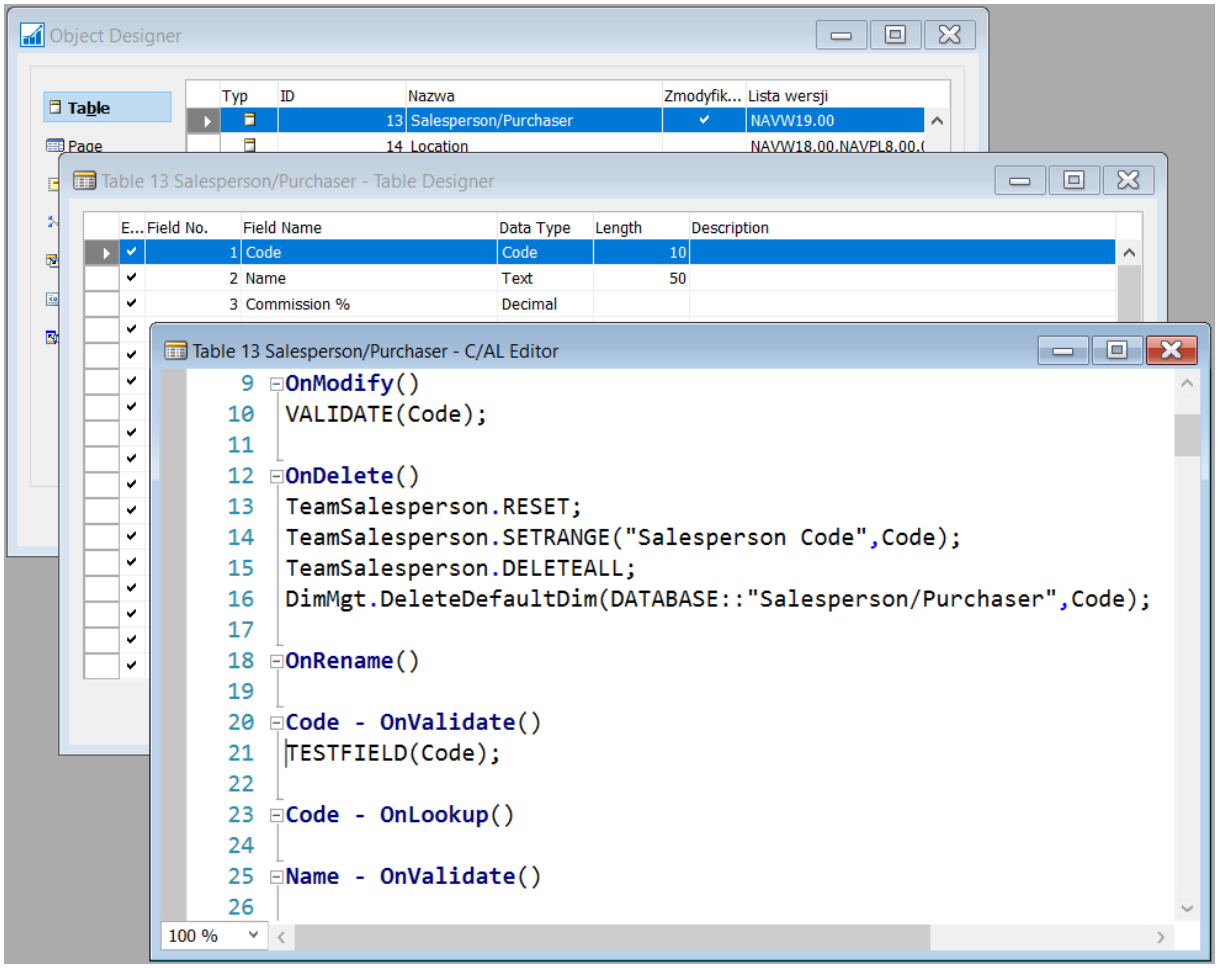
Informatyzacja Przedsiębiorstw

Microsoft Dynamics NAV 2016
Development Environment – C/AL

Plan zajęć

1 Informacje ogólne dotyczące środowiska deweloperskiego C/SIDE

- **WAŻNE: Oglądając w trybie „Design" jakiekolwiek obiekty standardowe** (czyli wszystkie o identyfikatorach spoza przydzielonej nam puli 50 000-50 099), należy ostatecznie wychodzić z nich **bez dokonania zapisu** – chyba, że w ćwiczeniu wyraźnie nakazano zapisanie jakichś modyfikacji w danym obiekcie. **Zapisanie przypadkowych zmian może poskutkować koniecznością przerobienia wszystkich ćwiczeń ponownie na nowej – czystej – bazie.**
- Kod będziemy pisać w **wyzwalaczach (triggerach) tabel** (nie należy go pisać w page'ach!). Przez Object Designera wybieramy obiekt, np. tabelę standardową 13 Salesperson/Purchaser, klikamy **F9** (lub **Widok→ C/AL Code**) i znajdujemy się w wyzwalaczach. Na początku umieszczone są wyzwalacze dotyczące ogólnie tabeli (obiektu): OnInsert, OnModify, OnDelete, OnRename, dalej każdego z pól: OnValidate, OnLookup.
  Przykład kodu:



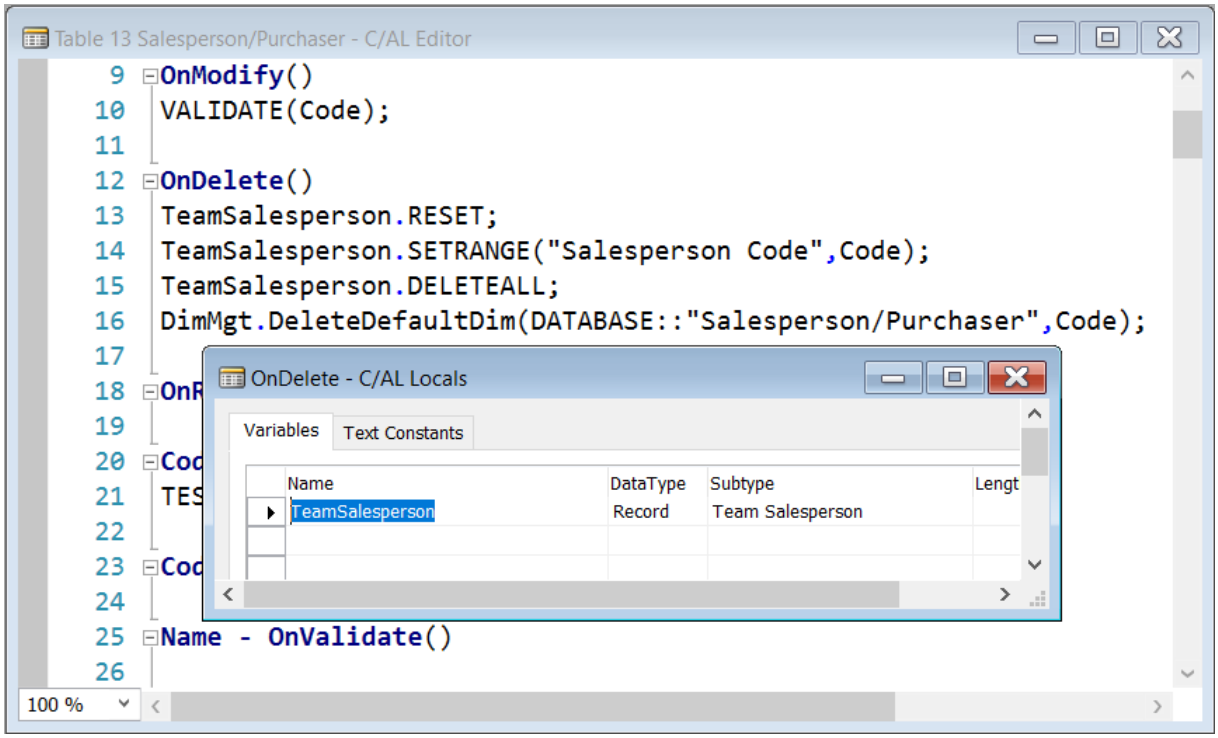
Na zrzucie ekranu widać np. trigger OnModify tabeli – kod w nim umieszczony spowoduje, że każda modyfikacja rekordu tabeli (dowolnego pola w rekordzie) skutkuje wykonaniem kodu umieszczonego w triggerze OnValidate pola Code. Funkcja TESTFIELD z jednym parametrem (nazwą pola) sprawdza, czy pole to nie zostało puste. Jeśli tak, to zgłasza błąd – wymagając uzupełnienia.

Trigger OnDelete – wykonywany w momencie usuwania rekordu z tabeli – zawiera kod, który usuwa z powiązanej tabeli Team Salesperson, wszystkie rekordy związane z usuwanym sprzedawcą.

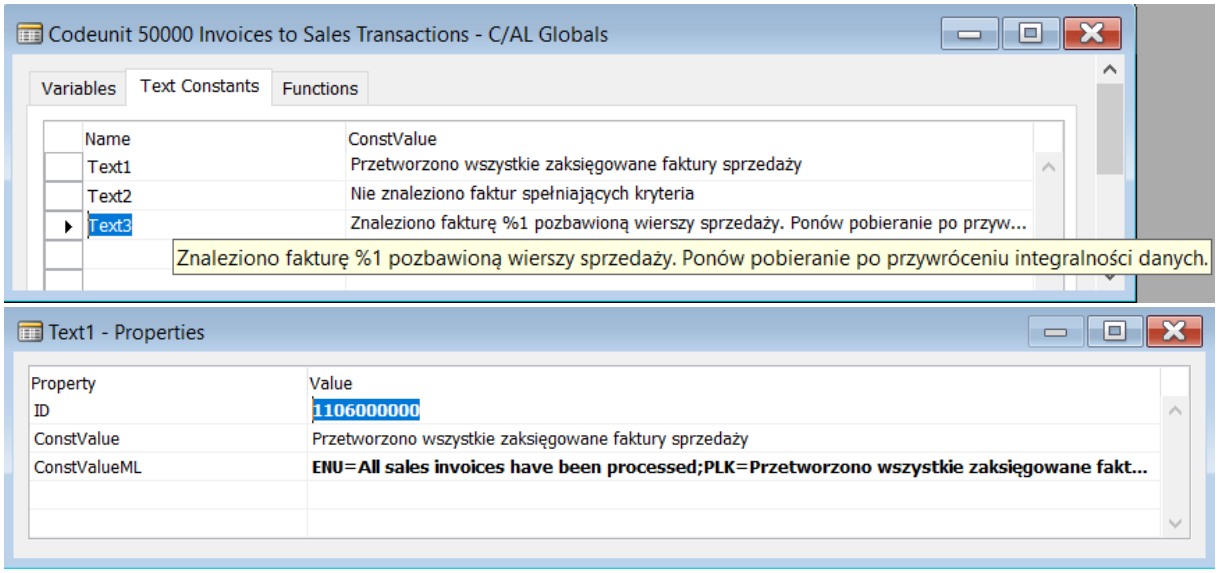
Trigger OnValidate jest standardowo uruchamiany po edycji pola z poziomu aplikacji użytkownika.

1.1 Zmienne

- Zmienne lokalne i globalne definiujemy w **Widok→ C/AL Locals** albo **Widok→ C/AL Globals**
- Przydatne są zmienne lokalne typu Rekord odwołujące się do tabel. Należy pamiętać, że zasięg zmiennych lokalnych jest ograniczony do danego triggera.
  Przykład – widok zmiennych lokalnych triggera OnDelete:



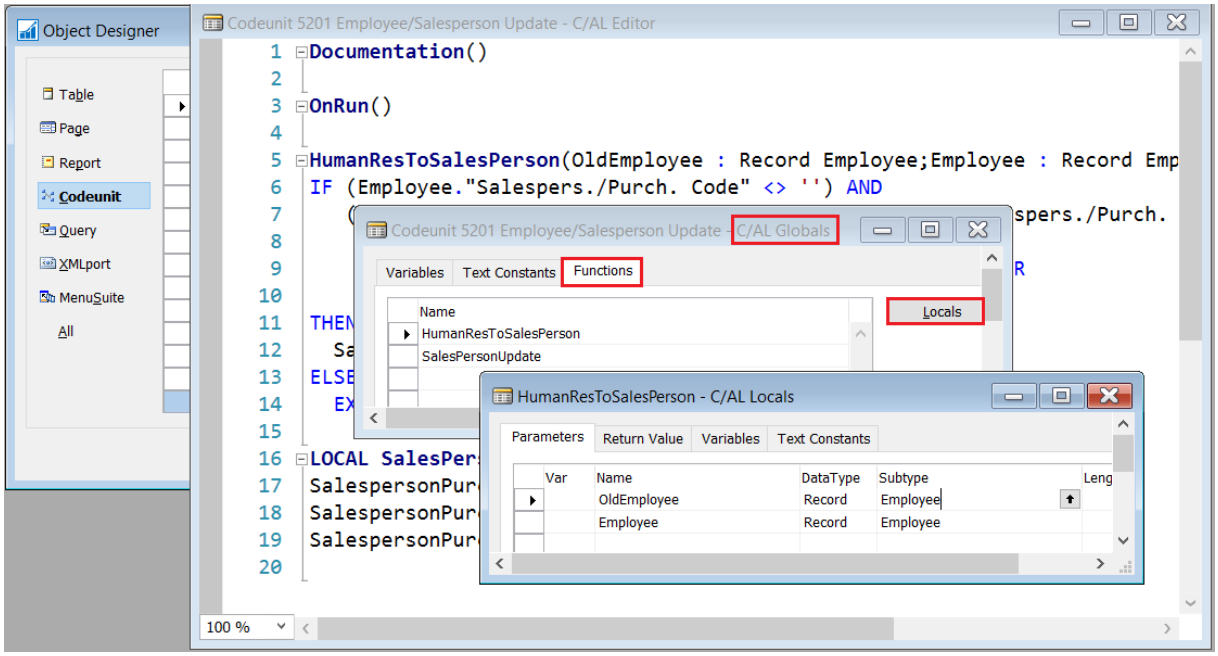
- Jako zmienne globalne najczęściej definiuje się stałe tekstowe (Text Constants), w których umieszcza się wielojęzyczne treści komunikatów (np. powiadomień lub ostrzeżeń o błędach).
  Przykład:



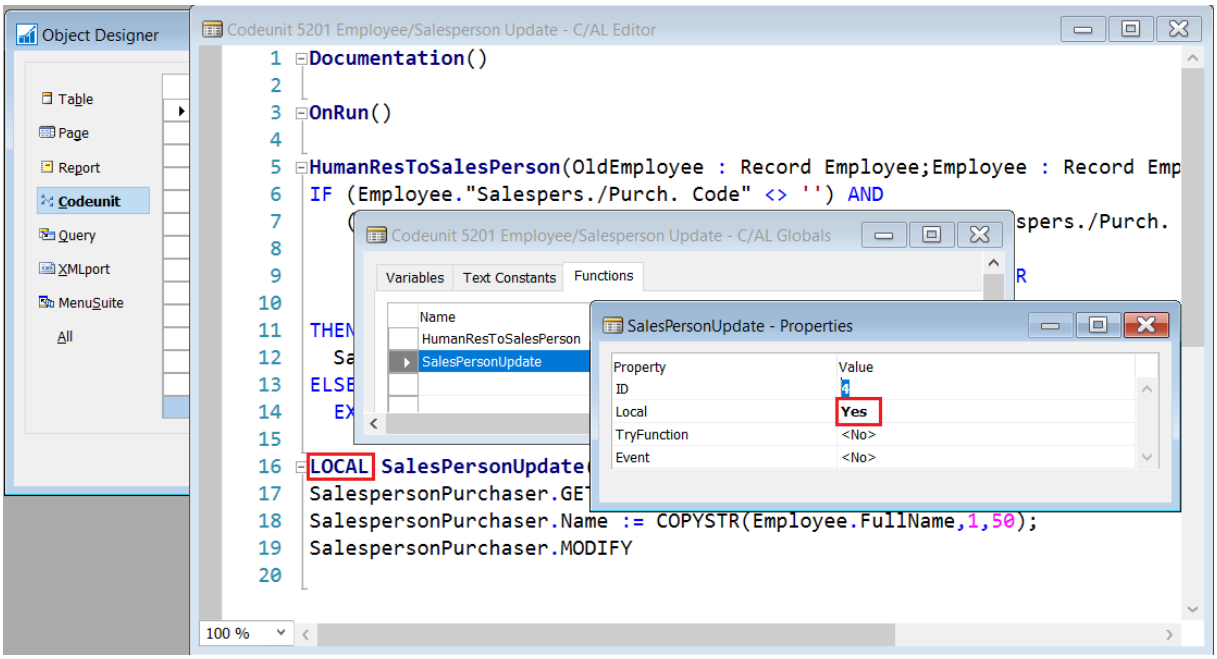
1.2 Funkcje

- Funkcje (np. w ramach Codeunit'ów – obiektów zawierających jednostki kodu) tworzy się jako zmienne globalne (menu **Widok→ C/AL Globals)**.

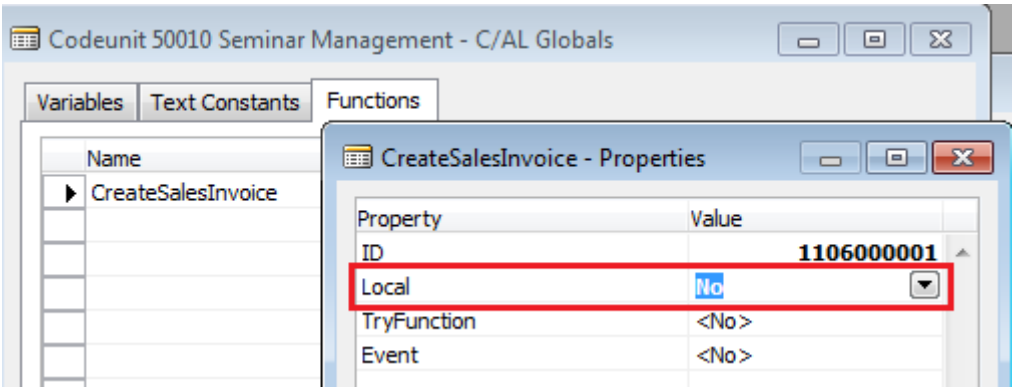
- Jeśli funkcja ma mieć jakiś parametr wejściowy, definiuje się go poprzez przycisk „Locals". Parametry mogą być przekazywane przez wartość – jak na przykładzie poniżej – albo przez referencję (jeśli zaznaczona jest opcja „Var").
  Przykład:



- Domyślnie, utworzone funkcje są widoczne tylko w obiekcie (np. Codeunicie) i nie można się do nich odwołać z innego miejsca. Funkcje mają wówczas ustawioną własność Local na „Yes”, a ich definicja poprzedzona jest słowem „Local”:



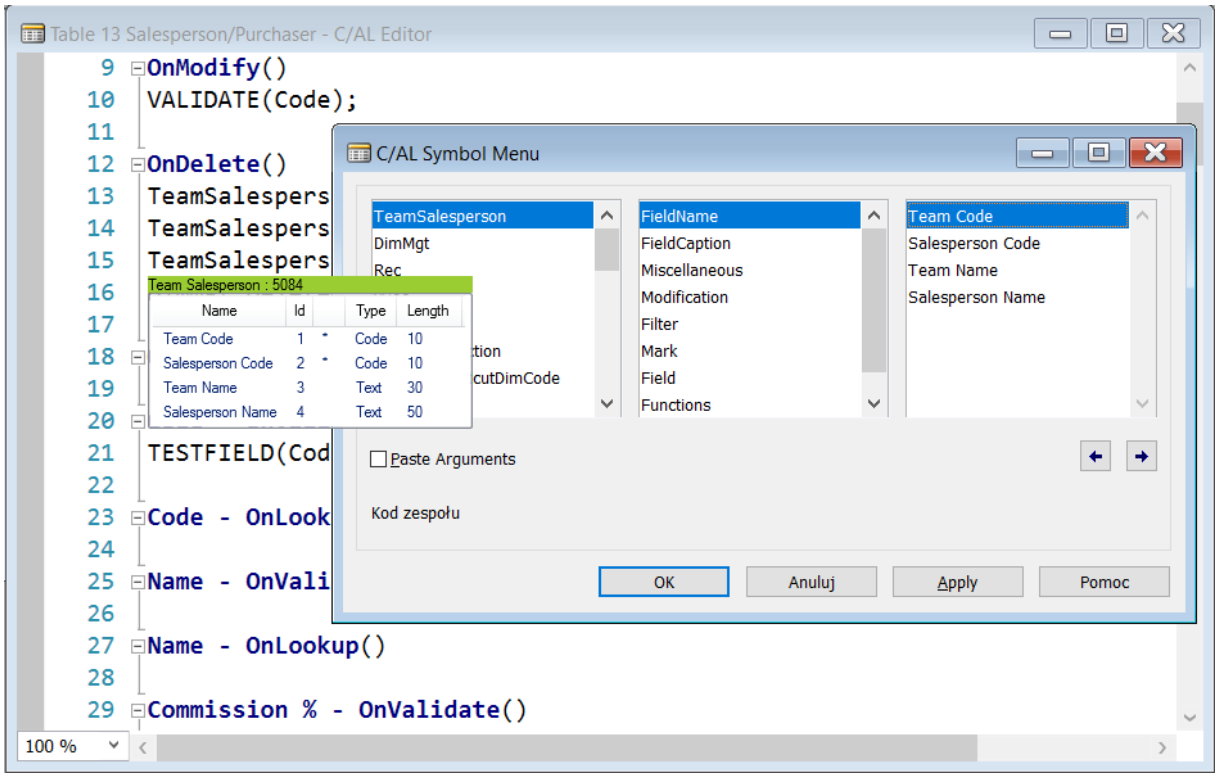
Aby z innego obiektu (np. page’a) można było odwoływać się do funkcji zdefiniowanej wewnątrz np. Codeunit’u trzeba ustawić jej własność Local na „No”:



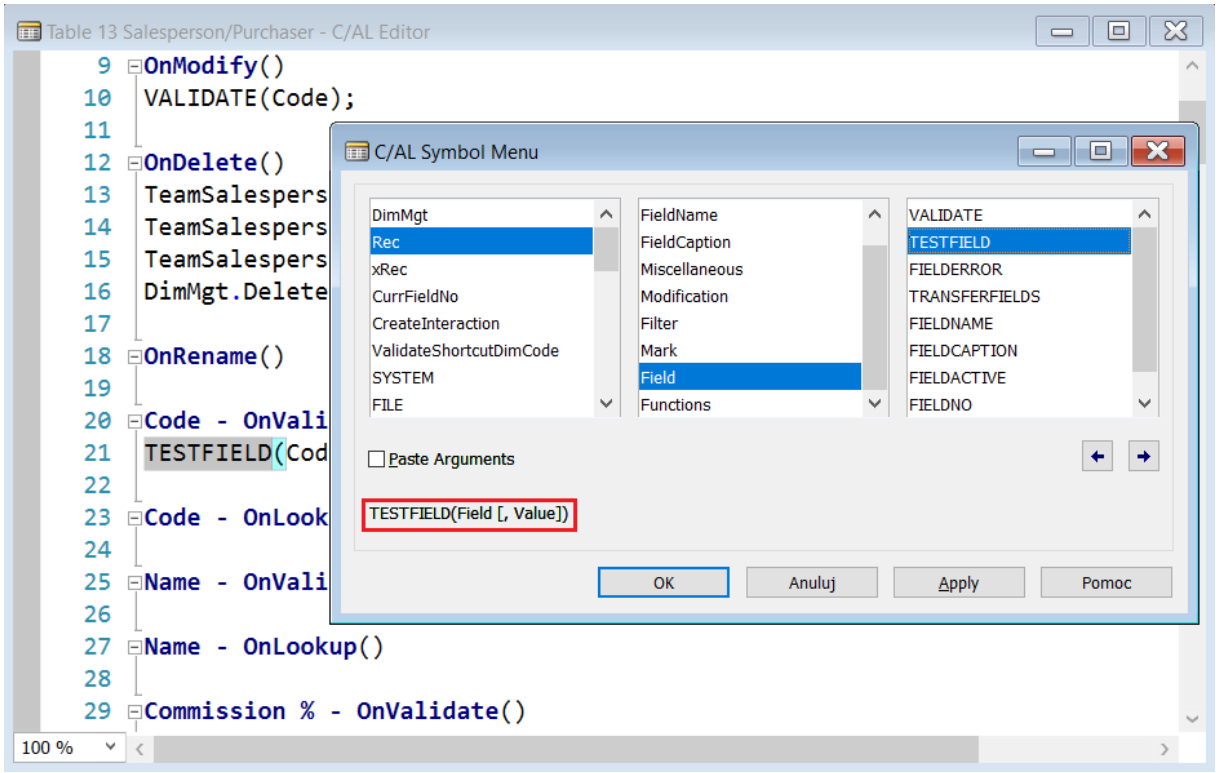
Wówczas można uruchomić funkcję zdefiniowaną w Codeunicie np. spod Akcji na page’u.

1.3 C/AL Symbol Menu

- Podczas pisania kodu wyświetlane są podpowiedzi.
  Można też korzystać z C/AL Symbol Menu (**F5** lub menu **Widok→ C/AL Symbol Menu**).
  Przykład:

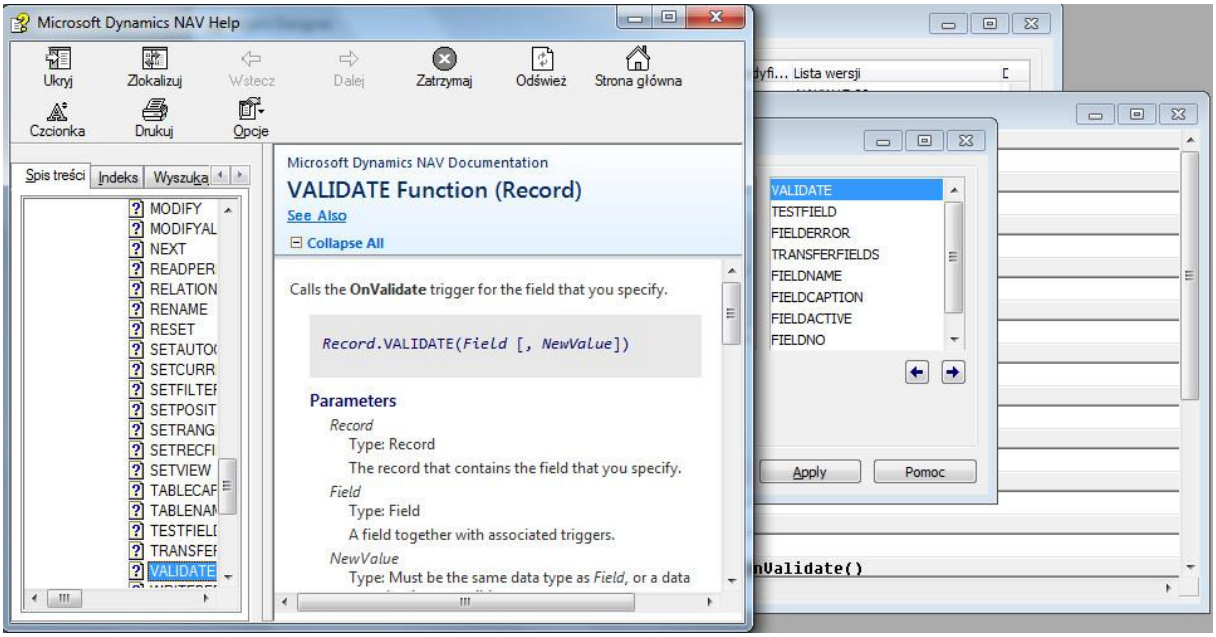


Przykład z wywołaniem funkcji TESTFIELD dla pola Code. Na dole – parametry funkcji.



1.4 Pomoc kontekstowa i Debugger

- W czasie pisania kodu pod klawiszem **F1** jest dostępna pomoc kontekstowa m.in. z przykładami wywołania funkcji, opisami parametrów, polami tabel i zmiennych.

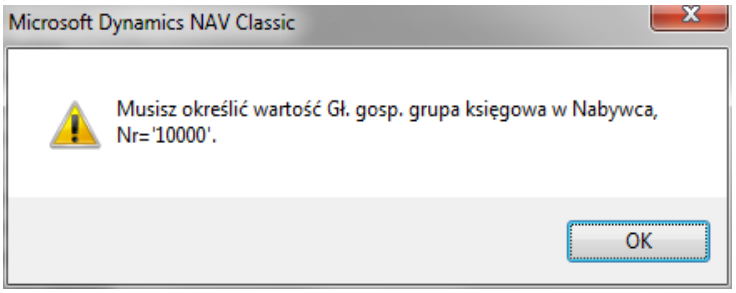


- Debugger uruchamiamy z menu **Narzędzia→Debugger→Sesja debugowania...** (SHIFT +CTRL +F11)

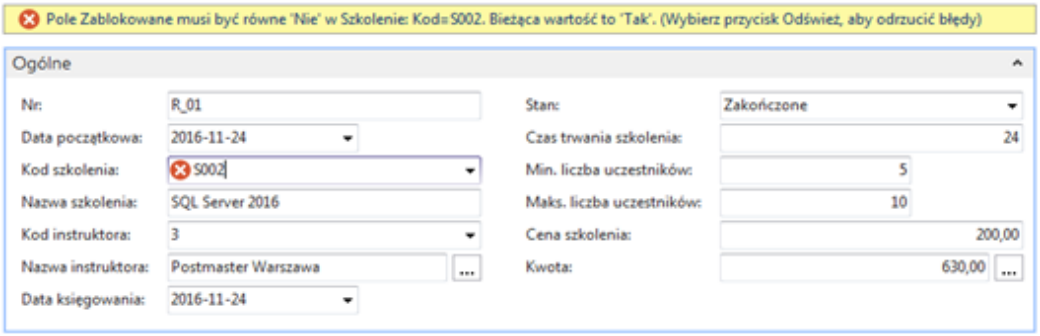
1.5 Przydatne funkcje

- **Przydatne funkcje:**
  - **GET** //pobiera **jeden rekord** po kluczu głównym (**nigdy po innym atrybucie**)
  - **RESET** //zdejmuje wszystkie filtry wcześniej nałożone na zmiennej rekordowej
  - **SETRANGE** //ustawia filtry na zmiennej rekordowej wg wartości wybranego pola; najlepiej filtrować po jakimś kluczu, więc przed SETRANGE często pojawia się też funkcja SETCURRENTKEY ustawiająca ten klucz
  - **SETFILTER** //działa podobnie jak SETRANGE, ale daje więcej składniowych możliwości definiowania filtrowania
  - **FINDFIRST**, **FINDLAST**, **FINDSET** //pobiera jeden (pierwszy, ostatni) lub wiele rekordów
  - **REPEAT UNTIL NEXT**=0 //typowa pętla pozwalająca na przeiterowanie po znalezionych rekordach
  - **TESTFIELD** //wywołana z jednym parametrem sprawdza, czy pole rekordu nie jest puste, z dwoma parametrami sprawdza, czy pole przyjmuje wartość drugiego parametru. Jeśli nie, to wyświetla komunikat informujący, czego dotyczy błąd i przerywa działanie wycofując wprowadzone zmiany. Skopiowanie tekstu komunikatu (Ctrl+C) przydaje się szczególnie wtedy, gdy komunikat dotyczy nabywcy czy faktury ze skomplikowanym numerem, który inaczej trzeba by przepisać.

```
Customer.GET('10000');
Customer.TESTFIELD("Gen. Bus. Posting Group");
```

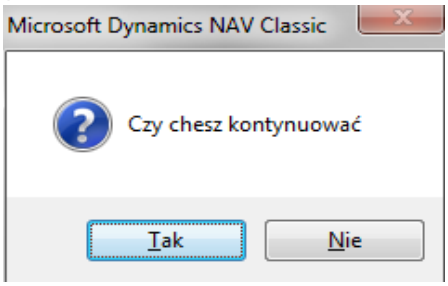



```
Seminar.GET ("Seminar Code") THEN BEGIN
Seminar.TESTFIELD(Seminar.Blocked, FALSE);
```



- **CONFIRM**, **MESSAGE**, **ERROR** //również umożliwiają wyświetlenie komunikatów, przy czym funkcja ERROR wyświetla odpowiedni komunikat, po czym przerywa działanie wycofując wprowadzone zmiany.

```
IF CONFIRM('Czy chesz kontynuować', TRUE) THEN
  MESSAGE('OK')
ELSE
  ERROR('Akcja przerwana');
```



- **Różnica między zwykłym podstawieniem, a funkcją VALIDATE**
  - **Name := Customer.Name** //pod pole Name zostanie podstawiona wartość pola Name ze zmiennej rekordowej Customer
  - **VALIDATE(Name, Customer.Name)** //po podstawieniu pod Name wartości pola Name ze zmiennej rekordowej Customer, zostanie wywołany trigger OnValidate w polu Name. Jeśli w kodzie triggera zamieszczono funkcję (np. ERROR, TESTFIELD, …), której wywołanie poskutkuje błędem, to (oprócz wyświetlenia odpowiedniego komunikatu) wycofane zostaną wszystkie zmiany wykonane w tym triggerze.

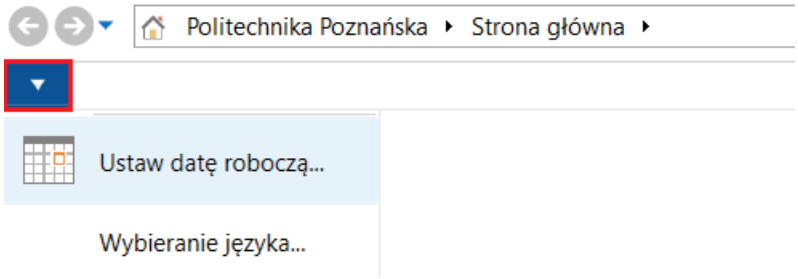
- **Różnica między triggerami OnLookUp i OnValidate pól**
  - **OnLookUp** jest wywoływany przy dostępie do pola rekordu, można w nim zamieścić np. jakieś filtrowanie. Uwaga: umieszczenie jakiegokolwiek kodu, czy nawet zdefiniowanie w tym triggerze jakiejkolwiek zmiennej lokalnej powoduje zaburzenie standardowego zachowania przycisku Lookup danego pola. Programujemy więc tylko, jeśli jest to rzeczywiście konieczne!
  - **OnValidate** jest wywoływany po podstawieniu wartości, po modyfikacji rekordu. Zamieszcza się tam często kod weryfikujący poprawność wpisanej przez użytkownika wartości i/lub realizujący inne zmiany, które mają wynikać z uzupełnienia danego pola daną wartością.

2 Przykłady kodu w triggerach tabel i ich pól

- Trigger OnModify tabeli – kiedy rekord tabeli zostanie zmodyfikowany, to jej pole „Last Date Modified” (typu Date) jest automatycznie uzupełniane datą roboczą.

```
OnModify()
"Last Date Modified" := WORKDATE;
```

Wartość daty roboczej ustawia się w aplikacji użytkownika.



Zamiast funkcji systemowej WORKDATE, można rozważyć użycie funkcji TODAY (bieżąca data). Do uzupełniania pól typu Time wykorzystuje się funkcją systemową TIME.

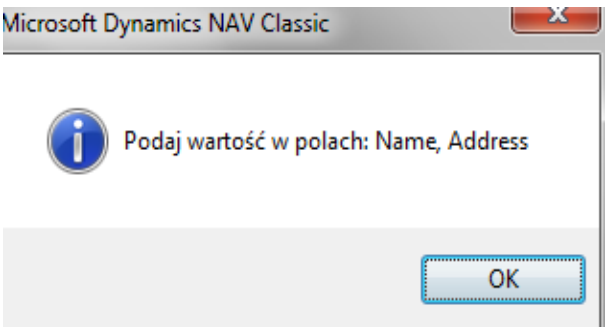
- Trigger OnValidate pola „Name” – kiedy użytkownik wprowadzi lub zmieni nazwę w polu „Name” (typu Text), to w pole „Search Name” (typu Code) wstawiana jest ta nowa nazwa pisana wielkimi literami.

```
Name - OnValidate()
IF ("Search Name" <> UPPERCASE(Name)) THEN
  "Search Name" := Name;
```

Należy pamiętać, aby teksty komunikatów podawać wielojęzycznie definiując odpowiednie stałe tekstowe (przykład w 1.1). Ich zakodowanie na sztywno, jak w poniższym przykładzie, powoduje wyświetlanie dziwacznych komunikatów.

```
MESSAGE('Podaj wartość w polach: %1, %2', FIELDNAME(Name), FIELDNAME(Address));
```

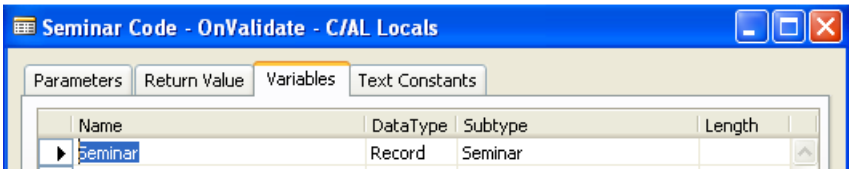
W aplikacji użytkownika wybrano język angielski – co widać, po wyświetlonych w komunikacie nazwach pól „Name” i „Address”, których nazwy są parametrami %1 i %2 funkcji MESSSAGE. Gdyby wybrano język polski, to funkcja FIELDNAME zwróciłaby polskojęzyczne nazwy tych pól. Sam komunikat „Podaj wartość…” został jednak zakodowany na sztywno, i tak też się wyświetlił:



- Trigger OnValidate pola „Seminar Code” – kiedy użytkownik wprowadzi lub zmieni kod szkolenia w polu „Seminar Code” (typu Code), to funkcja GET pobierze do zmiennej rekordowej „Seminar” jeden rekord z tabeli Seminar po jej kluczu głównym (o wartości takiej, jak bieżąca wartość „Seminar Code”).

```
Seminar Code - OnValidate()
IF Seminar.GET("Seminar Code") THEN BEGIN
  Seminar.TESTFIELD(Seminar.Blocked,FALSE);
  "Seminar Name":=Seminar.Name;
  "Seminar Duration":=Seminar."Seminar Duration";
  "Minimum Participants":=Seminar."Minimum Participants";
  "Maximum Participants":=Seminar."Maximum Participants";
  "Seminar Price":=Seminar."Seminar Price";
 END ELSE BEGIN
  "Seminar Name":='';
  "Seminar Duration":=0;
  "Minimum Participants":=0;
  "Maximum Participants":=0;
  "Seminar Price":=0;
END;
```

Zmienna rekordowa „Seminar” musi być zdefiniowana – np. jako zmienna lokalna w triggerze OnValidate pola:



Funkcja Funkcja TESTFIELD sprawdzi, czy dla pobranego rekordu Seminar pole Blocked ma wartość FALSE (w polach typu Boolean są wartości yes/no, ale w kodzie należy użyć wartości true/false). Jeśli pole Blocked nie ma wartości FALSE, to zostanie wyświetlony domyślny komunikat (wielojęzyczny komunikat systemowy o standardowej treści – przykład w 1.5), po czym nastąpi przerwanie działania. Jeśli pole Blocked ma wartość FALSE, to pola: „Seminar Name”, „Seminar Duration”, „Minimum Participants”, „Maximum Participants”, „Seminar Price” zostaną uzupełnione odpowiednimi wartościami ze zmiennej rekordowej Seminar lub wartościami pustymi, jeśli nie pobrano rekordu.

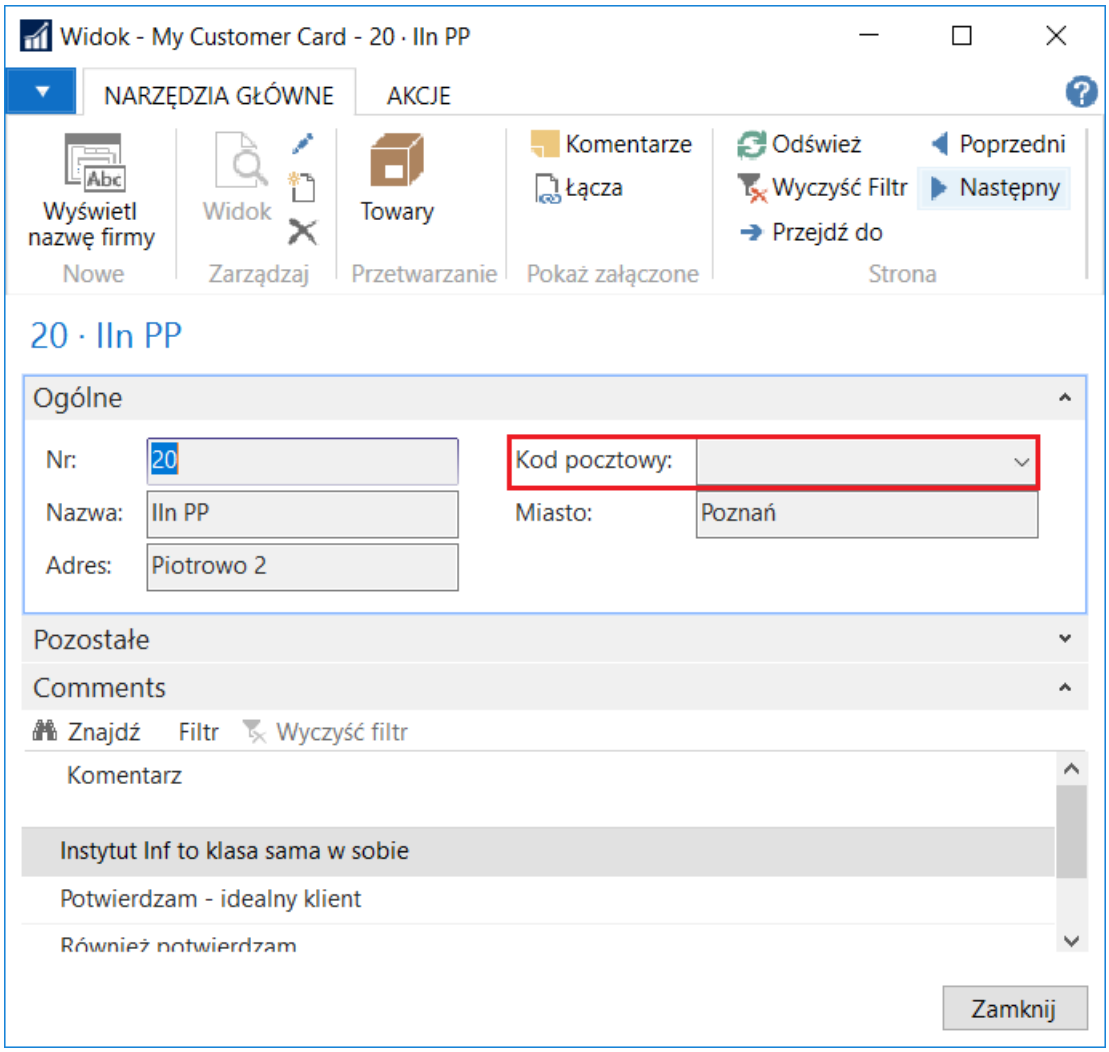
3 Zadania do samodzielnego wykonania

3.1 Uzupełnianie nazwy miejscowości na podstawie kodu pocztowego

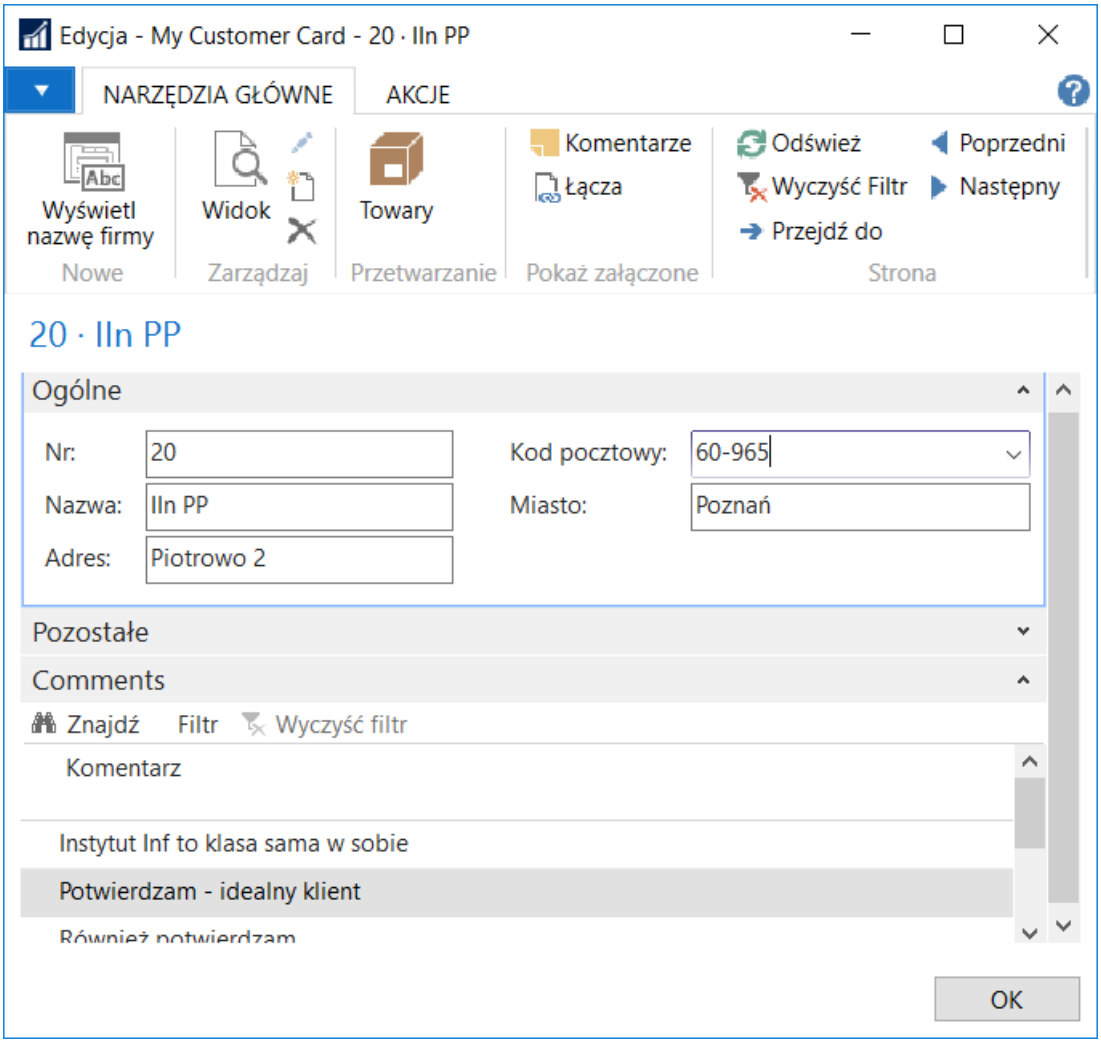
W tym ćwiczeniu chcemy zapewnić, by pole City w tabeli „My Customer Table” uzupełniało się na podstawie wybranego kodu pocztowego.

Ćwiczenie krok po kroku:

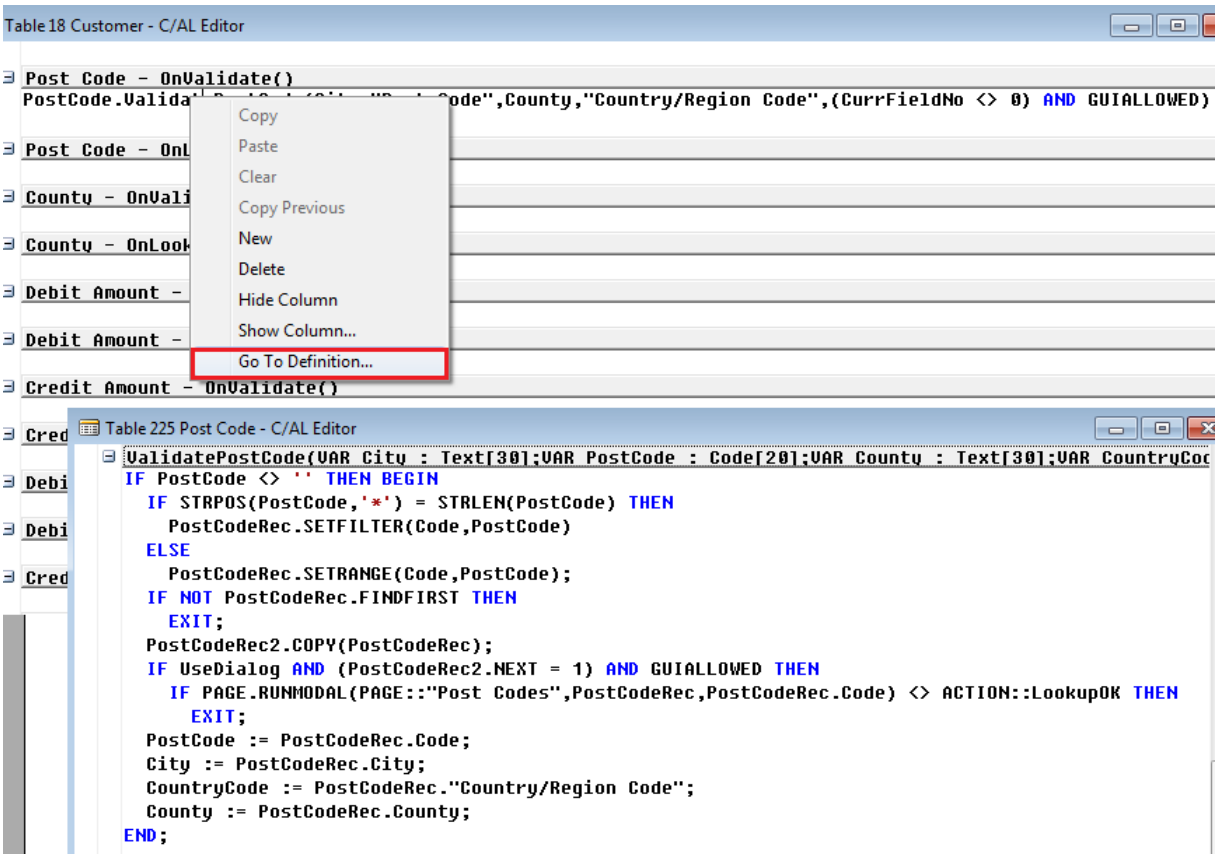
1. Uruchom „Object Designer” i przejdź do tabel.
2. Zapoznaj się z definicją tabeli standardowej „Post Code” (nr 225). Przejrzyj jej zawartość i dodaj kilka rekordów – znanych Ci kodów pocztowych wraz z odpowiadającymi im nazwami miejscowości.
3. Do tabeli „My Customer Table” (nr 50002) dodaj pole „Post code”, czyli „Kod pocztowy” (typ danych: Code, długość: 20) i ustaw we własnościach pola powiązanie (relację bezwarunkową) do tabeli standardowej „Post Code”.
4. Zwróć uwagę, że tabela „My Customer Table” zawierała już pola adres „Address” i miasto „City”. Nie zawiera natomiast innych danych adresowych jak, np. kod kraju „Country/Region Code”, czy hrabstwo „County” (gdyż nie uznano tych informacji za potrzebne).
5. Dodaj pole „Post Code” do odpowiedniego page'a typu kartoteka. Możesz je dodać przed polem "City".



6. Oprogramuj pole „Post Code” tabeli „My Customer Table” w ten sposób, by po wybraniu kodu pocztowego uzupełniło się w tabeli „My Customer Table” pole „City”. **Identyczna funkcjonalność jest zapewniona w tabeli standardowej „Customer” (nr 18), więc można przekleić wywołanie standardowej funkcji z tej tabeli** (znajduje się ono w triggerze OnValidate pola „Post Code”).
7. Dla prawidłowego wywołania przeklejonej funkcji zdefiniuj w wyzwalaczu OnValidate odpowiedniego pola tabeli „My Customer Table” zmienne lokalne:
   zmienną PostCode typu Record,
   zmienną "County" typu Text,
   zmienną "Country/Region Code" typu Code.
   Pozostałe parametry (City i "Post Code") są polami w aktualnym rekordzie.
8. Uruchom page'a kartoteka tabeli „My Customer Table” i sprawdź poprawność działania nowej funkcjonalności wybierając różne kody pocztowe.



Uwaga: Aby podejrzeć definicję funkcji „ValidatePostCode” klikamy prawym przyciskiem myszy w miejscu jej wywołania i wybieramy „Go To Definition…”.



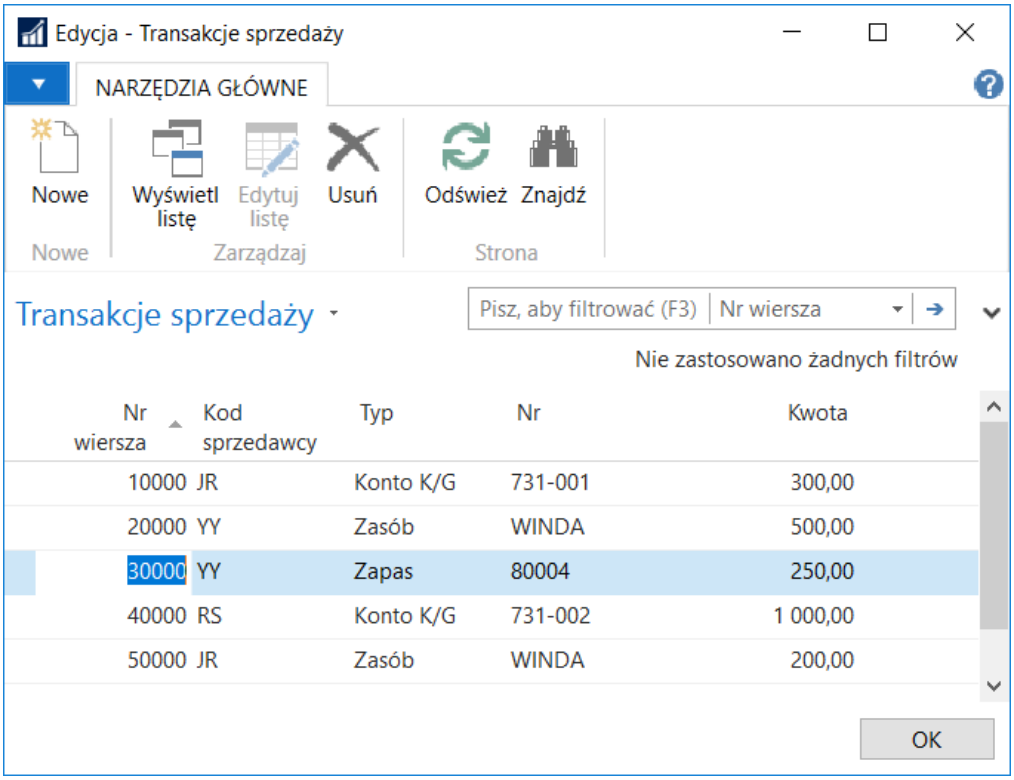
3.2 Automatyczne wypełnianie tabeli „Sales Transactions”

W tym ćwiczeniu chcemy dodać do tabeli „Sales Transactions” możliwość automatycznego tworzenia rekordów na podstawie faktycznych sprzedaży zaewidencjonowanych w zaksięgowanych w systemie fakturach sprzedaży.

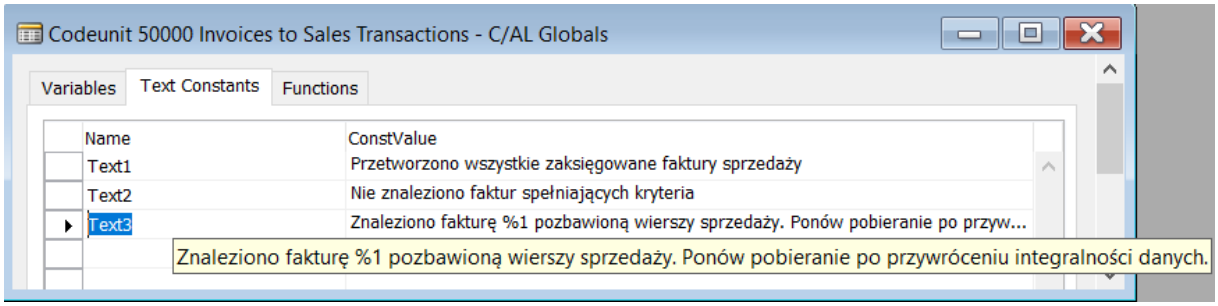
Uwaga: **Zaksięgowane FVS przechowywane są w tabeli standardowej „Sales Invoice Header” (nr 112) i tabeli standardowej „Sales Invoice Line” (nr 113)**. W pierwszej z tabel przechowywane są nagłówki faktur, w drugiej ich wiersze.

Ćwiczenie krok po kroku:

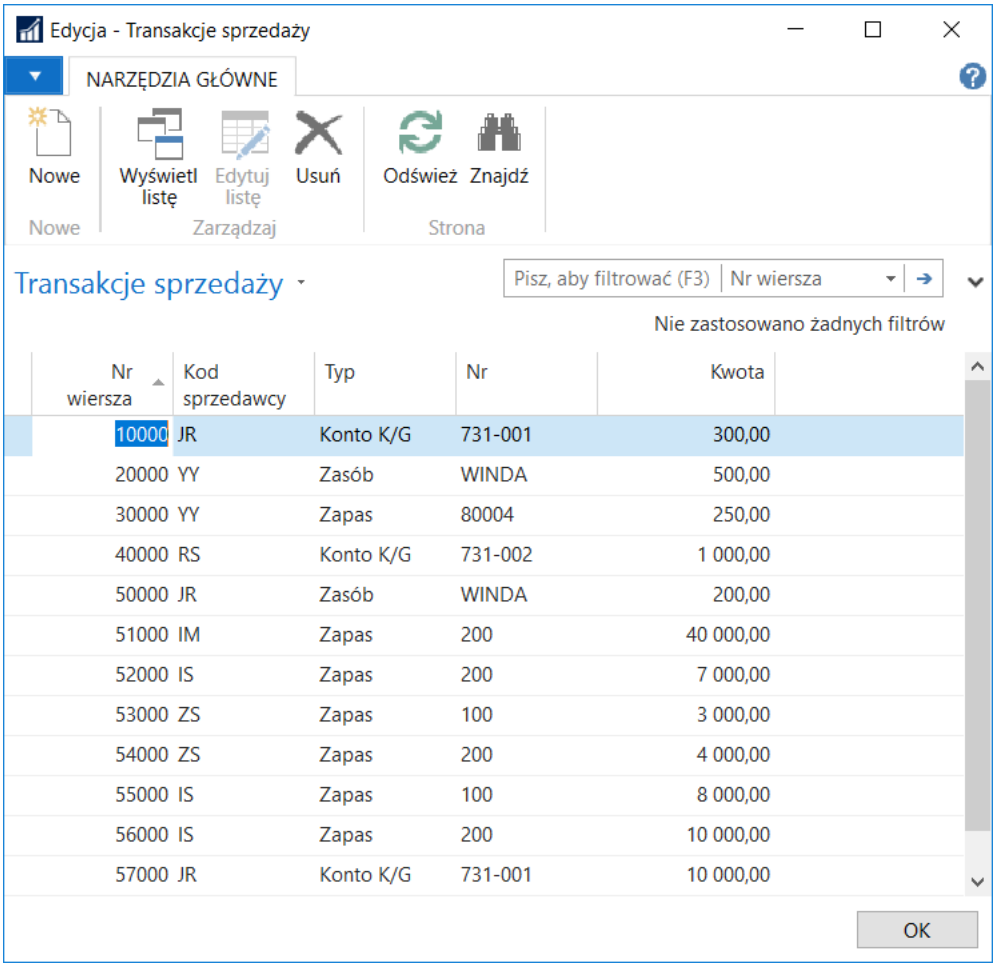
1. Uruchom aplikację użytkownika i obejrzyj zaksięgowane dotychczas faktury – Menu ***Działy→Zarządzanie Finansami→Należności→Historia→Zaksięgowane faktury sprzedaży*** warto zaksięgować jeszcze jedną FVS wybierając w nagłówku faktury kod sprzedawcy, który ma ustawioną prowizję >0 i dodając 2 lub 3 wiersze, każdy z różnymi towarami. Ewidencjonowanie faktur sprzedaży opisane było w materiałach IP_zadanie02_FK.
2. Uruchom „Object Designer” i przejdź do tabel.
3. Zapoznaj się z definicją tabeli standardowej „Sales Invoice Header” (nr 112). Przejrzyj jej kilka pierwszych pól. **Sprawdź, które pole jest jej kluczem głównym.**
4. Zapoznaj się z definicją tabeli standardowej „Sales Invoice Line” (nr 113). Przejrzyj jej kilka pierwszych pól. **Sprawdź, które pole jest jej kluczem głównym – czy jest to pojedyncze pole, czy klucz jest podwójny. Spróbuj ustalić, które pole tabeli wiąże ją z odpowiednim nagłówkiem?**
5. Przypomnij sobie definicję tabeli „Sales Transactions” (nr 50001). Spróbuj ustalić, które pola tabel standardowych „Sales Invoice Header” i „Sales Invoice Line” zawierają dane, które można zaciągnąć do tabeli „Sales Transactions”. Opis danych przechowywanych w tabeli „Sales Transactions” znajduje się w materiałach IP_Tabele. Sprawdź zawartość tabeli – może się nieznacznie różnić od przykładowej pokazanej poniżej, ale musi zawierać kilka wierszy.



6. Uruchom „Object Designer” i **utwórz nowy Codeunit** o nazwie „Invoices to Sales Transactions” („Transakcje sprzedaży z faktur”) i identyfikatorze 50 000.
7. W wyzwalaczu OnRun nowoutworzonego obiektu wklej przygotowany poniżej kod.
8. **Zdefiniuj wszystkie potrzebne do jego działania zmienne lokalne**, które zostały wskazane w komentarzach.
9. Zdefiniuj wszystkie potrzebne do wyświetlania komunikatów stałe tekstowe jako zmienne globalne Codeunitu. Zadbaj treść w obu językach: polskim i angielskim.

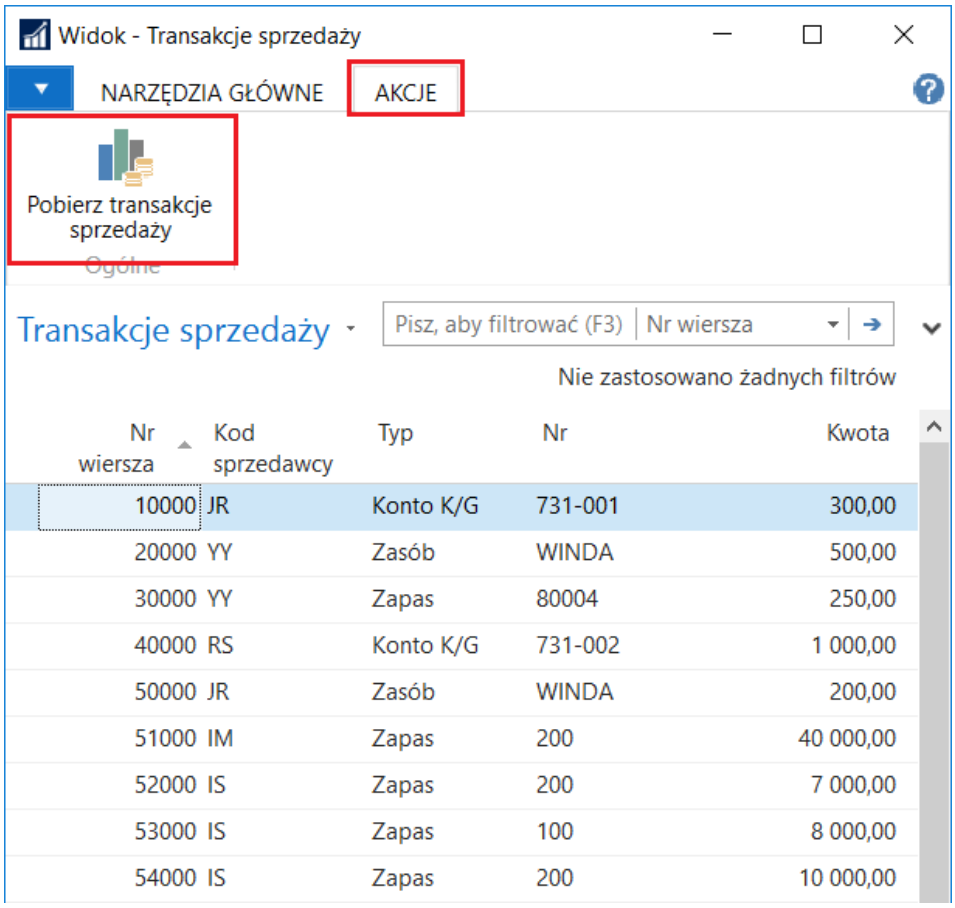


10. **Uzupełnij odpowiednio kod** w miejscach, które zostały wskazane w komentarzach.
11. Skompiluj, zapisz i zamknij utworzony Codeunit. Wypróbuj jego działanie (przycisk „Run”). Następnie sprawdź, czy w tabeli „Sales Transactions” pojawiły się nowe wiersze – zawierające dane pobrane z zaksięgowanych faktur FVS. Zawartość będzie się różnić od przykładowej pokazanej poniżej, ale powinna zawierać nowe wiersze wynikające ze sprzedaży komputerów – zapasów o numerach 100, 200 i 300.



Nr wiersza	Kod sprzedawcy	Typ	Nr	Kwota
10000	JR	Konto K/G	731-001	300,00
20000	YY	Zasób	WINDA	500,00
30000	YY	Zapas	80004	250,00
40000	RS	Konto K/G	731-002	1 000,00
50000	JR	Zasób	WINDA	200,00
51000	IM	Zapas	200	40 000,00
52000	IS	Zapas	200	7 000,00
53000	ZS	Zapas	100	3 000,00
54000	ZS	Zapas	200	4 000,00
55000	IS	Zapas	100	8 000,00
56000	IS	Zapas	200	10 000,00
57000	JR	Konto K/G	731-001	10 000,00

12. Uruchom page „Sales Transactions” w trybie projektowania, aby dodać przycisk umożliwiający uruchomienie Codeunitu „Invoices to Sales Transactions”. Dodawanie akcji w page’ach omówione zostało w materiałach IP_Pages.
13. Sprawdź, czy w zakładce „Akcje” page’a pojawił się przycisk uruchamiający stworzony Codeunit. Może on mieć inna ikonkę niż w przykładzie poniżej. Sprawdź działanie nowej akcji. W tabeli powinny się pojawić kolejne wiersze – będące powtórzeniem poprzednich.



Nr wiersza	Kod sprzedawcy	Typ	Nr	Kwota
10000	JR	Konto K/G	731-001	300,00
20000	YY	Zasób	WINDA	500,00
30000	YY	Zapas	80004	250,00
40000	RS	Konto K/G	731-002	1 000,00
50000	JR	Zasób	WINDA	200,00
51000	IM	Zapas	200	40 000,00
52000	IS	Zapas	200	7 000,00
53000	IS	Zapas	100	8 000,00
54000	IS	Zapas	200	10 000,00

Codeunit „Invoices to Sales Transactions”

```
LSalesInvoiceHeader.RESET; //zmienna lokalna rekordowa tabeli 112
LSalesInvoiceHeader.SETFILTER("Salesperson Code",'<>%1',''); //szukamy faktur, w
których podano kod sprzedawcy
IF LSalesInvoiceHeader.FINDFIRST THEN BEGIN
  REPEAT
    //MESSAGE('Znaleziono nagłówek faktury');
    LSalesperson.RESET; //zmienna lokalna rekordowa tabeli 13
    //TODO: po kluczu głównym pobrać do zmiennej LSalesperson rekord sprzedawcy z
nagłówka faktury
    IF (LSalesperson."Commission %" >0) THEN BEGIN //szukamy faktur sprzedawców o
prowizji >0
      LSalesInvoiceLine.RESET; //zmienna lokalna rekordowa tabeli 113
      //TODO: ustawić filtr na zmiennej rekordowej LSalesInvoiceLine, tak by
pobrać tylko te wiersze, które należą do bieżącego nagłówka faktury
      IF LSalesInvoiceLine.FINDFIRST THEN BEGIN
        REPEAT
          //MESSAGE('Znaleziono wiersz faktury');
          CLEAR(LSalesTransaction); // wyczyszczenie całej zawartości zmiennej
rekordowej tabeli Sales Transactions - wraz z wartością klucza głównego
          LSalesTransaction.RESET;
          IF LSalesTransaction.FINDLAST THEN //przejście do ostatniego rekordu
            LineNo:=LSalesTransaction."Line No."; // zmienna lokalna typu Integer

          LineNo:=LineNo+1000; //inkrementacja nr wiersza o 1000 (zamiast 10000)

          CLEAR(LSalesTransaction); //wyczyszczenie całej zawartości zmiennej
rekordowej - wraz z wartością pola będącego kluczem głównym
          LSalesTransaction."Line No.":=LineNo; // ustawienie wartości klucza
głównego, a dalej pozostałych pól:
          LSalesTransaction."Salesperson Code" := LSalesInvoiceHeader."Salesperson
Code";

          CASE LSalesInvoiceLine.Type OF
            LSalesInvoiceLine.Type::"G/L Account" :
              LSalesTransaction.Type := LSalesTransaction.Type::"G/L Account";
            LSalesInvoiceLine.Type::Item :
              LSalesTransaction.Type := LSalesTransaction.Type::Item;
            LSalesInvoiceLine.Type::Resource :
              LSalesTransaction.Type := LSalesTransaction.Type::Resource;
          END;

          LSalesTransaction."No.":=LSalesInvoiceLine."No.";
          LSalesTransaction.Amount:=LSalesInvoiceLine.Amount;
          //TODO: utworzony rekord wstawić do tabeli Sales Transactions
        UNTIL LSalesInvoiceLine.NEXT=0; //pobranie kolejnego wiersza faktury
      END
      ELSE
        ERROR(Text3,LSalesInvoiceHeader."No."); //błąd: faktura bez wierszy -
niespójność danych w tabelach!!!
  END
  UNTIL LSalesInvoiceHeader.NEXT=0; //pobranie kolejnego nagłówka faktury
  MESSAGE(Text1);
END
ELSE
  MESSAGE(Text2);
```

Uwagi:

Funkcja RESET zdejmuje wszystkie filtry ze zmiennej rekordowej, jakie ewentualnie mogły gdzieś wcześniej zostać nałożone na tę zmienną.

Funkcja SETFILTER wyfiltruje tylko te rekordy z tabeli Sales Invoice Header, które mają niepustą wartość pola Salesperson Code (kod sprzedawcy).

Funkcja FINDFIRST pobiera pierwszy rekord z przefiltrowanej przez SETFILTER zmiennej.

Funkcja NEXT pobiera następny rekord.