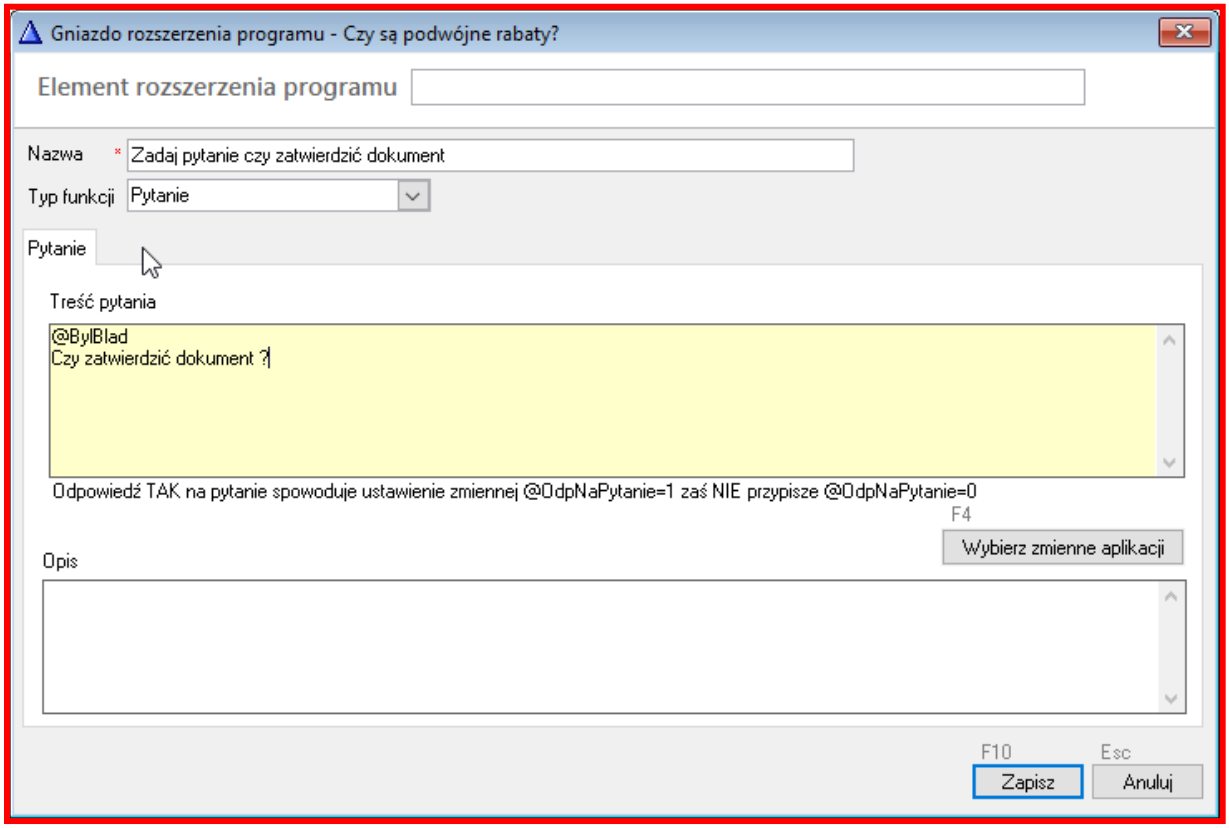
Z listy dostępnych zmiennych (klawisz F4) wybieramy zmienną @BylBlad i zatwierdzamy wybór. W polu „Warunek" program automatycznie zaproponuje „@BylBlad>0". Pozostawiamy taki warunek nie zmieniony (albo możemy wpisać @BylBlad=1). W polu Nazwa podajemy nazwę funkcji np. „Czy są podwójne rabaty?" i zatwierdzamy (klawiszem F10) formularz edycji funkcji w gnieździe rozszerzeń.

W efekcie zawartość gniazda o adresie Dokumenty handlowe\Okno danych dokumentu handlowego\Dodawanie lub poprawianie dokumentu\Zatwierdzenie danych dokumentu\Przed *powinna wyglądać jak na (Rys 23).*



Rys 23. Stan gniazda po dodaniu instrukcji IF (warunek) THEN ELSE (Przykład 2)

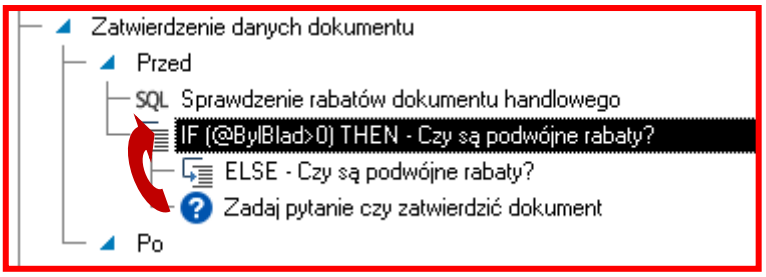
Teraz musimy dodać pytanie, które powinno wyświetlić się jeśli @BylBlad>0. Dodajemy więc kolejną funkcję i z listy typów funkcji wybieramy „Pytanie" podając parametry jak na (Rys 24).



Rys 24. Definiowanie funkcji typu Pytanie (Przykład 2)

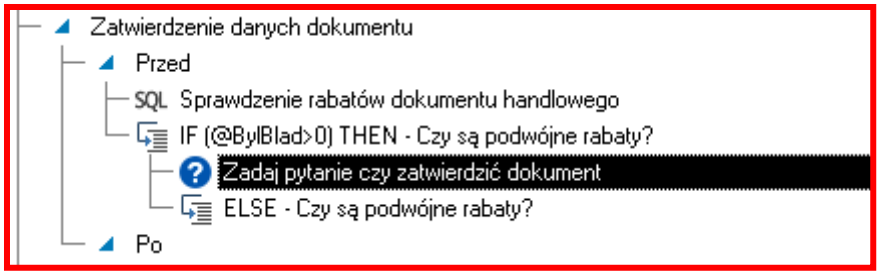
Zatwierdzamy formularz funkcji. Umieszczenie w treści pytania zmiennej @TekstBledu spowoduje, że wyświetli się w oknie dialogowym odebrany tekst błędu oraz ciąg znaków „Czy zatwierdzić dokument?".

Z uwagi na fakt, że domyślnie funkcje dodawane są kolejno musimy przesunąć myszą funkcję „Zadaj pytanie czy zatwierdzić dokument?" aby znalazła się ona w części, która wykonuje się po instrukcji warunkowej IF THEN tzn. jeśli warunek jest spełniony (Rys 25).



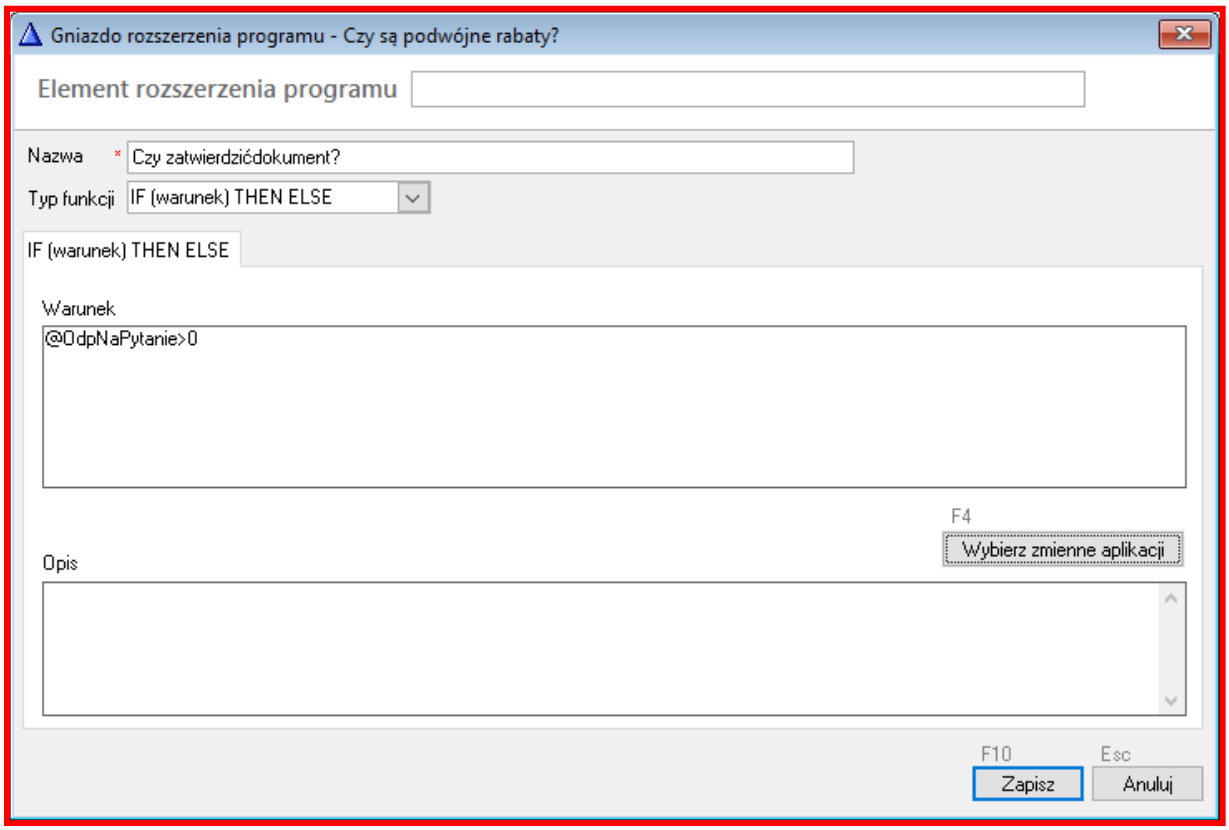
Rys 25. Przeniesienie Pytania przed ELSE (Przykład 2)

W efekcie uzyskamy stan w edytorze gniazd rozszerzeń jak na (Rys 26):



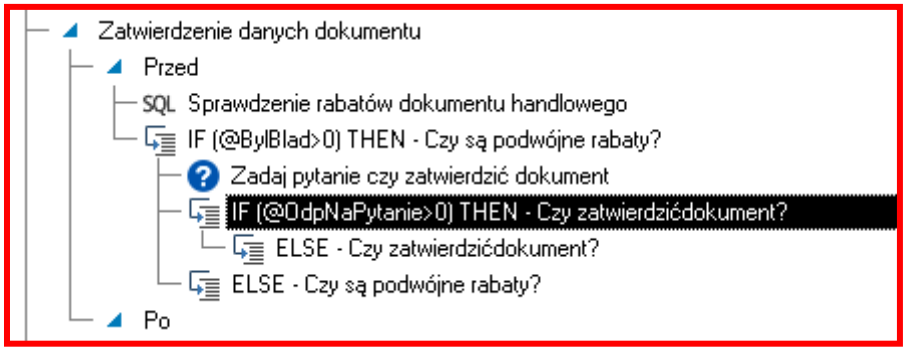
Rys 26. Stan gniazda po przeniesieniu Pytania (Przykład 2)

Następnie trzeba sprawdzić jak użytkownik odpowiedział na pytanie. W tym celu musimy dodać kolejną instrukcją warunkowego wykonania kodu IF THEN ELSE tym razem sprawdzając warunek @OdpNaPytanie > 0 czyli czy użytkownik chce zatwierdzić dokument (wcisnął TAK na oknie dialogowym pytania) (Rys 27).



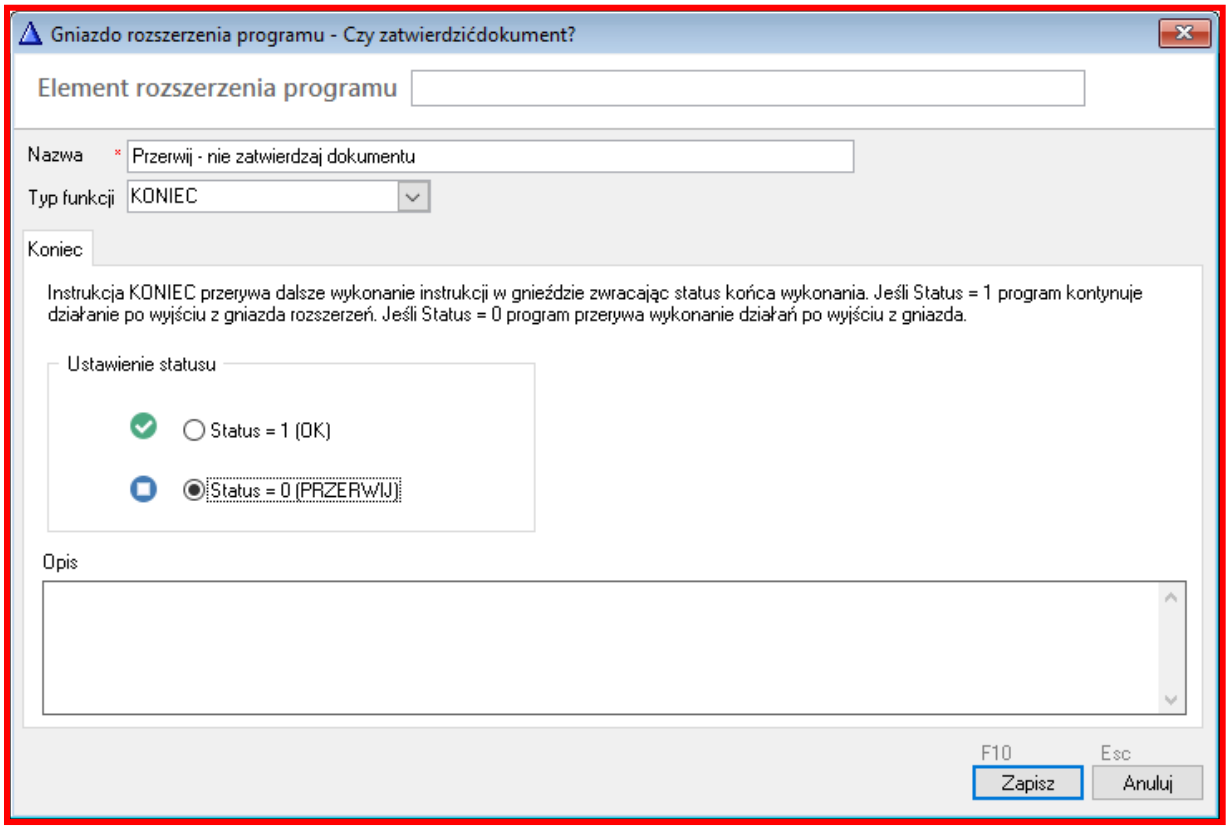
Rys 27. Dodawanie funkcji IF (warunek) THEN ELSE (Przykład 2)

Widok w edytorze gniazd rozszerzeń jaki musimy uzyskać pokazuje (Rys 28)



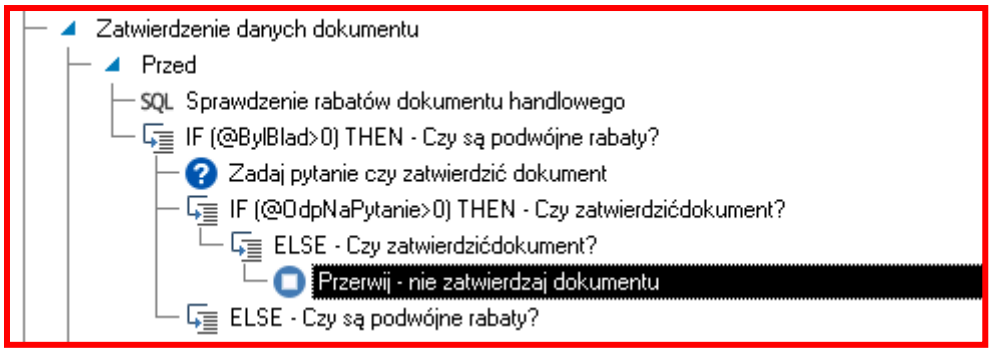
Rys 28. Obsługa odpowiedzi na pytanie (Przykład 2)

Jeśli użytkownik zdecyduje, że nie chce zatwierdzać dokumentu wówczas należy przerwać standardowe zatwierdzanie dokumentu w programie. Możemy to uzyskać korzystając z funkcji Koniec z opcją Przerwij. Funkcję Koniec należy dodać w sekcji „ELSE – Czy zatwierdzić dokument?" ponieważ chcemy aby wykonała się ona gdy użytkownik odpowie, że nie chce zatwierdzać dokumentu (Rys 29).



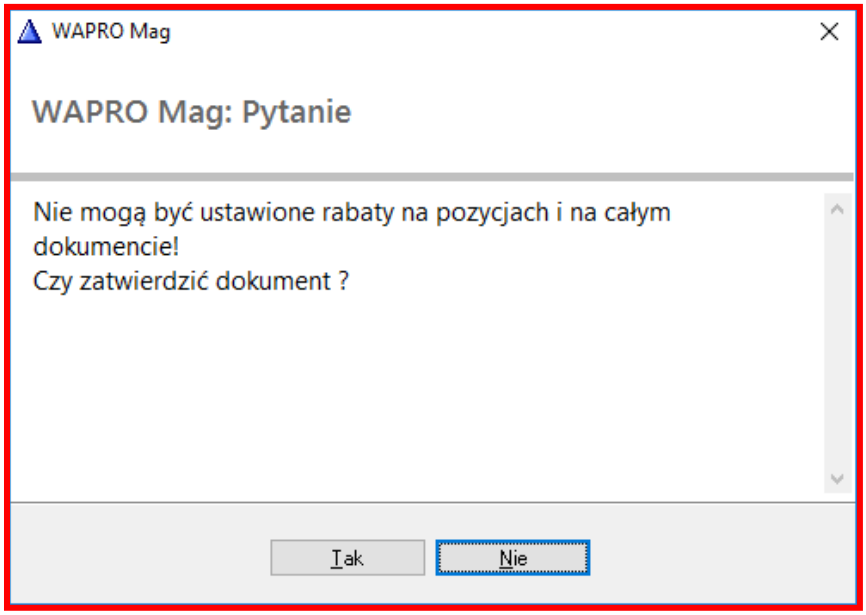
Rys 29. Definiowanie funkcji Koniec z opcją Przerwij (Przykład 2)

Ostatecznie definicja gniazda powinna wyglądać jak na (Rys 30).



Rys 30. Ostateczna postać definicji (Przykład 2)

Po zamknięciu edytora gniazd rozszerzeń możemy na dowolnym dokumencie handlowym przetestować poprawność działania. Ustawiamy rabat do faktury i dla pozycji po czym zatwierdzamy dokument. Powinniśmy otrzymać komunikat:



Rys 31. Uruchomienie wykonania gniazda (Przykład 2)

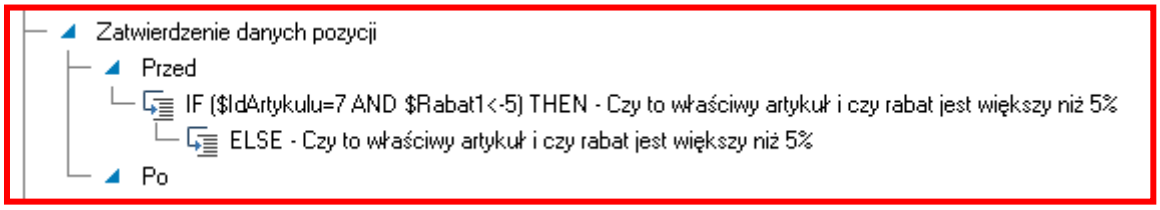
Gdy wybierzemy przycisk Nie program powróci do edycji formularza dokumentu handlowego (faktury). Dokument nie zostanie zatwierdzony dopóki nie usuniemy rabatu na dokumencie lub na wszystkich pozycjach albo nie udzielimy twierdzącej odpowiedzi na powyższe pytanie (Rys 31).

6.3 Przykład 3

W tym przykładzie sprawdzimy czy dla określonego artykułu został przypisany rabat. Dla tego artykułu udzielony rabat nie powinien być większy niż 5% . Pokażemy w jaki sposób możemy ustawić w takim przypadku rabat na formularzu pozycji dokumentu handlowego (magazynowego) aby automatycznie skorygować wartość rabatu w przypadku przekroczenia progu -5%.

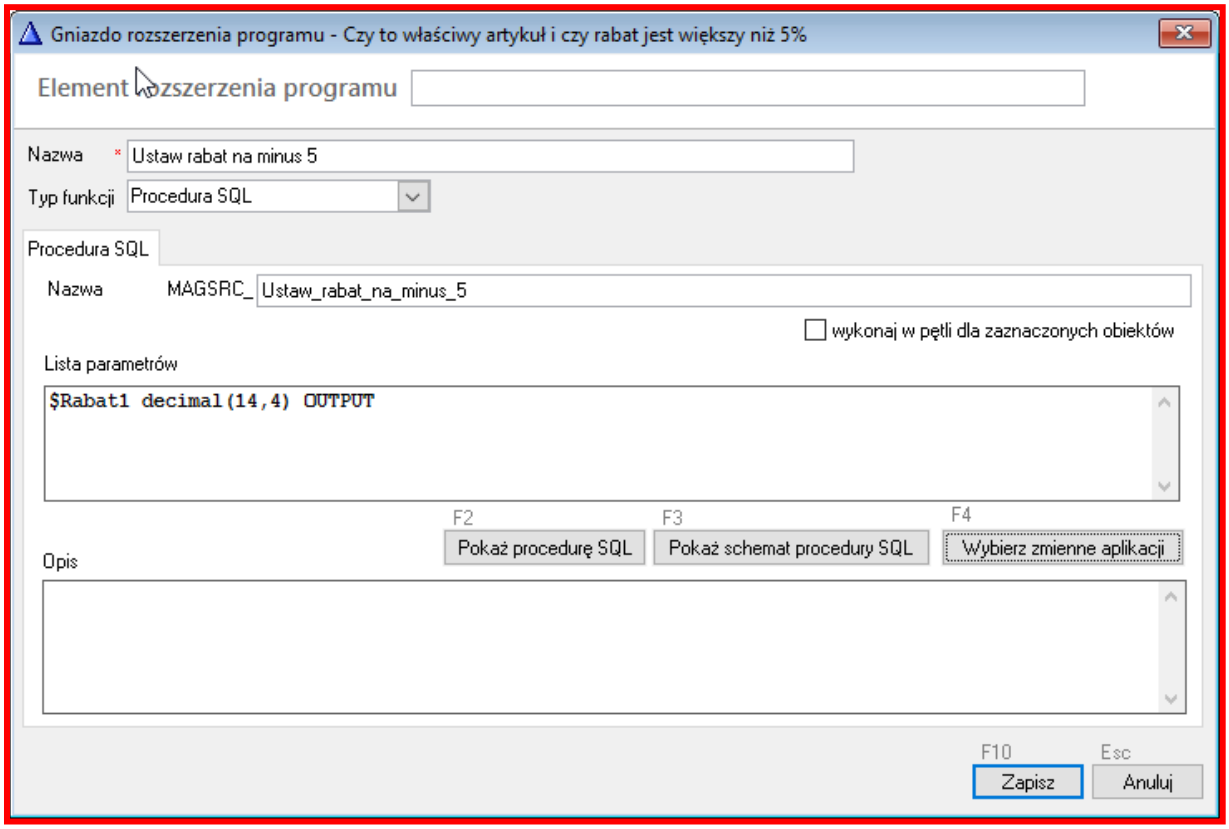
Na początek musimy sprawdzić czy to jest właściwy artykuł (przyjmijmy, że jego ID_ARTYKULU = 7). Sprawdzenia dokonamy przed zatwierdzeniem formularza pozycji dokumentu czyli w gnieździe o adresie Dokumenty handlowe\Okno danych pozycji dokumentu handlowego\Dodawanie lub poprawianie pozycji\Zatwierdzenie danych pozycji\Przed. Umieszczamy w ww. gnieździe instrukcję

warunkowego wykonania IF THEN ELSE, w której określamy warunek $IdArtykulu=7 AND $Rabat1<-5 korzystając oczywiście z listy dostępnych zmiennych. Rabat dla pozycji przechowywany jest w bazie danych w polu Rabat1 tabeli POZYCJA_DOKUMENTU_MAGAZYNOWEGO. Pole to przechowuje rabaty ze znakiem minus a narzuty ze znakiem plus – stąd w przykładzie warunek $Rabat1<-5. Na liście zmiennych znajdziemy jego odpowiednik w postaci zmiennej $Rabat1.



Rys 32. Warunek sprawdzający (Przykład 3)

Teraz pozostaje ustawić rabat na -5 aby nie przekraczać założonej wartości rabatu i jednocześnie nie zmuszać użytkownika do ręcznego korygowania jego wartości na formularzu. Skorzystamy z procedury SQL, w której dokonamy zmiany rabatu.

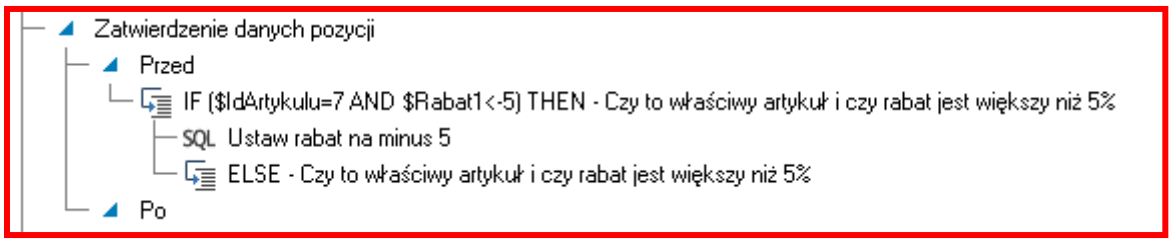


Rys 33. Definicja wywołania procedury SQL ustawiającej rabat na -5 (Przykład 3)

Schemat procedury modyfikujemy do poniższej postaci:

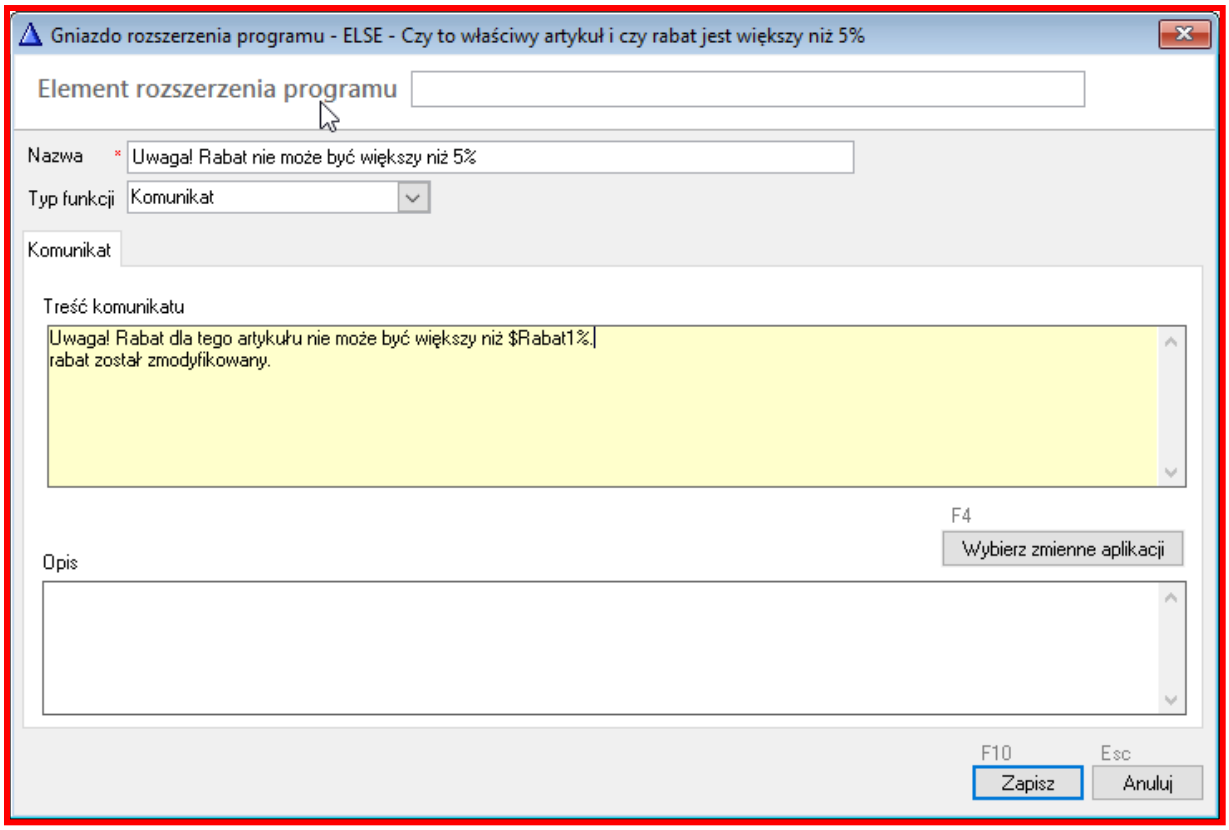
```
if exists (select 1 from sysobjects where name = 'MAGSRC_Ustaw_rabat_na_minus_5' and type = 'P')
  drop procedure MAGSRC_Ustaw_rabat_na_minus_5
go
create procedure MAGSRC_Ustaw_rabat_na_minus_5
@Rabat1 decimal(14,4) OUTPUT
as
declare @errmsg varchar(255)
 begin
  set @Rabat1 = -5
  return
 end
go
```

a następnie wykonujemy skrypt SQL aby umieścić procedurę w bazie danych. Warto zauważyć, że zmienna $Rabat1 jest typu OUTPUT co oznacza, że jej zmiany dokonane w procedurze SQL zostaną automatycznie przekazane do aplikacji. Dzięki temu nie musimy wykonywać żadnych więcej czynności aby ustawienie rabatu wpłynęło na dane na formularzu pozycji dokumentu. Wywołanie procedury umieszczamy w sekcji THEN instrukcji warunkowej jak pokazano na (Rys 34).



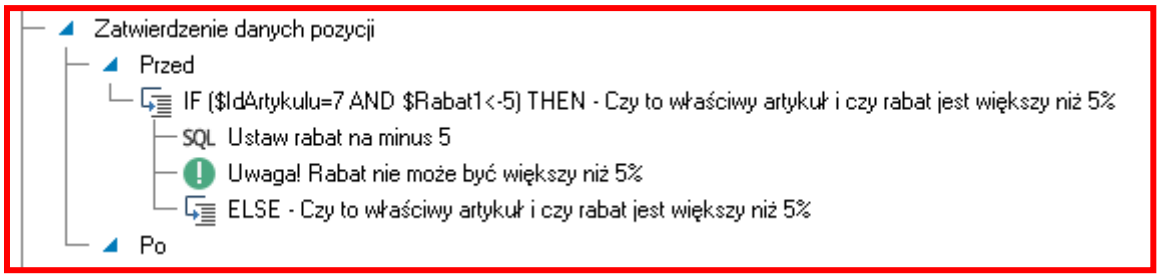
Rys 34. Zawartość gniazda po dodaniu procedury SQL (Przykład 3)

Oczywiście wartości pozycji zostaną automatycznie przeliczone na formularzu tzn. zmiana rabatu w gnieździe spowoduje taki sam efekt jakby użytkownik zmienił jego wartość na formularzu. Przed zatwierdzeniem warto jeszcze poinformować użytkownika stosownym komunikatem, że rabat został ustawiony na -5%.



Rys 35. Definicja komunikatu (Przykład 3)

Warto zauważyć, że w treści komunikatu wyświetlamy zawartość zmiennej $Rabat1 a ponieważ w procedurze SQL ustaliliśmy jej wartość na -5 taka też liczba pojawi się ww. komunikacie. Ostatecznie definicja gniazda powinna wyglądać następująco:



Rys 36. Ostateczna postać definicji gniazda (Przykład 3)

W ten sposób zabezpieczyliśmy formularz pozycji dokumentu handlowego (magazynowego) przed możliwością zmiany dla wybranego artykułu rabatu poniżej pewnego progu niezależnie od ustawień całej polityki rabatowej w WAPRO Mag.

6.4 Przykład 4

W tym przykładzie zostanie zaprezentowany sposób obliczania wartości usługi tzn. kalkulacja ceny netto usługi na podstawie zadanych parametrów. Przyjmiemy, że cena usługi wynika z wyliczenia sumy iloczynów ilości roboczogodzin w poszczególnych stawkach i wartości stawek wg wzoru:

Cena sprzedaży netto usługi = Ilość w stawce1 (rbh) * Stawka1(cena za rbh)

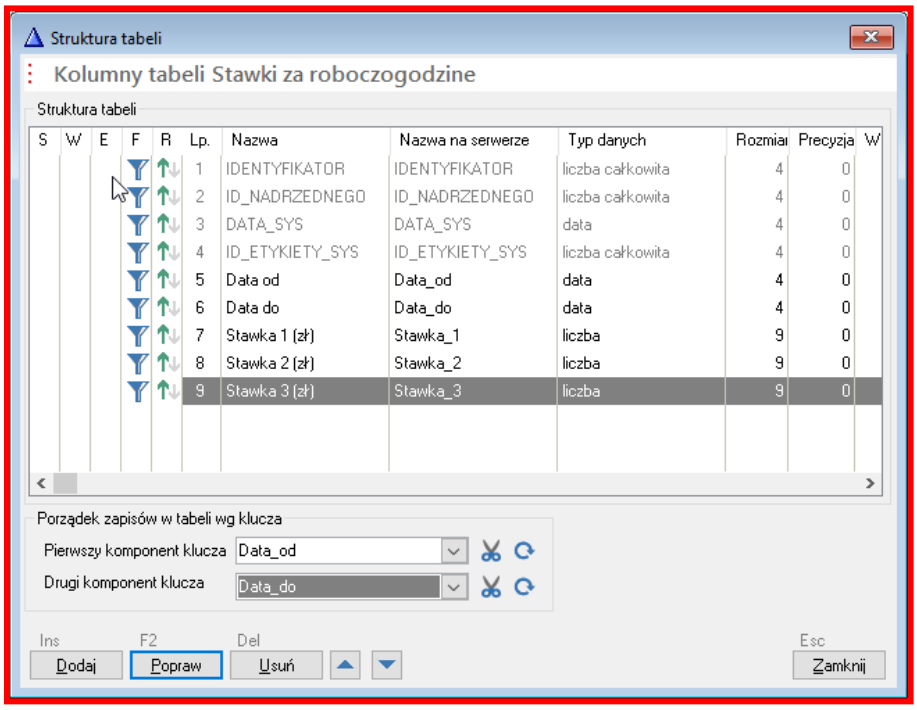
+Ilość w stawce2 (rbh) * Stawka2(cena za rbh) +Ilość w stawce3 (rbh) * Stawka3(cena za rbh)

Ceny stawek za roboczogodzinę (rbh) będą wartościami konfiguracyjnymi przechowanymi w tabeli dodatkowej skojarzonej z firmą. Ilości roboczogodzin będą podawane każdorazowo przed dodaniem lub modyfikacją usługi wybieranej z kartoteki asortymentowej podczas rejestracji faktury sprzedaży. Przyjmiemy założenie, że w jednym dokumencie sprzedaży może być tylko jedna pozycja usługowa. Dodatkowo ilości roboczogodzin ujęte w kalkulacji ceny będą zapisane w opisie pozycji i drukowane na fakturze.

Na początek trzeba zdefiniować odpowiednie tabele dodatkowe. Ceny stawek za roboczogodzinę będą przechowywane w tabeli „Stawki za roboczogodzinę" (nazwa techniczna MAGGEN_Stawki_za_roboczogodzine) (Rys 37) dodanej w grupie tabel Firma. W okresie od 01-01-2009 do 31-12-2009 przyjmiemy, że będą obowiązywały następujące wartości stawek:

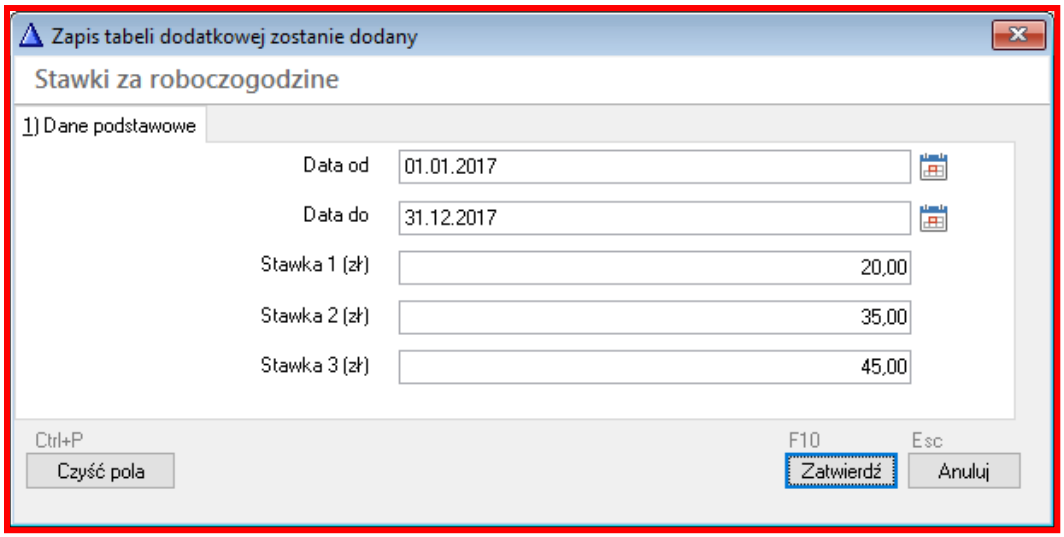
- Stawka 1 – 20 zł za rbh;
- Stawka 2 – 35 zł za rbh;
- Stawka 3 – 45 zł za rbh.

W tym celu należy dodać zapis w tabeli dodatkowej „Stawki za roboczogodzinę" jak pokazano na (Rys 38).



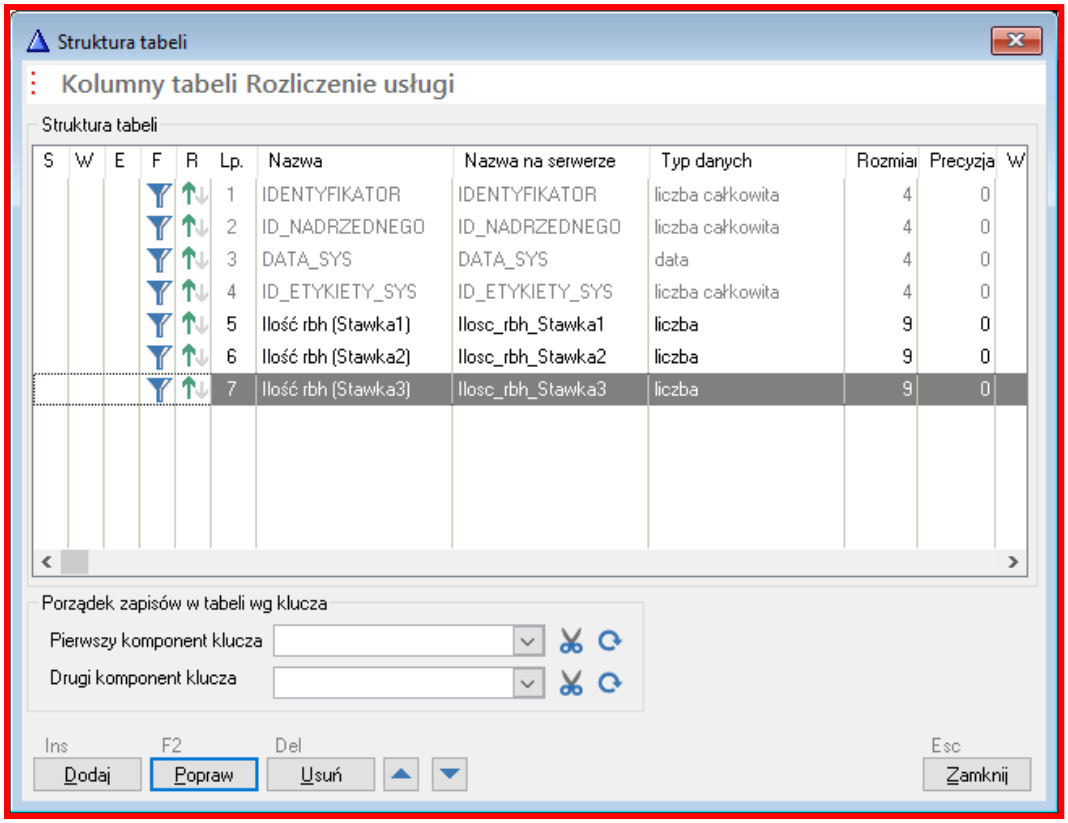
Rys 37. Struktura tabeli „Stawki za roboczogodzinę" (Przykład 4)

Ilości roboczogodzin potrzebnych do kalkulacji ceny będą przechowywane w tabeli „Rozliczenie usługi" (nazwa techniczna MAGGEN_Rozliczenie_uslugi) zdefiniowanej w grupie tabel Dokument_handlowy. Ta grupa tabel jest właściwa dla potrzeb przechowania danych potrzebnych do kalkulacji danych usługi albowiem rozliczenie usługi następować ma podczas fakturowania (wystawiania dokumentu handlowego).



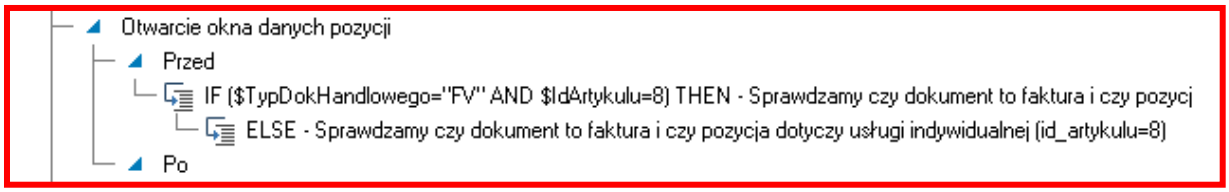
Rys 38. Definicja stawek za roboczogodzinę (Przykład 4)

Tabela „Rozliczenie usługi" przechowuje ilości roboczogodzin w stawkach (Rys 39).



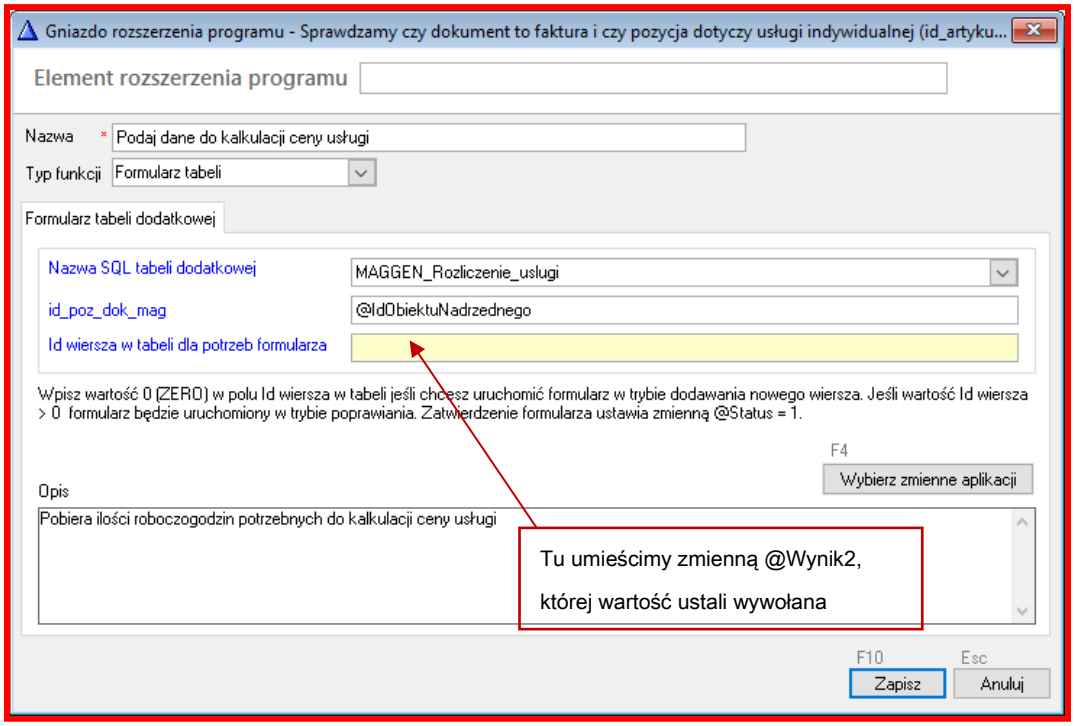
Rys 39. Struktura tabeli „Rozliczenie usługi" (Przykład 4)

W gnieździe rozszerzeń o adresie Dokumenty handlowe\Okno danych pozycji dokumentu handlowego\Dodawanie lub poprawianie pozycji\Otwarcie okna danych pozycji\Przed zdefiniujemy warunek, którego spełnienie będzie aktywowało rozliczenie usługi. Po pierwsze rozliczenie dotyczy konkretnej usługi (odpowiednie id_artykulu z kartoteki asortymentowej programu) a po drugie powinno wywołać się tylko przy fakturowaniu (odpowiedni typ dokumentu handlowego). W instrukcji warunkowej IF THEN ELSE określimy warunek jako $TypDokHandlowego ="FV" AND $IdArtykulu=8 (artykuł o przykładowym id_artykulu równym 8 to usługa podlegająca rozliczeniu). Zapis w gnieździe będzie wyglądał następująco:



Rys 40. Sprawdzenie warunku aktywującego rozliczenie usługi (Przykład 4)

Teraz czas zdefiniować funkcję pobierającą dane do kalkulacji ceny usługi czyli wywołać formularz tabeli dodatkowej „Rozliczenie usługi" (Rys 41). Z listy tabel wybieramy MAGGEN_Rozliczenie_uslugi. Program automatycznie oczekuje podania identyfikatora dokumentu handlowego stosownie do grupy tabel gdzie zdefiniowaliśmy „Rozliczenie usługi". Wybieramy z listy dostępnych zmiennych @IdObiektuNadrzednego. Pojawia się jednak problem jaką wartość umieścić w polu „Id wiersza w tabeli dla potrzeb formularza". W przypadku gdy dodajemy usługę wystarczy podać wartość 0 (zero). Jednakże w trybie poprawiania pozycji powinniśmy podać taką wartość identyfikatora wiersza jaka została nadana podczas dodawania pozycji. Identyfikator wiersza zawsze nadawany jest automatycznie po dodaniu wiersza do tabeli dodatkowej (jest to cecha konstrukcyjna tabel dodatkowych) wobec czego powinniśmy odczytać wartość identyfikatora wiersza i zapisać go w zmiennej (np. @Wynik2), którą podamy w polu „Id wiersza w tabeli dla potrzeb formularza". Odczytu wartości identyfikatora i jego zapisu w zmiennej @Wynik2 najlepiej dokonać za pomocą procedury SQL.

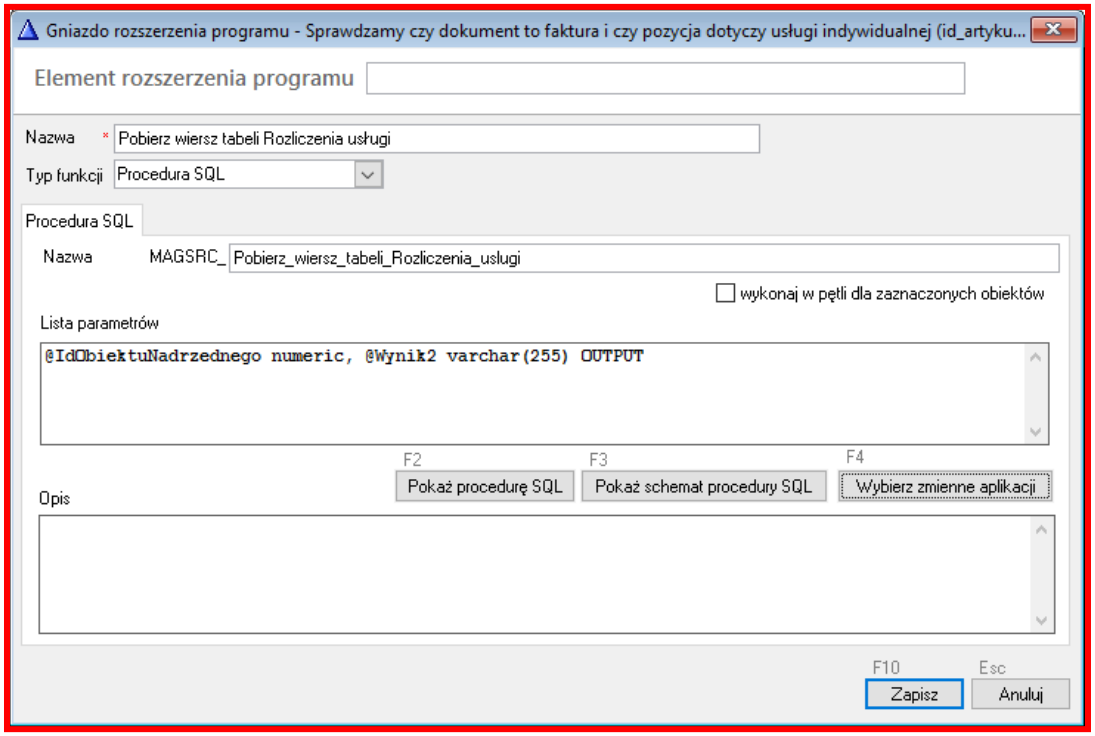


Rys 41. Definicja wywołania formularza tabeli „Rozliczenie usługi" (Przykład 4)

Procedura SQL pobierze wartość kolumny IDENTYFIKATOR tabeli MAGGEN_Rozliczenie_uslugi dla wiersza skojarzonego z daną fakturą (dokumentem handlowym) i przepisze jego wartość do zmiennej @Wynik2. Jeśli taki wiersz nie istnieje (tzn. dodajemy dopiero zapis w tabeli dodatkowej) wówczas procedura wykona proste przypisanie @Wynik2 = 0.

Na początek musimy zdefiniować parametry wywołania procedury (Rys 42) – będzie potrzebna zmienna @Wynik2 (bo w niej zapiszemy wartość szukanego identyfikatora) a także @IdObiektuNadrzednego (bo

zapisy tabeli dodatkowej „Rozliczenie usługi” skojarzone są z tabelą DOKUMENT_HANDLOWY poprzez relację ID_NADRZEDNEGO = ID_DOKUMENTU_HANDLOWEGO (patrz – Struktura tabeli ... - Rys 39])

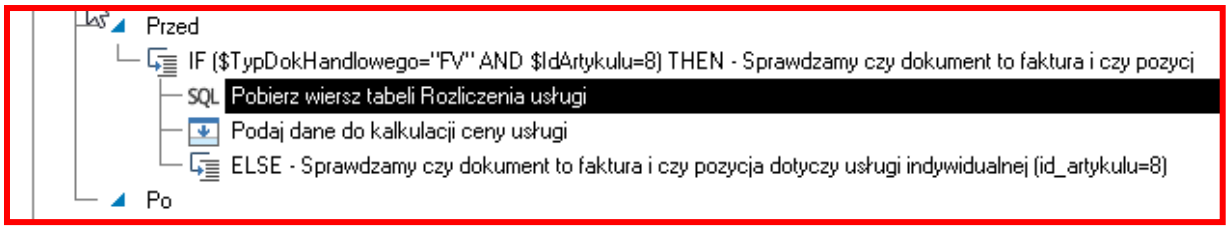


Rys 42. Definicja wywołania procedury SQL pobierającej identyfikator tabeli (Przykład 4)

Zmodyfikujemy zaproponowany dla procedury schemat zapisując następująco algorytm pobierania identyfikatora wiersza tabeli:

```
if exists (select 1 from sysobjects where name = 'MAGSRC_Pobierz_wiersz_tabeli_Rozliczenia_uslugi' and type
= 'P')
  drop procedure MAGSRC_Pobierz_wiersz_tabeli_Rozliczenia_uslugi
go
create procedure MAGSRC_Pobierz_wiersz_tabeli_Rozliczenia_uslugi
@IdObiektuNadrzednego numeric, @Wynik2 varchar(255) OUTPUT
as

 begin

  select top 1 @Wynik2 = IDENTYFIKATOR
  from MAGGEN_Rozliczenie_uslugi
  where ID_NADRZEDNEGO = @IdObiektuNadrzednego

  if @@rowcount = 0
    set @Wynik2 = 0

  return
 end
go
```

Ostatecznie funkcje umieszczone w gnieździe powinny wyglądać jak na (Rys 43).

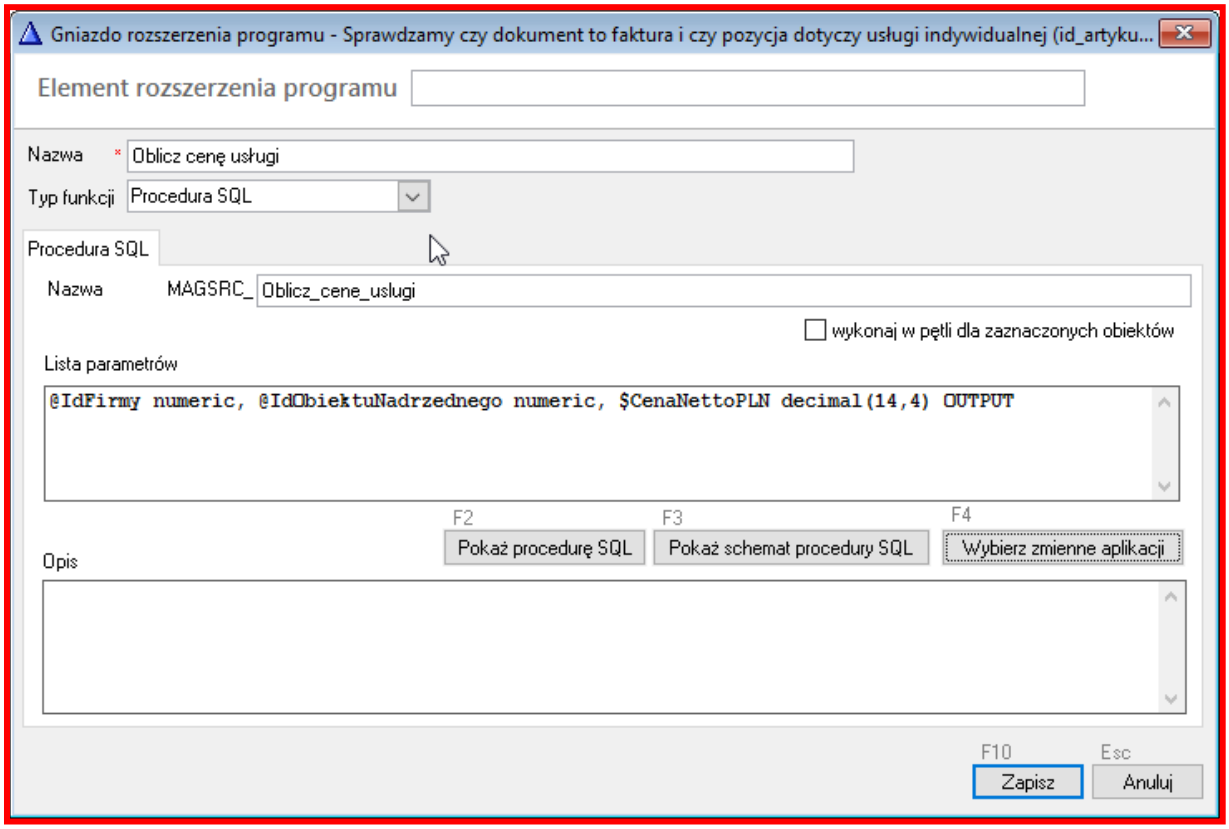


Rys 43. Poprawna definicja wywołania formularza tabeli „Rozliczenie usługi" (Przykład 4)

Warto zauważyć, że opisana sekwencja „procedura SQL pobierająca wiersz tabeli – formularz tabeli" gwarantuje nam, że dla danego dokumentu (obiektu nadrzędnego) w tabeli dodatkowej będzie istniał tylko jeden zapis. Przepisanie do zmiennej @Wynik2 identyfikatora wiersza tabeli, który następnie podawany jest do funkcji wywołującej formularz powoduje uruchomienie formularza w trybie poprawiania. Możemy więc zmienić zawartość pól w wierszu tabeli ale nie możemy dodać drugiego zapisu w „Rozliczeniu usługi" dla danego dokumentu handlowego za pomocą ww. funkcji. W efekcie, opisana technika realizuje implementację relacji „jeden do jeden" między wierszem tabeli DOKUMENT_HANDLOWY i wierszem tabeli MAGGEN_Rozliczenie_uslugi.

Mając dane potrzebne do ustalenia ceny usługi pozostaje dokonać stosownych przeliczeń, które wykonamy po zatwierdzeniu formularza tabeli „Rozliczenie usługi". Ponownie posłużymy się procedurą SQL.

W procedurze będziemy potrzebowali @IdObiektuHandlowego (id_dokumentu_handlowego), @IdFirmy (aby pobrać wartości stawek z tabeli dodatkowej „Stawki za roboczogodzinę" przypisanej do tabeli Firma) oraz zmienną @CenaNettoPLN, którą musimy wyliczyć (Rys 44).



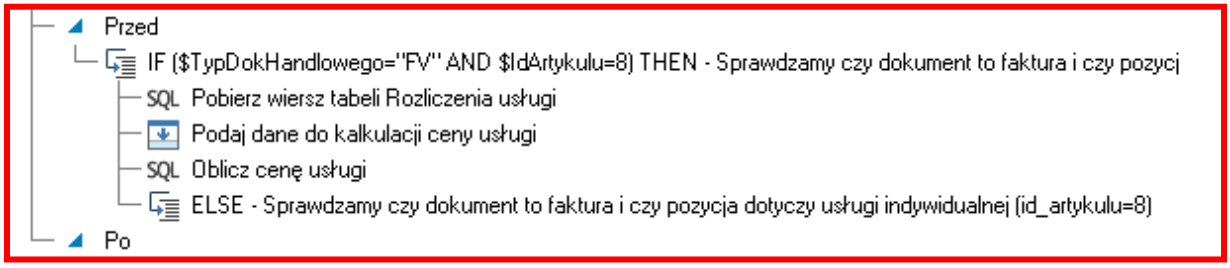
Rys 44. Definicja procedury obliczenia ceny usługi (Przykład 4)

Implementację algorytmu kalkulacji ceny usługi można sprowadzić do jednego zapytania SQL:

```
if exists (select 1 from sysobjects where name = 'MAGSRC_Oblicz_cene_uslugi' and type = 'P')
  drop procedure MAGSRC_Oblicz_cene_uslugi
go
create procedure MAGSRC_Oblicz_cene_uslugi
@IdFirmy numeric, @IdObiektuNadrzednego numeric, @CenaNettoPLN decimal(14,4) OUTPUT
as
declare @errmsg varchar(255)
 begin

   select top 1 @CenaNettoPLN = r.Ilosc_rbh_Stawka_1 * s.Stawka_1
                                              + r.Ilosc_rbh_Stawka_2 * s.Stawka_2
                                              + r.Ilosc_rbh_Stawka_3 * s.Stawka_3
   from  MAGGEN_Rozliczenie_uslugi r
      ,MAGGEN_Stawki_za_roboczogodzine s
   where r.ID_NADRZEDNEGO = @IdObiektuNadrzednego
    and r.Data_sys between s.Data_od and s.Data_do
    and s.ID_NADRZEDNEGO = @IdFirmy
   order by s.Data_do,s.Data_od

   return
 end
go
```

Warto zwrócić uwagę, że w tabeli stawek za roboczogodzinę określiliśmy daty obowiązywania stawek. W zapytaniu SQL wyliczającym cenę wykorzystujemy ten fakt wybierając te stawki, które spełniają kryterium daty (warunek r.Data_sys between s.Data_od and s.Data_do). W przypadku gdyby okresy dat się pokrywały (lub zachodziły na siebie w przedziałach „od-do") czyli zapytanie zwróciłoby kilka wierszy zawierających stawki, wówczas wybierzemy tylko jedną (select top 1) sortując wiersze wg s.Data_do,s.Data_od. Z kolei kolumna r.Data_sys wypełniana jest automatycznie dla dodanego wiersza tabeli dodatkowej i zawiera datę dodania wiersza. W naszym przykładzie oznacza to datę wprowadzenia ilości roboczogodzin do tabeli „Rozliczenie usługi". Tym sposobem możemy bardzo łatwo powiązać stawki za roboczogodziny z ilością roboczogodzin potrzebnych do kalkulacji ceny usługi.

Ostatecznie w gnieździe powinniśmy uzyskać widok definicji funkcji jak na Rys 45.



Rys 45. Pobranie danych do kalkulacji ceny i jej obliczenie (Przykład 4)

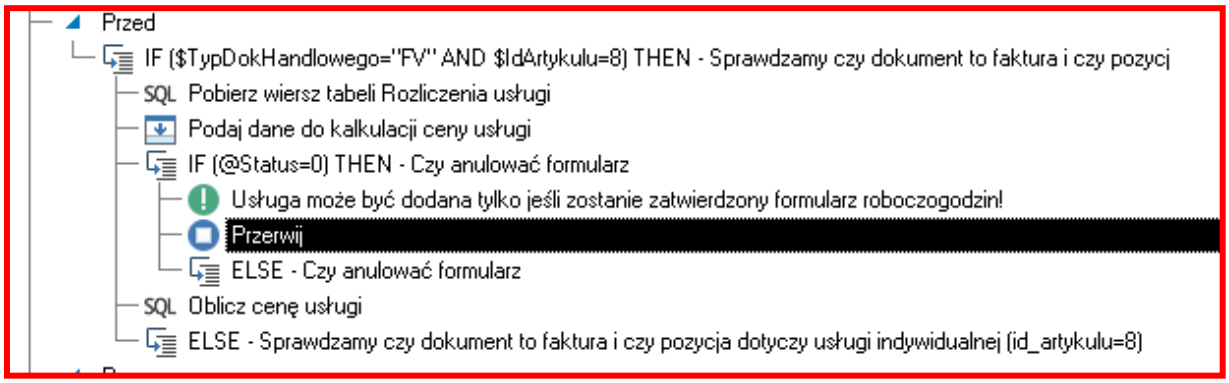
Praktycznie na tym można by zakończyć implementację rozwiązania w naszym przykładzie ponieważ cenę usługi mamy obliczoną i przekazywaną do formularza pozycji dokumentu handlowego. Pozostaje jednak oprogramować sytuacje, w których użytkownik może podjąć działania inne niż spodziewane np. :

a) użytkownik anuluje formularz tabeli „Rozliczenie usługi" zamiast zatwierdzić;

b) użytkownik podał zerowe ilości roboczogodzin – cena usługi wyniesie więc zero;

c) użytkownik pomimo obliczenia ceny usługi zmodyfikuje ją ręcznie przed zatwierdzeniem formularza pozycji dokumentu handlowego a do tego nie chcemy dopuścić;

Jeśli użytkownik anuluje formularz powinniśmy przynajmniej wywołać ostrzeżenie o konieczności określenia ilości i akceptacji roboczogodzin. Aby stwierdzić czy użytkownik anulował formularz wystarczy sprawdzić w warunku IF THEN ELSE czy zmienna @Status = 0. Potem w sekcji THEN warunku umieścimy komunikat o treści „Usługa może być dodana tylko jeśli zostanie zatwierdzony formularz roboczogodzin!" a następnie funkcję KONIEC z opcją Przerwij aby zatrzymać proces wystawiania pozycji dokumentu handlowego (podobnie jak w Przykład 2). W efekcie uzyskamy widok definicji funkcji w gnieździe jak na (Rys 46).



Rys 46. Zawartość gniazda po dodaniu obsługi anulowania formularza (Przykład 4)

Następnie należy zabezpieczyć się przed sytuacją podania zerowej ilości roboczogodzin przez użytkownika. Najprościej wykorzystać do tego mechanizm obsługi błędów w gniazdach tzn. w przypadku kiedy wyliczona cena będzie równa 0 wygenerujemy błąd w procedurze SQL obliczającej ilość. Brak jawnej obsługi tego błędu w gnieździe spowoduje przechwycenie tego błędu przez standardowe mechanizmy obsługi błędów gniazd, wyświetlenie komunikatu i przerwanie dalszego dodawania pozycji dokumentu handlowego. Wystarczy wobec tego zmodyfikować kod procedury obliczenia ceny usługi dodając warunek na sprawdzenie ceny i sygnalizację błędu:

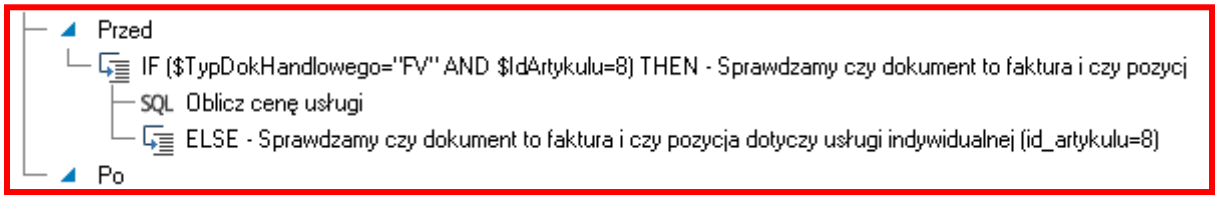
```
  if isnull(@CenaNettoPLN,0.0) = 0.0
    begin
     set @errmsg = 'Cena usługi wynosi 0! Nie można dodać usługi bez podania ceny wynikającej z rozliczenia
roboczogodzin wykonania usługi!'
     raiserror (@errmsg,16,1)
    end
```

Procedura SQL będzie więc wyglądała następująco:

```
if exists (select 1 from sysobjects where name = 'MAGSRC_Oblicz_cene_uslugi' and type = 'P')
  drop procedure MAGSRC_Oblicz_cene_uslugi
go
create procedure MAGSRC_Oblicz_cene_uslugi
@IdFirmy numeric, @IdObiektuNadrzednego numeric, @CenaNettoPLN decimal(14,4) OUTPUT
as
declare @errmsg varchar(255)
 begin

  select top 1 @CenaNettoPLN = r.Ilosc_rbh_Stawka_1 * s.Stawka_1
                                        + r.Ilosc_rbh_Stawka_2 * s.Stawka_2
                                        + r.Ilosc_rbh_Stawka_3 * s.Stawka_3
  from  MAGGEN_Rozliczenie_uslugi r
    ,MAGGEN_Stawki_za_roboczogodzine s
  where r.ID_NADRZEDNEGO = @IdObiektuNadrzednego
```

```
    and r.Data_sys between s.Data_od and s.Data_do
    and s.ID_NADRZEDNEGO = @IdFirmy
  order by s.Data_do,s.Data_od

  if isnull(@CenaNettoPLN,0.0) = 0.0
    begin
     set @errmsg = 'Cena usługi wynosi 0! Nie można dodać usługi bez podania ceny wynikającej z rozliczenia
roboczogodzin wykonania usługi!'
      raiserror (@errmsg,16,1)
    end

  return
 end
go
```

Pozostało zabezpieczyć formularz pozycji dokumentu handlowego przed ręczną modyfikacją ceny przez użytkownika. W praktyce musimy przed zatwierdzeniem formularza pozycji ponownie obliczyć cenę i jeśli będzie inna niż wyliczona zablokować możliwość dodania lub poprawienia pozycji dokumentu. W tym celu w gnieździe *Dokumenty handlowe\Okno danych pozycji dokumentu handlowego\Dodawanie lub poprawianie pozycji\Zatwierdzenie danych pozycji\Przed* musimy umieścić wywołanie funkcji obliczania ceny usługi oczywiście po sprawdzeniu warunku analogicznego jak na (Rys 40). Mamy więc stan zapisów w gnieździe jak na (Rys 47):

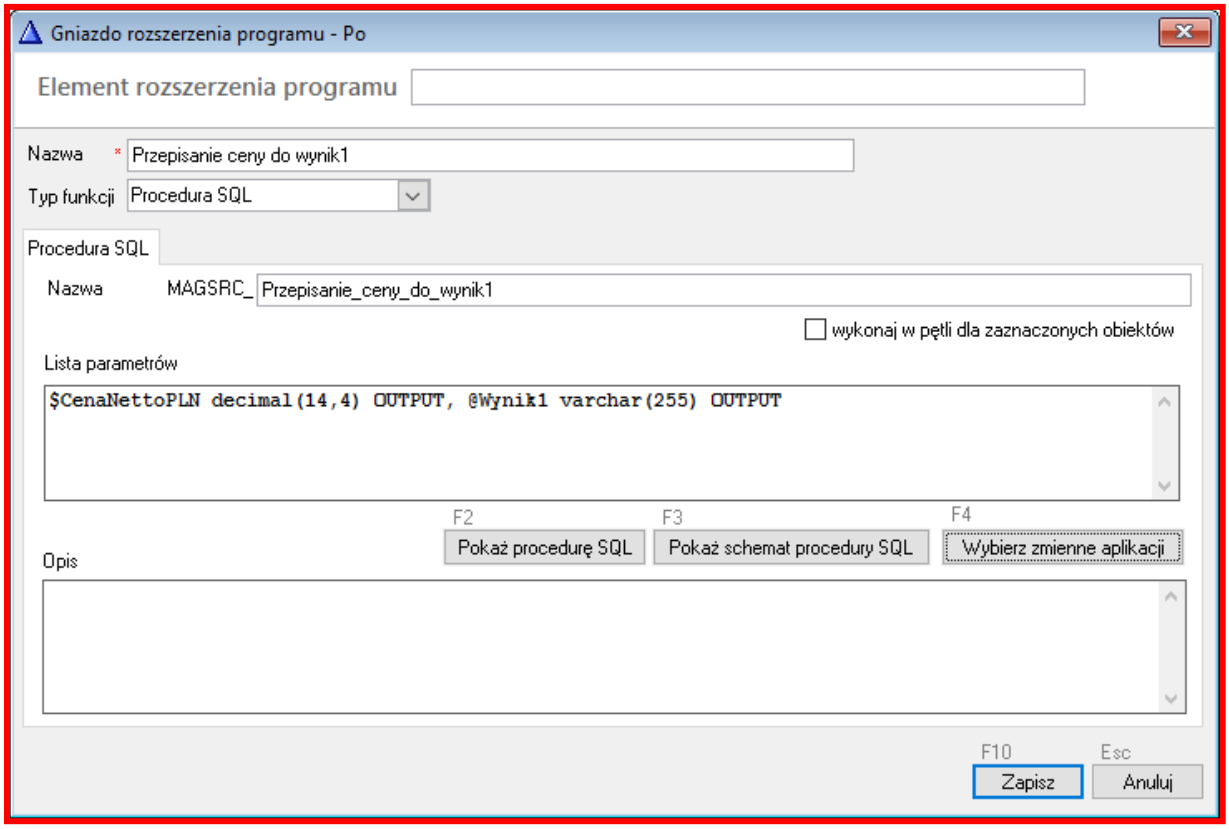


Rys 47. Przy zatwierdzaniu danych pozycji trzeba sprawdzić cenę (Przykład 4)

Wykorzystaliśmy ponownie wcześniej zdefiniowaną procedurę SQL do obliczenia ceny jednakże przydałoby się w przypadku kiedy cena wyliczona różni się od tej na formularzu pozycji dokumentu pojawił się stosowny komunikat. Aby sprawdzić czy ceny się różnią należy oczywiście zastosować warunek IF THEN ELSE. Pojawia się jednak problem – procedura obliczania ceny automatycznie podstawia wyliczoną wartość do zmiennej $CenaNettoPLN skojarzonej z wartością na formularzu wobec czego wartość ceny z formularza wprowadzona przez użytkownika zostaje nadpisana. Należy zatem przed uruchomieniem procedury obliczania ceny przepisać do zmiennej @Wynik1 (lub @Wynik2 lub @Wynik3) wartość $CenaNettoPLN a następnie po wyliczeniu ceny porównać czy @Wynik1 = $CenaNettoPLN. Przepisanie do zmiennej można zrealizować bardzo prostą procedurą SQL:

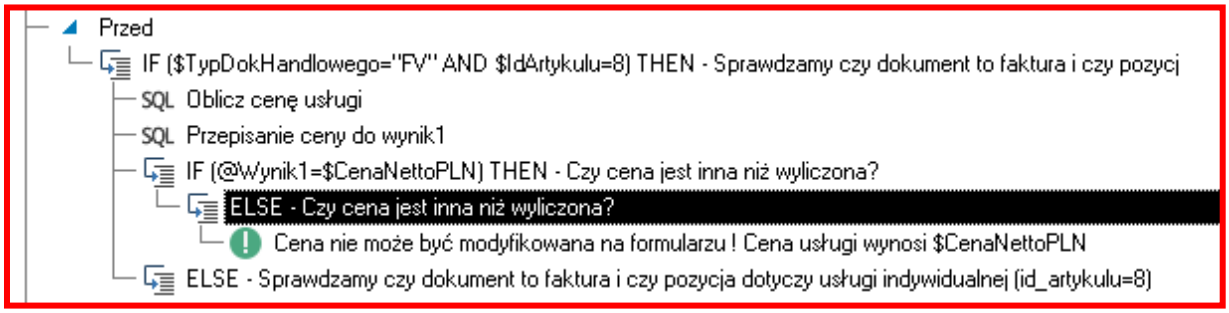
```
if exists (select 1 from sysobjects where name = 'MAGSRC_Przepisanie_ceny_do_wynik1' and type = 'P')
  drop procedure MAGSRC_Przepisanie_ceny_do_wynik1
go
create procedure MAGSRC_Przepisanie_ceny_do_wynik1
@CenaNettoPLN decimal(14,4) OUTPUT, @Wynik1 varchar(255) OUTPUT
as
declare @errmsg varchar(255)
 begin

 set @Wynik1 = @CenaNettoPLN

 end
go
```

której definicję wywołania obrazuje (Rys 48):



Rys 48. Procedura przepisania ceny do zmiennej @Wynik1 (Przykład 4)

Teraz możemy dodać stosowny warunek i komunikat. Jeśli cena @Wynik1 = $CenaNettoPLN to kontynuujemy zatwierdzanie pozycji zaś w przeciwnym przypadku (czyli po ELSE) wyświetlimy komunikat o treści np. „Cena nie może być modyfikowana na formularzu ! Cena usługi wynosi $CenaNettoPLN". Warto zauważyć, że nie musimy po komunikacie przerwać wykonania kodu bo w zmiennej $CenaNettoPLN jest już cena wyliczona wobec czego zostanie ona automatycznie przepisana na formularz i zapisana w pozycji dokumentu handlowego. Ostatecznie zawartość gniazda Dokumenty handlowe\Okno danych pozycji dokumentu handlowego\Dodawanie lub poprawianie pozycji\Zatwierdzenie danych pozycji\Przed dla naszego przykładu będzie wyglądać jak na (Rys 49).

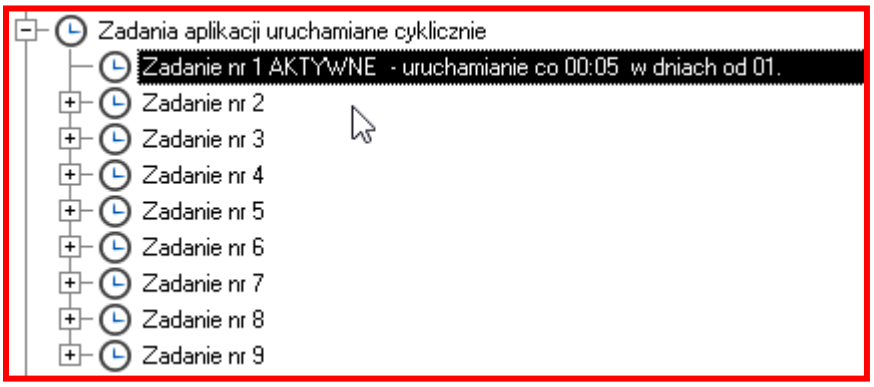


Rys 49. Obsługa zatwierdzania pozycji (Przykład 4)

7. Zadania aplikacji uruchamiane cyklicznie (gniazda cykliczne)

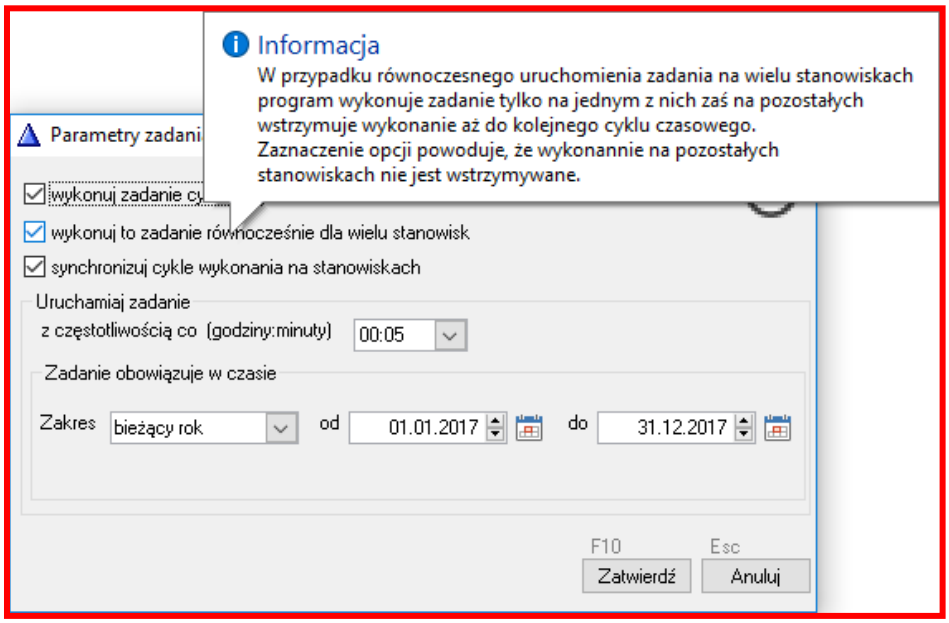
Czasami może pojawić się potrzeba uruchamiania funkcji w pewnych odstępach czasu. Przykładowo można monitorować pojawienie się określonych warunków np. czy pojawiło się zamówienie ze sklepu internetowego i uruchomić powiadomienie dla użytkownika aby zweryfikował dokument znajdujący się w buforze zamówień. W celu okresowego uruchamiania zestawu funkcji w WAPRO Mag wprowadzono zadania aplikacji uruchamiane cykliczne (gniazda cykliczne).

Za pomocą gniazd cyklicznych można np. oprogramować transmisję dokumentów między różnymi źródłami danych. Gniazda cykliczne są niezależne od wersji SQL Servera ponieważ uruchamiane są przez aplikację na każdym, aktywnym w danym momencie komputerze z uruchomionym programem.



Rys 50. Zadania aplikacji uruchamiane cyklicznie

Dostępnych jest 9 niezależnych zadań aplikacji uruchamianych cyklicznie. Każde zadanie to gniazdo rozszerzeń, w którym dostępne są wszystkie, typowe dla gniazd funkcje. Parametry każdego zadania ustala się pod przyciskiem „Parametry zadania" (Rys 51).



Rys 51. Parametry zadania aplikacji uruchamianego cyklicznie

Zadanie może być w stanie aktywnym lub nieaktywnym tzn. możemy w dowolnym momencie włączyć lub wyłączyć wykonanie zadania poprzez zaznaczenie opcji „wykonuj zadanie cyklicznie" i zatwierdzenie formularza (Rys 51).

Cykl czasu co ile zadanie ma być uruchamiane podaje się w godzinach i minutach przy czym najmniejszą jednostką czasu jak może zostać określona dla zadania są 3 minuty. Częstsze uruchamianie mogłoby spowodować zbyt duże obciążenie aplikacji i w efekcie spowolnić pracę programu. Cykl czasu rozpoczyna się z chwilą uruchomienia aplikacji toteż na każdym stanowisku najczęściej moment aktywacji zadania będzie inny nawet dla tego samego zadania. Taki tryb uruchamiania zadań określamy mianem asynchronicznego wykonania zadań aplikacji.

Zadanie aplikacji uruchamiane cyklicznie może być wykonywane niezależnie na każdym, aktywnym stanowisku pracy lub też może zostać uruchomione tylko na jednym zaś na pozostałych wstrzymane do czasu nadejścia kolejnego momentu uruchomienia zadania, który następuje po upływie określonego cyklu czasowego.

Jeśli w parametrach zadania określono, że ma być wykonywane niezależnie (zaznaczona opcja „wykonuj to zadanie równocześnie dla wielu stanowisk") wówczas zadanie zostanie wykonane nawet jeśli inne stanowisko w tym samym czasie również je wykonuje. W przypadku gdy nie zaznaczymy opcji „wykonuj to zadanie równocześnie dla wielu stanowisk" wówczas program analizuje przed każdym uruchomieniem zadania czy inne stanowisko w danym momencie nie wykonuje już tego zadania i, jeśli taka sytuacja ma miejsce, przerywa wykonanie aż do kolejnego momentu uruchomienia, który może wystąpić na dowolnym stanowisku.

Asynchroniczne wykonywanie zadań aplikacji w różnych momentach czasowych często jest niepotrzebne i może powodować większe obciążenie serwera. Wówczas najlepiej zaznaczyć opcję „synchronizuj cykle wykonania na stanowiskach" co spowoduje, że zanim zadanie zostanie uruchomione zegar cykli wykonania zostanie ustawiony na moment ostatniego uruchomienia zadania na dowolnym stanowisku czyli czas pozostały do wykonania zostanie skorygowany. W efekcie pomimo tego, że kolejne stanowiska mogą w dowolnym momencie wystartować odmierzając czas cykli wykonania zadań to po pewnym czasie wszystkie końcówki aplikacji będą wykonywać zadania dokładnie w tym samym momencie (z dokładnością do kilku sekund). Oczywiście przy odznaczonej opcji „wykonuj to zadanie równocześnie dla wielu stanowisk" uzyskamy efekt, że w danym momencie zadanie zostanie uruchomione tylko raz niezależnie od ilości stanowisk i program nie będzie podejmował nawet próby uruchomienia zadania aż do kolejnego cyklu.

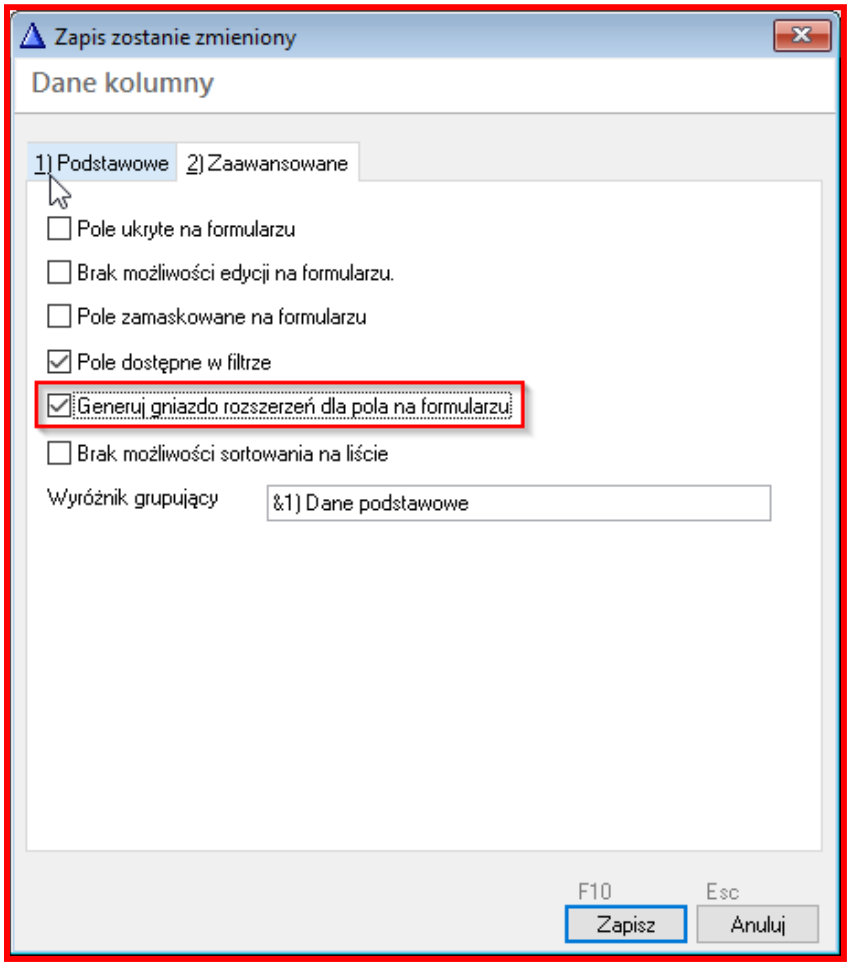
Sama synchronizacja czasu zadań nie jest obciążająca ani dla serwera bazy danych ani dla poszczególnych stanowisk pracy. Zadania wykonywane są w niezależnych wątkach aplikacji WAPRO Mag w systemie Windows ze wszystkimi konsekwencjami tego faktu. Z punktu widzenia użytkownika zadanie wykonywane jest w tle jego bieżącej aktywności. O tym, które stanowisko aktywuje zadanie do wykonania decydują często milisekundy potrzebne dla procesora danej końcówki na przełączenie wątku a także praca użytkownika, która również zajmuje czas potrzebny dla procesu aktywującego zadanie. Nie można zatem jednoznacznie wskazać, które stanowisko będzie wykonywało dane zadanie ale można mieć pewność, że zadanie zostanie wykonane zgodnie ze zdefiniowanymi parametrami uruchamiania.

Przy określaniu parametrów wykonania zadania istnieje również możliwość zdefiniowania zakresu czasu kiedy dane zadanie jest obowiązujące (Rys 51). Po upływie tego czasu zadanie przejdzie w stan nieaktywny (będzie wyłączone). Jest to szczególnie przydatne jeśli zadanie ma charakter tymczasowy.

8. Gniazda rozszerzeń na tabelach dodatkowych

Aplikacja tworzony gniazda rozszerzeń również dla tabel dodatkowych. Oznacza to, że dostępne są gniazda zarówno na liście zapisów tabeli dodatkowych jak również na formularzach tabel dodatkowych. Wygenerowane gniazda rozszerzeń dla tabel dodatkowych mają analogiczną strukturę jak na standardowych ekranach aplikacji ponieważ dotyczą zdarzeń tego samego typu np. otwarcie okna formularza, zatwierdzenie formularz itp. Również edytor gniazd rozszerzeń wywoływany na tabelach dodatkowych tak samo jak w innych miejscach w aplikacji tzn. za pomocą skrótu klawiszowego CTRL+SHIFT+F12.

Jednakże podczas definiowania struktury tabeli dodatkowej na zakładce „Zawansowane" w parametrach kolumny można włączyć generowanie gniazd rozszerzeń dla poszczególnych pól, które znajdą się na formularzu tabeli dodatkowej (Rys 52). Po zaznaczeniu opcji „Generuj gniazdo rozszerzeń dla pola na formularzu" i zatwierdzeniu definicji tabeli, w edytorze gniazd rozszerzeń uruchomionym na formularzu tabeli, pojawi się zdarzenie o nazwie *Pole formularza: <nazwa pola> \ Po* gdzie <nazwa pola> będzie zawierała nazwę opisową kolumny tabeli dodatkowej. Powyższe gniazdo rozszerzeń jest wykonywane po zatwierdzeniu pola na formularzu (np. klawiszem enter) w trakcie edycji zapisu tabeli. Oczywiście każda zmiana w polu formularza ponownie uruchamia ww. zdarzenie.



Rys 52. Generowanie gniazd rozszerzeń dla pól formularzy tabel dodatkowych

Na formularzu tabeli dodatkowej wszystkie pola (oraz pola techniczne tabeli dodatkowej) są reprezentowane w gniazdach rozszerzeń formularza w postaci zmiennych kontekstowych o nazwach odpowiadających nazwom kolumn SQL tabeli w bazie danych. Zmienne skojarzone z polami formularza

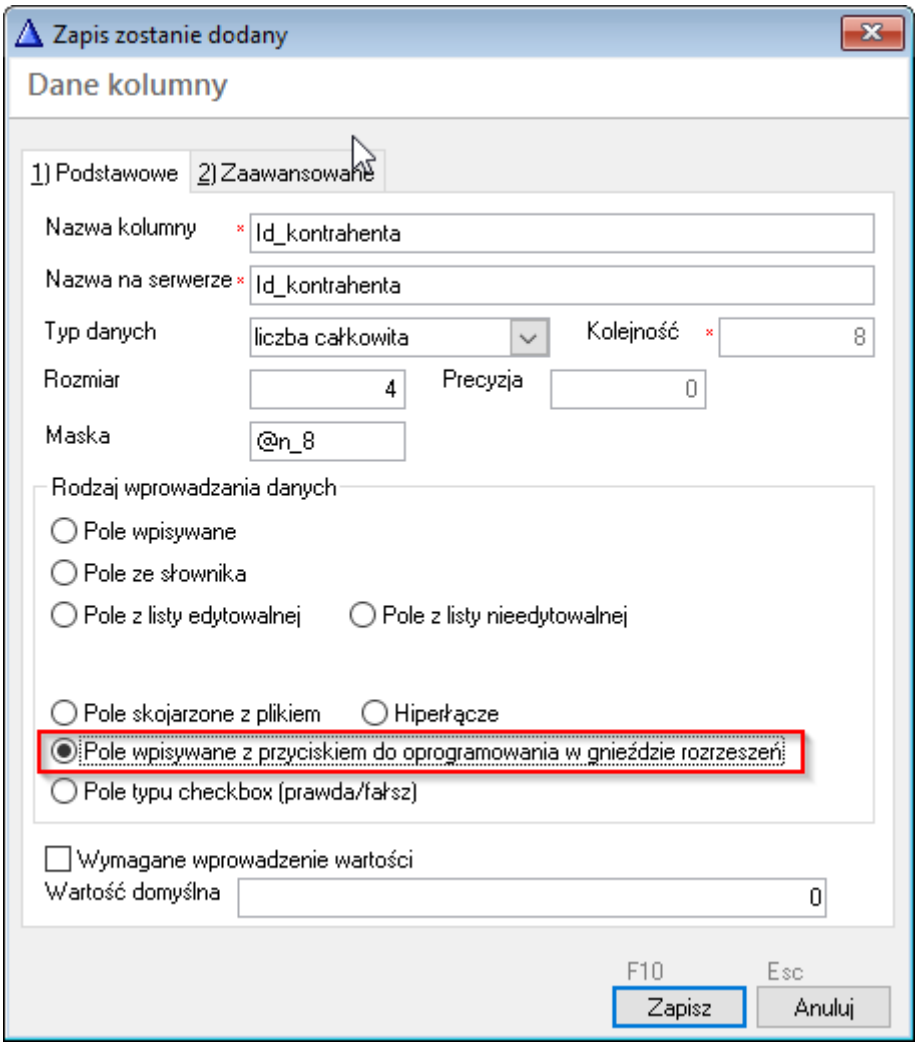
mogą być zmieniane w procedurach składowanych wykonywanych w gniazdach formularza ponieważ zostały zadeklarowane jako „SQL Output". W efekcie w prosty sposób możemy inicjować wartości poszczególnych pól formularza tuż po otwarciu okna albo wykorzystać gniazdo *Pole formularza: <nazwa pola> \ Po* do ustalenia wartości pozostałych pól formularza po wypełnieniu pola <nazwa pola>.

Gniazda uruchamiane po zatwierdzeniu pola formularza są szczególne użyteczne gdy chcemy wyliczyć na formularzu wartość jednego z pól na podstawie wartości pozostałych pól formularza. Dzięki tym gniazdom możemy tworzyć niemal dowolne zależności między poszczególnymi polami formularza.

Oczywiście w gniazdach dostępnych na listach i formularzach tabel dodatkowych możemy uruchamiać inne formularze tabel, które również mogą zawierać kod obsługi gniazd rozszerzeń (także uruchamiający kolejne formularze). Można więc tworzyć kaskadowe wywołania list i formularzy tabel dodatkowych. Oznacza to, że można konstruować zaawansowane zależności między tabelami dodatkowymi (relacje).

9. Gniazda rozszerzeń dla pól formularza –pole wpisywane z przyciskiem do oprogramowania w gnieździe rozszerzeń.

W trakcie definicji kolumny w danej tabeli dodatkowej do dyspozycji mamy opcję generowania gniazda rozszerzeń dla konkretnego pola – „Pole wpisywane z przyciskiem do oprogramowania" (Rys 53). Opcja ta generuje tylko jeden typ gniazda rozszerzeń „PO". Oznacza to, że po naciśnięciu przycisku może wykonać się zestaw oprogramowanych operacji. Opcja ta będzie najczęściej używana tego by wywołać w tym miejscu np.: swój własny słownik lub inna tabelę dodatkową. Opcja ta będzie alternatywą do aktualnie istniejących słowników w postaci słowników pól dodatkowych. Gniazda te działają na poziomie zdefiniowanego formularza.

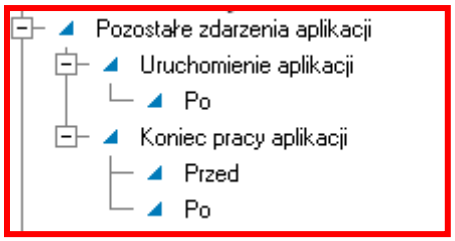


Rys 53. Definicja przycisku do oprogramowania w gnieździe rozszerzeń.

Pozostałe zdarzenia aplikacji

W grupie zadań „Pozostałe zdarzenia aplikacji" znajdują się gniazda rozszerzeń (Rys 54) związane z funkcjonowaniem całej aplikacji tj.:

- Uruchomienie aplikacji (Po);
- Koniec pracy aplikacji (Przed i Po);



Rys 54. Pozostałe zdarzenia aplikacji

Powyższe gniazda mogą być bardzo użyteczne dla funkcjonowania własnych rozszerzeń. W gnieździe Pozostałe zdarzenia aplikacji\Uruchomienie aplikacji\Po można umieszczać wszystkie funkcje inicjujące

i sprawdzające stan aplikacji (rozszerzeń własnych), które wykonają się po zalogowaniu użytkownika i załadowaniu z bazy danych ustawień konfiguracyjnych programu.

W gnieździe Pozostałe zdarzenia aplikacji\Koniec pracy aplikacji\Przed można np. zablokować możliwość zakończenia aplikacji do czasu wykonania przez użytkownika określonych działań. Za pomocą funkcji Koniec z opcją przerwij wykonaną w tym gnieździe można uniemożliwić zakończenie programu (oczywiście należy uważać aby całkowicie nie zablokować możliwości zamknięcia aplikacji !). Można też np. wyświetlić przypomnienie o czynnościach administracyjnych (kopii bazy danych) albo konieczności wygenerowania zestawień jeśli nie zostały zrobione.

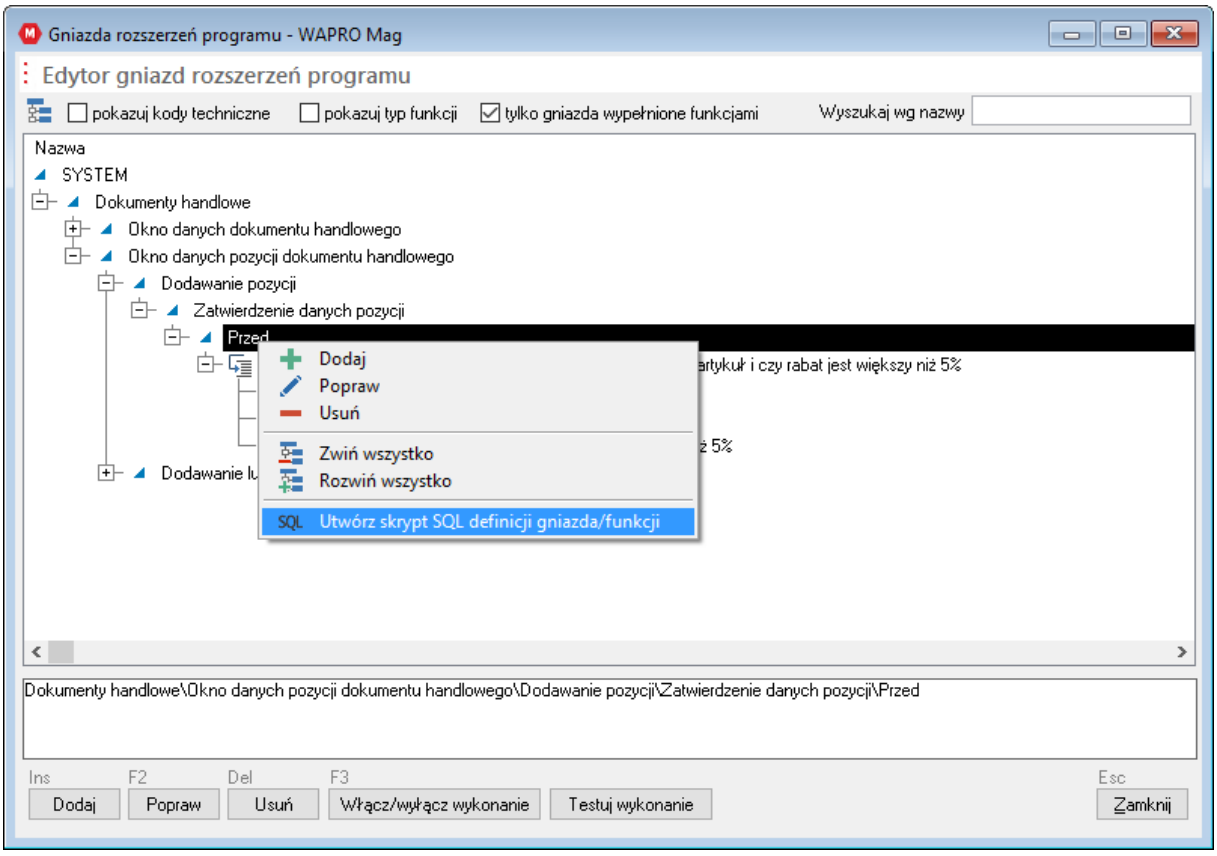
Z kolei jeśli zaistnieje potrzeba zapisania danych (np. przepisania między bazami danych, zmiany parametrów konfiguracyjnych użytkownika) po zakończeniu pracy użytkownika należy skorzystać z gniazda Pozostałe zdarzenia aplikacji\Koniec pracy aplikacji\Po. Co ważne, użytkownik aplikacji nie może przerwać wykonania kodu w tym gnieździe. Jest to więc dobre miejsce dla umieszczenia rozszerzeń, które muszą wykonać się po poprawnym zamknięciu aplikacji.

10. Tworzenie skryptów SQL instalujących funkcje w gniazdach

Skrypt SQL instalujący kod w gniazdach na innej instancji bazy danych można utworzyć dla dowolnej funkcji lub gniazda. Operacja tworzenia skryptu dostępna jest pod prawym przyciskiem myszy pod nazwą „Utwórz skrypt SQL definicji gniazda/funkcji" (Rys 55).

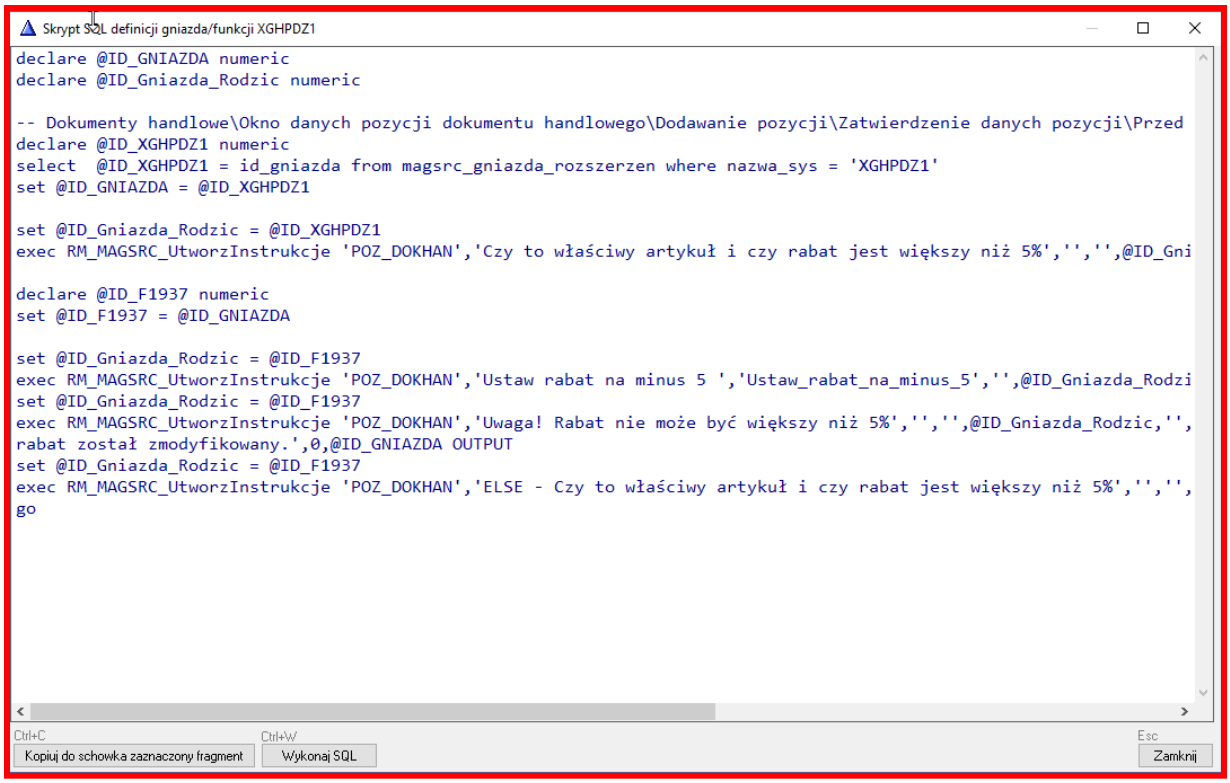
Zakres działania operacji tworzenia skryptu zależy od miejsca (pozycji) kursora w edytorze gniazd rozszerzeń i dotyczy wszystkich elementów podrzędnych drzewa. Dla instrukcji złożonej IF (warunek) THEN ELSE nie jest istotna pozycja kursora tzn. nieważne czy kursor zostanie ustawiony na gałęzi IF THEN czy na ELSE zostanie utworzony skrypt dla całej logicznie powiązanej paczki funkcji czyli tak jakby kursor stał na elemencie IF THEN. Ustawienie pozycji kursora na gałęzi SYSTEM i uruchomienie tworzenia skryptu spowoduje w efekcie przygotowanie instalacji dla wszystkich gniazd w programie.

Skrypt konstruowany jest z wykorzystaniem kodów technicznych gniazd, które są identyczne w każdej instancji bazy danych. Kody techniczne budowane są w taki sposób, że przechowują jednocześnie w nazwie adres umiejscowienia gniazda. Dodatkowo dla każdego gniazda adres w postaci opisowej podawany jest jako komentarz SQL celem ułatwienia orientacji w wygenerowanym skrypcie. Funkcje umieszczane w gniazdach mają również nadawane automatycznie kody techniczne z tym, że zasadą jest iż kod techniczny funkcji składa się z litery F oraz identyfikatora ID_GNIAZDA z tabeli MAGSRC_GNIAZDA_ROZSZERZEN. Z tego powodu ważna jest kolejność funkcji dodawanych w skrypcie SQL – kod techniczny danej funkcji będzie zapewne inny w innej instancji bazy danych ponieważ opiera się na identyfikatorze generowanym przez serwer SQL. Z tego powodu zaleca się stosować nazwy funkcji unikalne w ramach instancji bazy aby jednoznacznie identyfikować funkcje.



Rys 55. Tworzenie skryptu SQL definicji gniazda/funkcji

Skrypt SQL jest zabezpieczony przed wielokrotną instalacją do bazy danych. Procedura składowana RM_MAGSRC_UtworzInstrukcje wykorzystywana przy konstrukcji skryptu SQL instalacji kodu gniazda (Rys 56) sprawdza czy istnieje instrukcja (funkcja) posiadająca wszystkie dane takie jak podawane parametry w tabeli MAGSRC_GNIAZDA_ROZSZERZEN. Jeśli taka funkcja istnieje wówczas nie zostanie ponownie dodana. Jest to kolejny ważny argument za stosowaniem unikalnych nazw funkcji w ramach instancji. Jeśli w gnieździe zdefiniujemy 2 funkcje o identycznych parametrach a w szczególności o takiej samej nazwie i podpiętych do tej samej gałęzi drzewa (tego samego gniazda) to po utworzeniu skryptu i jego uruchomieniu na innej instancji utworzy się tylko jedna funkcja – druga ponieważ ma identyczne parametry jak pierwsza – zostanie przez procedurę RM_MAGSRC_UtworzInstrukcje całkowicie pominięta.



Rys 56. Przykład skryptu SQL instalującego funkcje w gniazdach

Podczas przygotowywania rozwiązania docelowego dla klienta należy pamiętać o kolejności wykonywania skryptów SQL. Zasadniczo należy wykonywać na bazie docelowej skrypty wygenerowane z bazy testowej w kolejności:

1) Skrypty tworzące tabele dodatkowe;

2) Skrypty instalujące definicje funkcji w gniazdach rozszerzeń oraz tabele dodatkowe

Zmiana kolejności instalacji skryptów może spowodować, że przygotowane rozwiązanie po prostu nie będzie działało. Wynika to ze sposobu działania gniazd rozszerzeń:

- gniazda dla operacji dodatkowych tworzone są podczas dodawania operacji dodatkowych;
- definicja funkcji typu Tabela dodatkowa lub Formularz tabeli zakłada istnienie poprawnej definicji tabeli w bazie danych;