**PAŃSTWOWA UCZELNIA ZAWODOWA**

**WE WŁOCŁAWKU**

**INSTYTUT NAUK SPOŁECZNYCH I TECHNICZNYCH**

Kierunek: INFORMATYKA

Specjalność: Systemy informatyczne i bazy danych

Mateusz Wojciechowski

Projektowanie baz danych z wykorzystaniem modelu UML

Praca inżynierska

**napisana pod kierunkiem**

*dr inż. Antoniego Zajączkowskiego*

Nr albumu: 10459

##### **Włocławek 2021**

Spis treści

[Wstęp 5](#_Toc72767029)

[Cel pracy 7](#_Toc72767030)

[Problematyka poruszana w pracy 8](#_Toc72767031)

[Rozdział I: Część teoretyczna. 9](#_Toc72767032)

[1.1. Baza danych. 9](#_Toc72767033)

[1.2. Języki baz danych. 11](#_Toc72767034)

[1.3. Rodzaje modeli bazodanowych. 12](#_Toc72767035)

[1.4. Podział baz danych. 15](#_Toc72767036)

[1.5. Język SQL. 16](#_Toc72767037)

[1.6. SQL Server. 17](#_Toc72767038)

[1.7. Język UML. 18](#_Toc72767039)

[1.8. Rodzaje modeli UML. 21](#_Toc72767040)

[1.9. Power Designer. 25](#_Toc72767041)

[1.10. Microsoft Visio. 27](#_Toc72767042)

[1.11. Diagram związków encji. 28](#_Toc72767043)

[1.12. Model fizyczny – Physical Data Model. 30](#_Toc72767044)

[Rozdział II. Projektowanie systemu bazodanowego. 32](#_Toc72767045)

[2.1. Opis modelu konceptualnego 32](#_Toc72767046)

[2.1.1. Opis środowiska funkcjonowania systemu 32](#_Toc72767047)

[2.1.2. Szczegółowe określenie funkcji systemu 33](#_Toc72767048)

[2.1.3. Uwarunkowania prawne systemu 35](#_Toc72767049)

[2.2. Wymagania systemu baz danych 36](#_Toc72767050)

[2.3. Opracowanie modelu konceptualnego 38](#_Toc72767051)

[2.3.1. Model relacyjny danych 39](#_Toc72767052)

[2.3.2. Schemat relacyjnej bazy danych 39](#_Toc72767053)

[2.3.3. Struktura danych 45](#_Toc72767054)

[2.3.4. Uwarunkowania technologiczne 46](#_Toc72767055)

[2.4. Modelowanie danych 48](#_Toc72767056)

[2.4.1. Model przypadków użycia 48](#_Toc72767057)

[2.4.2. Diagram aktywności 51](#_Toc72767058)

[2.4.3. Diagram sekwencji 54](#_Toc72767059)

[2.4.4. Diagram klas UML 58](#_Toc72767060)

[2.4.4.1. Tworzenie diagramu klas 59](#_Toc72767061)

[2.4.4.2. Określanie atrybutów i operacji klasy 61](#_Toc72767062)

[2.4.4.3. Określenie związków pomiędzy klasami 66](#_Toc72767063)

[2.4.4.4. Diagram klas dla systemu wypożyczalni płyt DVD 71](#_Toc72767064)

[2.4.4.5. Generowanie modelu konceptualnego 74](#_Toc72767065)

[2.4.5. Diagram związków encji 78](#_Toc72767066)

[2.4.5.1. Encje 79](#_Toc72767067)

[2.4.5.2. Związki 88](#_Toc72767068)

[2.4.5.3. Model konceptualny dla systemu wypożyczalni płyt DVD 93](#_Toc72767069)

[2.4.5.4. Generowanie modelu fizycznego 96](#_Toc72767070)

[2.4.6. Model fizyczny 97](#_Toc72767071)

[2.4.6.1. Tabele 98](#_Toc72767072)

[2.4.6.2. Referencje 107](#_Toc72767073)

[2.4.6.3. Widoki 113](#_Toc72767074)

[2.4.6.4. Procedury 115](#_Toc72767075)

[2.4.6.5. Wyzwalacze 118](#_Toc72767076)

[2.4.6.6. Użytkownicy i role 121](#_Toc72767077)

[2.4.6.7. Transformacja CDM na PDM 126](#_Toc72767078)

[2.4.6.8. Model fizyczny dla systemu wypożyczalni płyt DVD 129](#_Toc72767079)

[2.5. Implementacja bazy danych 130](#_Toc72767080)

[2.5.1. Tworzenie nowej bazy danych 131](#_Toc72767081)

[2.5.2. Generowanie obiektów w bazie danych 136](#_Toc72767082)

[Rozdział III: Testowanie systemu bazy danych 141](#_Toc72767083)

[3.1. Testowanie systemu 141](#_Toc72767084)

[Podsumowanie 167](#_Toc72767085)

[Bibliografia 170](#_Toc72767086)

[Spis tabel i rysunków 172](#_Toc72767087)

# Wstęp

W systemach informatycznych bazy danych odgrywają bardzo ważną rolę. Utworzona baza danych może decydować o jakości i użyteczności całego systemu informatycznego. Szczególnej uwagi wymaga proces projektowania baz danych, gdzie poszczególne informacje należy odpowiednio zaprojektować. Wybór poszczególnych danych jest bardzo istotny, gdyż od niego zależy prawidłowe działanie bazy danych. Aby ten wybór był odpowiedni, musimy wskazać informacje, które powinny być przechowywane w bazie danych, oraz określić ich strukturę. Istotność projektowania baz danych znajduje swoje odzwierciedlenie w wielu specjalistycznych metodach i narzędziach, które wspomagają jego realizację.

Projektowanie baz danych możemy podzielić na kilka etapów. Rezultatem tego procesu są powstałe zbiory modeli na różnych poziomach abstrakcji. Na samym początku określamy specyfikację wymagań, gdzie ustalamy warunki lub właściwości jakie musi spełnić tworzony system. Uszczegółowienie wymagań jest możliwe dzięki modelowaniu procesów biznesowych za pomocą diagramów UML, takich jak diagramy przypadków użycia i diagramy aktywności. Dzięki użyciu tych diagramów zostanie określony sposób funkcjonowania fragmentu świata rzeczywistego w ramach którego, ma zostać wykorzystywana baza danych. Kolejny etap procesu projektowania obejmuje utworzenie diagramu klas UML, który będzie przedstawiał całą strukturę systemu, stanowiąc podstawę dla jego konstrukcji. Za pomocą utworzonego diagramu klas generowany jest model konceptualny, w którym możemy zidentyfikować obiekty należące do analizowanego fragmentu rzeczywistości. Następnie gotowy model konceptualny jest przekształcany na model fizyczny, który będzie stanowił podstawę implementacji docelowej bazy danych.

Automatyzacja takiego procesu jest możliwa w pewnym stopniu za pomocą narzędzi CASE, czyli komputerowego wspomagania projektowania oprogramowania. Czynności, które mogą być realizowane w sposób automatyczny to transformacja modeli, czyli generowanie nowych modeli na podstawie już istniejących oraz kontrola poprawności i spójności utworzonych części projektu.

W pierwszym rozdziale zostały przedstawione podstawowe pojęcia z zakresu baz danych, języka SQL, UML, diagramu związków encji (konceptualnego) oraz modelu fizycznego. Dodatkowo zostały opisane technologie, które zostały użyte do zaprojektowania systemu baz danych.

W drugim rozdziale zostały w sposób szczegółowy opisane procesy tworzenia systemu baz danych. Opisano wymagania jakie ma spełniać tworzony system oraz określono uwarunkowania prawne, które mają zapewnić ochronę danych przez system. Został także opracowany wstępny model konceptualny, gdzie przedstawiono docelowy schemat bazy danych dla systemu wypożyczalni filmów. W tym modelu zostały przedstawione typy danych oraz encje wraz z odpowiednimi relacjami jakie będą użyte w projektowanej bazie danych. W tym rozdziale opracowano diagramy przypadków użycia dla tworzonego systemu, który będzie definiował interakcje, zachodzące pomiędzy użytkownikami, a systemem. Na podstawie diagramu przypadków użycia możemy przedstawić diagram aktywności, gdzie należy przedstawić kroki, które będą wykonywane przez daną funkcję systemu. Natomiast przy pomocy diagramu sekwencji zostaną zaprezentowane interakcje, jakie będą zachodzić pomiędzy obiektami wraz z uwzględnieniem w czasie komunikatów, przesyłanych pomiędzy nimi. Scharakteryzowano również wszystkie obiekty, które zostaną użyte do zaprojektowania modelu klas UML, konceptualnego i fizycznego. Wyjaśniono także znaczenie symboli, jakie stosuje się na diagramach tworzonych w ramach tych modeli. Pokazano również, jak wykorzystać funkcje programu PowerDesigner do zautomatyzowania czynności projektowych. Dotyczy to szczególnie generowania modelu fizycznego. W sposób szczegółowy została opisana implementacja bazy danych, utworzona na podstawie opracowanych wcześniej modeli. Omówione zostały mechanizmy dostępne w PowerDesignerze, umożliwiające generowanie obiektów bazy danych.

W trzecim rozdziale przedstawiono testowanie zaprojektowanej bazy danych. Omówiono działanie utworzonego systemu pod względem funkcjonalności oraz sprawdzono błędy, jakie mogą wystąpić w systemie.

# Cel pracy

Celem pracy jest zaprojektowanie bazy danych wypożyczalni płyt DVD za pomocą języka UML, gdzie do jego utworzenia wykorzystano środowisko PowerDesigner. PowerDesigner jest narzędziem typu CASE, służącym do projektowania różnych elementów składowych systemów informatycznych, w szczegółowości baz danych. Posiada on funkcje, które umożliwiają modelowanie obiektowe systemów w oparciu o standard UML. Utworzenie bazy danych dla wypożyczalni płyt DVD ma na celu ułatwienie wypożyczanie i oddawanie filmów przez klientów, czy zarządzanie klientami i filmami przez pracowników. Poprawnie utworzona baza danych może zostać użyta m.in. przez developerów do zaprojektowania aplikacji internetowej lub desktopowej, gdzie zostanie dodatkowo wzbogacona o interfejs użytkownika.

# Problematyka poruszana w pracy

Głównym powodem zaprojektowania bazy danych za pomocą języka UML jest ułatwienie projektantom baz danych na lepsze zrozumienie tworzonych systemów. W obecnym świecie tworzenia oprogramowania systemy stają się coraz bardziej złożone. Wiele z tych systemów jest używanych do przetwarzania ogromnej ilości danych przechowywanych w systemach baz danych. W rzeczywistości wiele starszych systemów korzysta z systemu zarządzania relacyjnymi bazami danych (RDBMS). Zaprojektowanie baz danych z użyciem diagramów UML poprawi zrozumiałość tworzonego systemu i ułatwi komunikację między zespołem programistycznym, a zespołem zajmującym się bazą danych. Projektowanie procesów biznesowych jest przetwarzane na poziomie bazy danych, zanim zostaną wykorzystane przez aplikacje. Operacje bazy danych zaimplementowane w procedurach składowanych, wyzwalaczach i funkcjach mogą być bardzo złożone. Zaprojektowanie tych operacji przy pomocy odpowiednich modeli ułatwia zrozumienie możliwości ponownego wykorzystania systemu bazy danych.

# Rozdział I: Część teoretyczna.

## Baza danych.

Baza danych (Pękała, 2015) może być postrzegana jako pewien model fragmentu rzeczywistości, który jest związany z firmą, organizacją lub innym systemem. Baza danych stanowi pewną kolekcję danych, którego zadaniem jest reprezentowanie danego fragmentu rzeczywistości. Zatem możemy stwierdzić, że baza danych jest zbiorem powiązanych ze sobą informacji zorganizowanych w strukturę, która pozwala na łatwe ich przetwarzanie. Baza danych jest zbiorem struktur danych zgromadzonych według określonej systematyki lub metody, indywidualnie dostępnych w jakikolwiek sposób, w tym środkami elektronicznymi, wymagający istotnego co do jakości lub ilości, nakładu inwestycyjnego w celu sporządzenia, weryfikacji lub prezentacji jego zawartości [13].

Według Date’a (Date, 2000) pojęcie bazy danych jest zbiorem danych trwałych, które są wykorzystywane przez system aplikacji danego przedsiębiorstwa. Baza składa się z danych wejściowych i wyjściowych. Dane wejściowe odnoszą się do danych, które wchodzą do systemu po raz pierwszy i mogą one spowodować zmianę trwałych danych, ale na samym początku nie są częścią bazy danych. Natomiast dane wyjściowe są generowane przez system przy pomocy komunikatów oraz wyników. W danych wyjściowych także informacja może wychodzić od trwałych danych i tak samo jak w danych wejściowych informacje nie są uważane za część bazy [5].

Natomiast według autorów Garcii – Moliny, Ullmana i Widom ’a (Garcia-Mollina, Ullman i Widom, 2002) pojęcie to jest zbiorem informacji, które istnieją od dłuższego czasu, być może przez wiele lat. Pojęcie bazy danych odwołuje się do zbioru danych, zarządzanego przez system zarządzania bazą danych (SZBD). System zarządzania bazą danych pozwala między innymi na tworzenie nowych baz danych i schematów przy użyciu języka definicji danych, czy przechowywanie bardzo dużych ilości danych przez długi okres umożliwiając sprawny dostęp do danych [7].

W związku z powyższym można stwierdzić, że baza danych jest pewnego rodzaju sposobem organizacji patrzenia na otaczającą nas rzeczywistość, sposobem kolekcjonowania faktów i wydarzeń związanych z wybranym fragmentem rzeczywistości. Baza danych składa się z danych wejściowych, które są dostarczane do bazy przez użytkownika oraz danych wyjściowych, gdzie dane są generowane przez system i stanowią wynik działania. Baza danych jest zarządzana przez SZBD, który pozwala użytkownikowi na odpowiednie administrowanie bazą oraz przechowywanie dużych ilości danych przez dłuższy czas. Istnieje zatem kilka paradygmatów tworzenia baz danych, zwanych modelami. Modele baz danych, zostaną omówione w dalszej części pracy.

Możemy zauważyć, że baza danych nie jest tworem samoistnym, ale do zapewnienia obsługi bazy danych wykorzystujemy system zarządzania bazą danych (SZBD). Do najpopularniejszych systemów zarządzania bazą danych możemy zaliczyć: Oracle, IBM DB2, Sybase, MS SQL Server, w pakiecie MS Office: ACCESS, a ponadto Postgre SQL lub MySQL. W niniejszej pracy dyplomowej korzystam z SZBD MS SQL Server. Wybrałem właśnie ten serwer bazodanowy, ponieważ w nowoczesny i prosty sposób możemy z poziomu interfejsu graficznego zarządzać bazami danych SQL Server. Jedną z głównych zalet korzystania z MS SQL jest możliwość lepszej integracji z innymi aplikacjami pracującymi w trybie graficznym. Istotnym faktem jest, że w oparciu o bazę MS SQL zarządzanie większymi bazami danych jest łatwiejsze. Dzięki SQL Server praca na różnych wersjach systemu operacyjnego jest bardziej stabilna, oznacza to w praktyce, że ważniejsze jest spełnienie wymagań sprzętowych niż posiadanie określonej wersji systemu Windows.

## Języki baz danych.

Do opisu, tworzenia i zarządzania bazą danych jest potrzebny pewien język. W tym celu został stworzony Strukturalny Język Zapytań (SQL – Structured Query Language). SQL (Pękała, 2015) jest językiem deklaratywnym, który działa w następujący sposób: piszemy zapytanie do systemu zarządzającego bazą danych (SZBD) deklarując co chcemy uzyskać i nie interesuje nas to w jakich krokach odpowiedź zostanie uzyskana. SZBD sam znajduje najlepszy dla siebie sposób wykonania danego zapytania. SQL jest językiem, w którego skład wchodzą następujące moduły, które pozwalają na wykonywanie działań na bazie danych:

1. *DML (ang. Data Manipulation Language – Język Manipulacji Danymi)* – umożliwia wykonywanie określonych operacji na danych przy pomocy języka operowania na danych. Zawiera m.in. polecenia SELECT (wyszukiwanie, czyli podstawę każdego zapytania SQL), UPDATE (aktualizowanie danych), INSERT (wstawianie danych), DELETE (usuwanie danych).
2. *DDL (ang. Data Definition Language – Język Definiowania Danych)* – umożliwia projektowanie i implementację nowej bazy przy użyciu narzędzi i języka definicji danych. Zawiera m.in. polecenia CREATE (tworzenie), DROP (usuwanie), ALTER (zmienianie).
3. *DCL (ang. Data Control Language – Język Kontroli Danych)* – pozwala na zarządzanie bazą danych. Zawiera on m.in. następujące polecenia GRANT (nadaje uprawnienia użytkownikom), REVOKE (odbiera uprawnienia użytkownikom), COMMIT (zatwierdza wprowadzone dane), ROLLBACK (wycofuje wprowadzone dane).
4. *DQL (ang. Data Query Language – Język Zapytań o dane)* – umożliwia selektywny dostęp do danych za pomocą języka zapytań i tworzonych w nim kwerend [13].

## Rodzaje modeli bazodanowych.

Model danych (Benyon-Davies, 2001) z punktu widzenia architektury, jest zbiorem ogólnych zasad użycia danych. Jest zbiorem pewnych reguł, które określają jaka jest jej struktura, jakie możemy wykonać na niej operacje oraz które stany w bazie danych są poprawne. Pod względem architektury model danych opisuje zbiór wymagań dotyczący danych dla danej aplikacji. Czyli pod względem technicznym model bazy danych musi opisywać pewne zasady, na podstawie których jest implementowana jej struktura oraz jakie mogą być rodzaje tych danych (wielkość, ilość, ograniczenia np. numer PESEL może mieć tylko 11 znaków, a numer telefonu posiada maksymalnie 9 znaków), jakie są ich relacje i zależności między nimi. Model danych określa również jakie operacje są dozwolone w bazie danych.

Główne modele baz danych:

* Model hierarchiczny – model ten jest zbudowany z wielu typów rekordów, które są powiązane za pomocą związków nadrzędny – podrzędny. Dane są przechowywane w postaci rekordów, które są ze sobą połączone za pomocą łączy. Rekord jest zbiorem pól, przy czym każde pole będzie zawierało tylko jedną wartość. Każde pole przechowuje dany atrybut i jest przyporządkowane do dowolnego typu danych np. INT. Powiązanie nadrzędny – podrzędny jest związkiem jeden-do-wiele pomiędzy dwoma typami rekordów. Typ rekordu po stronie jeden jest nadrzędnym typem rekordu, a rekord po stronie wiele jest typem podrzędnym rekordu.
* Model sieciowy – opiera się na modelu hierarchicznym, pozwalającym na definiowanie relacji wiele do wielu pomiędzy połączonymi rekordami, co oznacza wiele rekordów nadrzędnych. Model sieciowy wykorzystuje dwa podstawowe elementy: rekordy i zbiory. Rekordy obejmują pola, w których są magazynowane dane. Zbiory określają relację jeden do wielu pomiędzy rekordami, gdzie jeden rekord jest „właścicielem” zbioru zawierającego „członków” zbioru. Jeden rekord może być jednocześnie „właścicielem” jak i „członkiem” wielu zbiorów.
* Model relacyjny – ten model będzie dalej używany w niniejszej pracy. Model relacyjny jest najpopularniejszym modelem, który jest używany przez programistów oraz administratorów baz danych. Model ten jest najlepiej opisany i udokumentowany, jednocześnie najbardziej intuicyjny. Model relacyjny opiera się na matematycznej teorii mnogości, w szczególności na pojęciu relacji. W najnowszym ujęciu w modelu relacyjnym dane są pogrupowane w relacje, które są reprezentowane przez tablice. Relacje są pewnym zbiorem rekordów o identycznej strukturze, powiązane wewnętrznie za pomocą związków zachodzących pomiędzy danymi. Takie podejście w porównaniu z innymi modelami danych ułatwia zapis, redukując popełnianie błędów, ale bardzo cierpi na tym wydajność [1].
* Model obiektowy – opisuje zbiór obiektów, w których zachowanie, stan oraz związki są określone według obiektowego modelu danych. W modelu tym są zachowane wszystkie paradygmaty wzięte z programowania obiektowego, które obejmują: obiekty i tożsamość, obiekty złożone, hermetyzację klasy, dziedziczenie oraz polimorfizm. Stany są reprezentowane przez wartości atrybutów obiektu. Zachowanie jest definiowane za pomocą metod, które działają na stan obiektu po wywołaniu odpowiedniej operacji. Każdy obiekt jest zdefiniowany przez pojedynczy identyfikator obiektu, który istnieje niezależnie od wartości atrybutu obiektu [2].
* Model obiektowo – relacyjny – jest architekturą hybrydową baz danych. Model ten łączy w sobie cechy modelu obiektowego i relacyjnego, w których aplikacje bazodanowe korzystają w szerokim zakresie z wielu funkcji obiektowych, traktując relacyjną bazę danych jako wewnętrzny mechanizm przechowywania danych. Pozwala wykorzystywać dane tak jak na obiektach, gdzie posiada bazę relacyjną jako wewnętrzny mechanizm przechowywania danych. Model obiektowo – relacyjny posiada wiele cech, które umożliwiają efektywną produkcję aplikacji. Wśród nich można wymienić m.in. przystosowanie do multimediów, dane przestrzenne, abstrakcyjne typy danych, metody (funkcje i procedury) definiowane przez użytkownika w różnych językach, typy referencyjne i inne [4].
* Model semistrukturalny – jest modelem danych, gdzie dane są przedstawiane w postaci drzewa. Dane są przedstawiane w sposób elastyczny, jak to jest w przypadku relacyjnych baz danych. Model ten jest przedstawiany w postaci matematycznej w postaci grafu skierowanego. W tym modelu graf jest ukierunkowanym wykresem, w którym każdy węzeł przedstawiony na wykresie, można zakwalifikować do węzła jednostki lub odniesienia. Model semistrukturalny ma bardzo dobre zastosowanie w plikach XML, gdzie dane są przedstawiane w postaci drzewa, pokazującego hierarchiczną strukturę elementów i niejawne relacje między nimi. Model ten jest przydatny także przy integracji źródeł o różnej strukturze, zwłaszcza przy budowie hurtowni wirtualnych [9].

## Podział baz danych.

W zależności od obranego kryterium, bazy danych możemy podzielić na następujące kategorie:

* Według modelu danych:
  + Hierarchiczne,
  + Sieciowe,
  + Relacyjne,
  + Obiektowe,
  + Obiektowo – relacyjne,
  + Semistrukturalne
* Według liczby baz danych:
  + Scentralizowane,
  + Rozproszone,
* Według zastosowań:
  + OLTP (ang. On – Line Transaction Processing) – bazy danych przetwarzania transakcyjnego.
  + OLAP (ang. On – Line Analytical Processing) – bazy danych przetwarzania analitycznego.
  + CAD (ang. Computer Aided Design) – bazy danych wspomagające projektowanie.
  + GIS (ang. Geographical Information Systems) – bazy danych do przechowywania informacji geograficznej.
  + CASE (ang. Computer Aided Software Engineering) – bazy danych do wspomagania wytwarzania oprogramowania [8].

## Język SQL.

SQL (ang. Structured Query Language) (Taylor, 2003) jest strukturalnym językiem zapytań, który służy do tworzenia, modyfikacji oraz zarządzania bazami danych. SQL należy do języków deklaratywnych, co oznacza, że jest oparty na paradygmatach programowania nieopartych na programowaniu imperatywnym. Język SQL opisuje procesy, które są realizowane jako część instrukcji zmieniających status programu.

Język SQL został stworzony przez Donalda Chamberlina i Raymonda Boyce’a w 1974 roku, w firmie IBM. Badacze tej firmy przeprowadzili wczesny rozwój w relacyjnych systemach zarządzania bazami danych, stworzyli podjęzyk danych działających na tych systemach. Początkowa wersja nazywała się SEQUEL (ang. Structured English QUEry Language). Ostatecznie nazwa tego produktu została zamieniona na SQL. Przedsiębiorstwo IBM nie używało opracowanych rozwiązań i wprowadziło je dopiero w momencie, kiedy zaczęła z nich korzystać konkurencja. Zyskując coraz większą popularność SQL otrzymał wsparcie Amerykańskiego Narodowego Instytutu Normalizacji (ANSI) oraz Międzynarodowej Organizacji Normalizacji (ISO). W 1986 roku ANSI wydał formalny standard o nazwie SQL – 86. ANSI zaktualizowało ten standard 3 lata później do SQL – 89 i ponownie w 1992 roku do SQL – 92. Najnowsza wersja standardu SQL to SQL - 2003 (ISO / IEC 9075 – X: 2003). Spowodowało to, że język ten stał się częścią łączącą bardzo różnorodne produkty. Dopiero pojawienie się bardziej zintegrowanych i hermetycznych systemów komputerowych wymusiło rozbicie SQL jako platformy łączącej różne produkty.

Wraz z postępem języka SQL pojawiły się dialekty, które miały podołać wymaganiom danych produktów. Jednym z przykładów jest Transact – SQL (T-SQL), który został stworzony przez firmę Microsoft i zastosowany w ich produkcie SQL Server, który stosowany jest do realizacji pracy. Przedsiębiorstwo Oracle zaprojektowało zaś PL/SQL. Najsłynniejszym silnikiem relacyjnym, który jest używany w serwisach WWW jest MySQL [20].

## SQL Server.

SQL Server (Microsoft SQL Server, 2020) to system zarządzania relacyjnymi bazami danych, który jest wspierany i rozpowszechniany przez firmę Microsoft. Program ten obsługuje szeroki zakres aplikacji, które są związane z przetwarzaniem transakcji, analizy biznesowej i analizy w środowiskach IT. SQL Server jest jednym z trzech dominujących na rynku systemów do zarządzania bazami danych obok firm Oracle i IBM. SQL Server jest powiązany z Transact – SQL (T-SQL), implementacją języka SQL firmy Microsoft, która dodaje zestaw własnych rozszerzeń do standardowego języka programowania.

SQL Server składa się ze struktury tabel, które są oparte na wierszach, łączących elementy danych w różnych tabelach ze sobą, unikając nadmiarowego przechowywania danych w różnych miejscach bazy danych. Model relacyjny zapewnia także integralność referencyjną i inne ograniczenia integralności w celu zachowania dokładności danych. Kontrole te są częścią szerszego przestrzegania zasad atomowości, spójności, izolacji i trwałości, które są znane jako właściwości ACID i mają na celu zagwarantowanie, że transakcje bazodanowe są przetwarzane niezawodnie. Podstawowym składnikiem programu Microsoft SQL Server jest silnik bazy danych, który steruje magazynowaniem, przetwarzaniem i bezpieczeństwem danych. SQL Server zawiera silnik relacyjny, przetwarzający polecenia i zapytania oraz silnik pamięci masowej, zarządzający plikami bazy danych, tabelami, stronami, indeksami, buforami bazy danych oraz transakcjami. Silnik bazy danych wykonuje również procedury przechowywane, wyzwalacze, widoki oraz inne obiekty jakie można utworzyć w bazie danych. Pod silnikiem bazy danych znajduje się system operacyjny programu SQL Server lub SQLOS, który obsługuje funkcje wykonywane na niskim poziomie takie jak, zarządzanie pamięcią We/Wy, planowanie zadań i blokowanie danych, aby uniknąć konfliktu aktualizacji.

Firma Microsoft łączy również różne narzędzia, służące do zarządzania danymi analizy biznesowej BI (ang. Business Inteligence) i analityczne przy użyciu programu SQL Server. Do analizy danych można użyć narzędzia SQL Server Analysis Services, który jest silnikiem analitycznym przetwarzającym dane do wykorzystania w aplikacjach BI i wizualizacji danych. Natomiast narzędzie SQL Server Reporting Services, wspiera tworzenie i dostarczanie raportów BI.

Po stronie zarządzania danych, program Microsoft SQL Server obejmuje usługi SQL Server Integration Services, SQL Server Data Quality Services i SQL Server Master Data Services.

W pakiecie z systemem zarządzania bazą danych są dostarczane również dwa narzędzia dla administratorów baz danych i programistów. Są to programy: SQL Server Data Tools narzędzie służące do tworzenia relacyjnych baz danych i SQL Server Management Studio, które można wykorzystać do wdrażania, monitorowania i zarządzania bazami danych [10].

## Język UML.

UML jest według Fowler’a i Scott’a (Fowler i Scott, 2005) językiem modelowania, w którym dane są przedstawiane według notacji graficznej, gdzie metody są wykorzystywane do przedstawiania modeli. Proces jest wsparciem, które jest udzielane z użyciem metod, dotyczących etapów projektowania i tworzenia oprogramowania. Proces jest procesem iteracyjnym i przyrostowym, gdzie oprogramowanie nie jest dostarczane w całości w wersji ostatecznej, ale jest dostarczane we fragmentach i kolejnych wersjach oprogramowania. Proces składa się z czterech faz tworzenia oprogramowania: inicjacji, rozwinięcia, budowy i wdrożenia [6].

Według głównych twórców tego języka Booch’a, Rumbaugh’a i Jacobson’ a w książce pt. „The Unified Modelling Language User Guide” (Booch, Rumbaugh i Jacobson, 2005) UML jest językiem służącym do wizualizacji, określania, konstruowania i dokumentowania systemów, które w sposób intensywny korzystają z oprogramowania. Stanowi tylko jeden z elementów metodyki tworzenia oprogramowania. Jest on niezależny od procesu wytwórczego, używany przede wszystkim w procesie opartym na przypadkach użycia, zorientowanym na architekturę oraz w modelu iteracyjnym i przyrostowym. Modele UML są tworzone w notacji graficznej, gdzie jego elementom są przypisane odpowiednie symbole, które łączą się z diagramami. Odnosi się on w szczególny sposób do specyfikacji wszystkich ważniejszych decyzji, dotyczących analizy, projektowania i implementacji, które podejmujemy podczas opracowywania i wdrażania systemu intensywnie korzystającego z oprogramowania [3].

Według specyfikacji UML (Object Management Group - specyfikacja UML, 2000), utworzonego przez organizację Object Management Group (OMG), jest on językiem, służącym do określania, wizualizowania, tworzenia i dokumentowania systemów informatycznych, gdzie możemy tworzyć modele biznesowe i inne systemy niebędące oprogramowaniem. Model ten reprezentuje zbiór najlepszych rozwiązań inżynieryjnych, które okazały się skuteczne przy tworzeniu dużych i złożonych systemów [12].

Natomiast w książce pt. „The Unified Modelling Language Reference Manual (Rumbaugh, Jacobson i Booch, 1999), UML jest graficznym językiem modelowania wizualnego ogólnego przeznaczenia, który służy do określenia wizualizacji, konstruowania i dokumentowania systemów informatycznych. Jest on przeznaczony do użytku ze wszystkimi metodami programowania, etapami cyklu życia, domenami aplikacji i nośnikami. Język modelowania ma na celu ujednolicenie dotychczasowych doświadczeń z technikami modelowania i włączenia aktualnych najlepszych praktyk, które dotyczą standardowego podejścia do oprogramowania. UML zawiera pojęcia związane z semantyką, notacją oraz wytycznymi. Ma on części statyczne, dynamiczne, środowiskowe i organizacyjne. Język ten jest obsługiwany przez interaktywne narzędzia do modelowania wizualnego, które posiadają generatory kodów i raportów.

Czyli możemy powiedzieć, że język UML służy do modelowania danych, gdzie przy użyciu różnych technik możemy modelować i modyfikować systemy informatyczne, stosując odpowiednie operacje, ułatwiając ich dostęp do danych. UML jest najczęściej używany do graficznego przedstawiania danych, gdzie elementy są przypisywane za pomocą odpowiednich symboli, które są połączone ze sobą na diagramach. Został on zdefiniowany przez Object Management Group (OMG) w listopadzie 1997 roku, który działał na metamodelu UML – Meta – Object Family (MOF). Język UML ma za cel modelowanie dużych systemów, które muszą być ekspresyjne, aby system mógł spełnić określone funkcjonalności. UML jest również przeznaczony do modelowania systemów, które zostały zaimplementowane za pomocą różnych języków programowania, baz danych, dokumentach organizacyjnych itd. Powstały także nowe metody na bazie tego modelu, wśród których najbardziej znaną metodą jest Rational Unified Process (RUP). Inne metody, które są oparte na UML – u, to m.in. metoda abstrakcji (ang. Abstraction Method) czy metoda rozwoju systemów dynamicznych (ang. Dynamic Systems Development Method). Metody, które zostały zaprojektowane, zapewniały bardziej specyficzne rozwiązania, czy osiąganie innych celów [18].

## Rodzaje modeli UML.

Język UML, definiuje dziewięć rodzajów diagramów, które można podzielić na dwie kategorie:

1. Diagramy strukturalne:
   * Diagramy klas – służą do modelowania statycznego widoku projektu systemu. Obok diagramów przypadków użycia są one wielokrotnie stosowane i jednocześnie są kluczowym rodzajem diagramów w języku UML. Przedstawiają one strukturę systemu w modelach obiektowych poprzez ilustrację struktury klas i zależności między nimi. Diagramy klas przedstawiają typy obiektów w programie, gdzie w odróżnieniu od diagramu obiektów, pokazuje się instancje obiektów i ich zależności istniejące w danym momencie. Są one używane do modelowania statycznych aspektów perspektywy projektowej. Wiąże się z tym silnie modelowanie słownictwa systemu, kooperacji lub schematów. Pozwalają one na sformalizowanie specyfikacji danych i metod.
   * Diagramy komponentów – służą do interpretacji systemu i związków między komponentami. Diagram ten nie reprezentuje koncepcji w postaci klasy lub jej stanu, ale przedstawia elementy ze świata abstrakcyjnego, które są reprezentowane w komputerze. Jeden komponent może być implementacją więcej niż jednej klasy. Interfejs jest pewnego rodzaju zestawem operacji, który określa usługę, która jest dostarczana przez klasę lub komponent lub od nich wymagana. Komponent jest wymienną częścią systemu, który jest zgodny z jej implementacją oraz służy do ponownego wykorzystania przez połączenie z innymi komponentami. Operacja komponentu jest dostępna tylko poprzez interfejs. W przypadku interfejsu klasy, relację pomiędzy interfejsem, a komponentem nazywamy realizacją. Diagram komponentów przedstawia strukturę i potencjalne ścieżki komunikatów komponentu. Sam diagram nie pokazuje kolejności komunikatów.
   * Diagramy wdrożenia – służą do odwzorowania zależności pomiędzy oprogramowaniem lub sprzętem. Diagramy wdrożenia przedstawiają konfigurację węzłów, które są przetwarzane w czasie wykonywania i artefaktów, które na nich istnieją. Stronę sprzętową reprezentują węzły, czyli poszczególne urządzenia obliczeniowe, komunikacyjne i przechowujące, które są powiązane ze ścieżkami komunikacyjnymi. Przy tych diagramach kluczową rolę odgrywają stereotypy, nadawane poszczególnym elementom. Pozwalają one doprecyzować znaczenie i funkcję oprogramowania oraz sprzętu. Diagramy wdrożenia odgrywają istotną rolę przy wdrażaniu systemów wbudowanych, aplikacjach typu klient – serwer oraz w systemach rozproszonych.
   * Diagramy obiektów – przedstawiają obiekty występujące w systemie wraz z ich związkami. Diagramy obiektów modulują wystąpienia elementów zawartych w diagramach klas. Przedstawiają one zestaw obiektów i ich relacje w danym momencie. Służą do tworzenia przykładów, które pomagają zrozumieć diagramy klas, a przede wszystkim powiązań między klasami. Diagramy te skupiają się głównie z obiektów i łączy. Diagramy te skupiają się na obiektach, a nie związkach pomiędzy klasami, gdzie w większości przypadków używa się głównie obiektów i asocjacji.
2. Diagramy behawioralne:

* Diagramy przypadków użycia – są graficznym przedstawieniem przypadków użycia, aktorów oraz związków pomiędzy nimi, które występują w danej dziedzinie programowej. Służą one do modelowania widoku przypadków użycia systemu. Diagramy przypadków użycia wychwytują fragmenty funkcji, które są udostępniane przez system. W większości obejmują modelowanie kontekstu systemu, podsystemu, klasy lub modelowania wymagań dotyczących zachowania tych elementów. Poprzez interakcję aktorów z przypadkami użycia zaprezentowana na diagramach przypadków użycia, z jednej strony aktorzy pełnią rolę wobec systemu. Z drugiej strony przypadki użycia określają usługi świadczone przez system na rzecz aktorów (użytkowników, klientów). Diagramy przypadków użycia dostarczają abstrakcyjnego spojrzenia na system – spojrzenie z pozycji aktorów, którzy go używają. Przypadki użycia można także dziedziczyć, gdzie jeden przypadek użycia dziedziczy zachowanie innego i jednocześnie dodaje lub modyfikuje część tego zachowania.
* Diagramy sekwencji – są rodzajem diagramu interakcji, podkreślającym kolejność komunikatów w czasie. Przedstawiają one zestawy ról oraz komunikaty wysyłane i odbierane przez instancje odgrywające role. Uczestnikami diagramów sekwencji są obiekty, które są opisane nazwą obiektu i jego klasą, wymieniające komunikaty między sobą. Są one zapisane w postaci prostokąta oznaczonym operatorem „sd”. Diagramy sekwencji składają się z pionowych linii życia poszczególnych obiektów uczestniczących w interakcji oraz wymienianych komunikatów między nimi. Czas jest reprezentowany w postaci osi pionowej diagramu. Linia życia obiektu jest czasem, w którym instancja obiektu jest w stanie przyjmować i wysyłać komunikaty. Obiekt jest tworzony poprzez wysłanie do niego komunikatu, a obiekt jest usuwany poprzez oznaczenie na linii życia znaku „X”.
* Diagramy kolaboracji – są rodzajem diagramu interakcji, które podkreślają strukturalną organizację obiektów wysyłających i odbierających wiadomości. Skupiają się na obiektach wchodzące w skład interakcji, w którym są przesyłane komunikaty odbierane przez obiekty. Komunikacje są przedstawiane za pomocą linii, łączących obiekty, przesyłane między obiektami, komunikaty i dane są umieszczane obok tych linii. Komunikaty są opatrzone za pomocą etykiet, które wskazują na kolejność ich wysyłania. Etykiety są postaciami liczbowymi, oddzielone z użyciem kropek. Diagramy kolaboracji prezentują rzeczywiste powiązania obiektów i ich relacje, co może ułatwić zrozumienie interakcji.
* Diagramy stanów – przedstawiają maszyny stanów, które składają się ze stanów, przejść, zdarzeń i czynności. Pokazują one wszystkie możliwe stany obiektu oraz przejścia, powodujące zmianę stanu. W tym diagramie możemy odczytywać sekwencję danych sygnałów wejściowych powodujące przejścia na dany stan oraz jakie decyzje mają zostać podjęte w reakcji na pojawienie się określonych stanów wejściowych. Diagramy stanów mogą być dołączane do klas, przypadków użycia lub całych systemów w celach wizualizacji, określania, konstruowania dynamiki dla pojedynczego obiektu.
* Diagramy aktywności – służą do modelowania czynności i zakresu odpowiedzialności elementów bądź użytkowników systemu. Pokazują one aktywności bez pokazywania bytów realizujących daną aktywność, gdzie są używane jako punkt startowy dla procesów modelowania zachowania. Diagramy aktywności zwykle zawierają działania, węzły aktywności, przepływy oraz wartości obiektów. Akcje reprezentują wykonanie pojedynczej operacji, której nie można robić na mniejszej części w diagramie. Przejście przepływu sterowania, które jest przedstawione w diagramie za pomocą ciągłej strzałki, jest elementem, prezentującym przejście pomiędzy węzłami diagramu czynności. Przejście przepływu obiektów, które oznaczamy przerywaną strzałką, jest elementem, który prezentuje przejście pomiędzy węzłami diagramu widoku interakcji [3].

## Power Designer.

Power Designer (PowerDesigner, 2020) jest narzędziem służącym do tworzenia modeli procesów biznesowych oraz modeli danych koncepcyjnych, logicznych i fizycznych do projektowania baz danych, w tym modeli relacyjnych i wymiarowych. Program może zarządzać modelem procesu biznesowego z projektem bazy danych, zapewniając, że kroki procesu, tworzące dane mają reprezentację danych w modelu logicznym. W Power Designer można utworzyć bazę danych z modelu fizycznego i zbudować różne implementacje fizyczne z jednego modelu logicznego. Dodatkowo w programie można także odtworzyć istniejącą bazę danych w postaci diagramu modelu. Power Designer współpracuje z wieloma systemami zarządzania bazami danych (SZBD). Główne dane wyjściowe z narzędzia obejmują diagramy relacji między jednostkami (ER)[[1]](#footnote-1), raporty analizy wpływu dotyczące zmian w projekcie oraz standardowe lub niestandardowe raporty dotyczące wszystkich obiektów w projekcie (tabele, pola, relacje).

Program Power Designer umożliwia:

* Zarządzanie wymaganiami – analiza wymagań jest powiązana w sposób szczegółowy z wszystkimi modelami w celu śledzenia.
* Generowanie dokumentu – lista jest oparta na kreatorze, zapewniająca użytkownikowi bezpośredni kontrolowany dostęp do metadanych. Sprzyja to lepszej komunikacji pomiędzy wszystkimi członkami zespołu projektowego.
* Analiza wpływu – modele integrują się ze wszystkimi typami modeli w celu pełnej analizy wpływu całego przedsiębiorstwa lub całego projektu. Usprawnia komunikację i współpracę w celu zwiększenia szybkości reakcji całej organizacji na zmiany.
* Mapowanie informacji – edytor mapowania przeciągania i upuszczania umożliwia łatwą, szybką i dokładną dokumentację zależności.
* Otwarte wsparcie – wszystkie główne platformy systemów zarządzania relacyjnymi bazami danych (RDBMS) są obsługiwane razem w jednym narzędziu, czyniąc Power Designer najbardziej wszechstronną i pełną obsługą.
* Możliwość dostosowania – optymalizacja wydajności, kontrolując wszystko, od interfejsu użytkownika do sposobu w jakim modele Power Designer są zarządzane zadaniami, generując jednocześnie kod.
* Zintegrowany ze środowiskami programistycznymi – w sposób płynny można integrować się ze środowiskami programistycznymi, gdzie podłączając dowolne środowisko typu Eclipse, czy Microsoft Visual Studio, możemy bez żadnych problemów synchronizować automatycznie utworzone modele i kod.
* Repozytorium korporacyjne – w pełni zintegrowane repozytorium jest obsługiwane w czasie projektowania przez wiele różnych relacyjnych baz danych. Oferuje takie funkcje takie jak m.in. zabezpieczenia modeli i podmodeli, które są oparte na rolach, kontrola wersji, zarządzanie konfiguracją, czy wszechstronne możliwości wyszukiwania.
* Udostępnianie metadanych – repozytorium Power Designer posiada przeglądarkę, która zapewnia wszystkim użytkownikom pełny dostęp klienta do wszystkich autoryzowanych metadanych, w tym grafiki oraz pełnych definicji i opisów elementów [17].

## Microsoft Visio.

Microsoft Visio (Microsoft Visio, 2020) jest programem należącym do pakietu aplikacji biurowych Microsoft Office, służącym do tworzenia schematów i diagramów. W roku 2000 firma Microsoft, przejęła przedsiębiorstwo Visio, gdzie program był rozwijany. Jest to oprogramowanie typu trialware, gdzie jest dostępny w wersji testowej przez 30 dni, a po wygaśnięciu licencji możemy kupić oprogramowanie dla własnych potrzeb. Visio posiada szereg funkcjonalności takich jak:

* Tworzenie profesjonalnych diagramów – tworzenie schematów blokowych, diagramów czy schematów organizacyjnych w sposób łatwy i intuicyjny. Możemy korzystać z najnowocześniejszych szablonów i kształtów w znanym środowisku pakietu Office.
* Lepsza współpraca – współpraca nad schematami blokowymi w celach uwzględniających wnioski od osób biorących udział w danym projekcie. Korzystając z pakietu Office 365 cały zespół może pracować nad diagramem.
* Wgląd na dane ze świata rzeczywistego – łączenie danych w postaci schematów blokowych i diagramów w czasie rzeczywistym. Za pośrednictwem programu Visio lub pakietu Office 365 możemy automatycznie aktualizować schematy blokowe, aby odzwierciedlić zmiany w podstawowych danych.
* Korzystanie z sieci Web – można pracować nad diagramami, korzystając z przeglądarki internetowej z dowolnego miejsca, w którym się znajdujemy. Dzięki takiemu rozwiązaniu można bez żadnego problemu rysować schematy blokowe, mapować sieci IT, tworzyć wykresy organizacyjne czy dokumentować procesy biznesowe [11].

## Diagram związków encji.

Diagram związków encji (ang. Entity – Relationship Diagram) (Benyon-Davies, 2001) jest rodzajem graficznego przedstawienia związków między encjami, stosowany w projektowaniu systemów informatycznych do prezentacji konceptualnych modeli danych używanych w systemie. W tym diagramie obszar analizy jest reprezentowany za pomocą trzech elementów: encji, związków i atrybutów. Encję można zdefiniować jako rzecz, która jest rozpoznawana jako rzecz, istnieje niezależnie i jednoznacznie jest identyfikowana. Encja jest abstrakcją złożoności pewnej dziedziny, gdzie przedstawia się pewien aspekt świata rzeczywistego, odróżniający go od innych aspektów świata rzeczywistego. Encje służą do logicznego grupowania danych. Związki są powiązaniem między encjami, gdzie w diagramie ERD między dwiema encjami może istnieć więcej niż jeden związek. Metoda diagramów ERD ma na celu potwierdzenie tylko bezpośrednich związków. Encja jako aspekt świata rzeczywistego charakteryzuje się pewną liczbą właściwości lub atrybutów. Wartościom przypisanych atrybutów, odróżniamy jedną encję od innych. Dane, które są używane do realizacji danego systemu informatycznego nie zawsze należą do jednej z trzech kategorii: encji, związku lub atrybutu. Osoba tworząca model danych musi podjąć decyzję, który punkt widzenia jest najważniejszy dla systemu informacyjnego. Taką analizę danych nazywamy często modelem semantycznym, gdzie dane są przedstawione w taki sposób, w jaki są one widziane dla danej organizacji. Końcowym efektem diagramów związków encji jest graficzny model encji i związków dla danej dziedziny rozważań. Na diagramie encja jest reprezentowana przez prostokątną ramkę, gdzie wewnątrz tej ramki, jest wpisana nazwa encji. Nazwy encji są oznaczane rzeczownikami w liczbie pojedynczej. Istnieje tylko jedna instancja kategorii. Związki między encjami są reprezentowane za pomocą linii między encjami i są reprezentowane za pomocą linii między danymi ramkami. W niektórych notacjach etykiety są umieszczane na liniach związków, pomagając w wyjaśnianiu niejednoznaczności. Atrybut jest reprezentowany przez kółko lub owal, który jest połączony z daną encją za pomocą linii. Symbole są oznaczane z użyciem kółek, ponieważ dane elementy danych w diagramach zależności są stosowane w atrybutach diagramów encji.

Diagram związków encji (ERD) pokazuje logiczne związki między różnymi encjami, przy czym istnieją dwie zbiory reguł mające następujące cechy:

1. Uczestnictwo (lub opcjonalność) która mówi o tym, czy każda encja musi, czy może wystąpić równocześnie z inną. Encja może wystąpić w związku, jeżeli wszystkie instancje encji muszą brać udział w związku. Możemy zakładać, że uczestnictwo jest wymagane oraz jest domyślnym uczestnikiem. Jeżeli uczestnictwo jest opcjonalne, to dodajemy symbol „O” przy specyfikowanej encji.
2. Liczebność (inaczej krotność, stopień) – określa, ile encji, może wejść w skład związku:
   * 1:1 („jeden do jednego”) – encji odpowiada dokładnie jedna encja,
   * 1:N („jeden do wielu”) – encji odpowiada jedna lub więcej encji,
   * M:N („wiele do wielu”) – jednej lub więcej encjom odpowiada jedna lub więcej encji.

W przypadku związków M:N często stosuje się normalizację diagramu, która polega na dodaniu encji pośredniczącej i zamianę związku M:N dwoma związkami 1:N z nową encją [1].

## Model fizyczny – Physical Data Model.

Fizyczny model danych (ang. Physical Data Model) (West, 2011) jest strukturą bazy danych, tabel i kolumn lub wiadomości, które są wysyłane pomiędzy procesorami komputerów. Tabele są reprezentowane przez typy jednostek, a linie typów relacji są reprezentowane przez klucze kandydujące i obce między tabelami. Klucze kandydujące są minimalnym zestawem atrybutów relacji, które dokładnie identyfikują każdy rekord tej relacji. W danej relacji może się znajdować wiele kluczy kandydujących, spośród których wybiera się głównie jeden klucz główny. Struktura tabeli jest często dostosowana do konkretnych potrzeb procesów działających na danych w celu zapewnienia jak najlepszej wydajności.

W skład struktury fizycznego modelu danych wchodzą:

* Ograniczenia danych, które mogą być przechowywane,
* Denormalizacja w celu poprawy wydajności określonych zapytań,
* Referencyjne reguły integralności w celu implementacji typów relacji,
* Reguły i dane pochodne, które są istotne dla procesów aplikacji, obsługującej model fizyczny danych [21].

Implementacja fizycznego modelu danych wymaga zrozumienia cech i ograniczeń wydajności wykorzystywanego systemu baz danych. Najczęściej są to kolumny relacyjnych baz danych, typy danych, a relacje między tabelami są implementowane w konkretnym produkcie relacyjnych baz danych.

Aby zaprojektować fizyczny model danych, wymaga to dogłębnej wiedzy na temat konkretnego SZBD, który jest stosowany w celu:

* Reprezentowania logicznego modelu danych w schemacie bazy danych,
* Dodawania encji i definicji atrybutów, które są potrzebne do spełnienia wymagań operacyjnych,
* Konfigurowania bazy danych i dostosowanie jej do wymagań wydajnościowych.

Cechy modelu fizycznego obejmują:

* Definicje, które są specyficzne dla SZBD,
* Definicje tabeli, kolumny i innych fizycznych obiektów w SZBD, reprezentujące byty i atrybuty w modelu logicznym. Atrybuty kolumn są definiowane i wdrażane przez różne osoby dla poszczególnych SZBD,
* Reguły integralności referencyjnej ustanawiają relację między tabelami i kolumnami. Obejmują one klucze kandydujące i obce, ograniczenia i wyzwalacze, różniące się w poszczególnych bazach danych,
* Jednostki wydajności i optymalizacji oparte na określonych funkcjach SZBD, takich jak indeksy, wyzwalacze, procedury, obszary tabel, partycje i zmaterializowane widoki [19].

Jest możliwość przekształcenie utworzonego modelu danych na model fizyczny danych (PDM), który można wykreować na podstawie istniejącej bazy danych, za pomocą samego skryptu SQL.

# Rozdział II. Projektowanie systemu bazodanowego.

## 2.1. Opis modelu konceptualnego

Aby zaprojektować bazę danych w jak najlepszy sposób, musimy się zastanowić do czego będzie służyć, jakie mają być uwarunkowania fizyczne i gospodarcze danego podmiotu, dla którego ma zostać utworzona. Każdy tworzony system informatyczny jest komputerową realizacją fragmentu świata rzeczywistego. Musimy zbudować taki fragment tego świata rzeczywistego, który łatwo i sensownie można odzwierciedlić w systemie informatycznym. Wykorzystując model konceptualny możemy zauważyć w jak działają diagramy ERD i UML. Diagramy tego typu, powinny obejmować najważniejsze z punktu widzenia implementacji systemu, zbiory encji lub klasy, które są niezbędne do zaimplementowania tworzonego systemu.

### 2.1.1. Opis środowiska funkcjonowania systemu

W tym podrozdziale przedstawię opis środowiska dla tworzonej bazy danych, dla zarządzania wypożyczalnią płyt DVD. Programista, który ma utworzyć taki system nie jest do końca zorientowany w specyfice danego przedsiębiorstwa, dla którego tworzy dany system informatyczny. Osoba zamawiająca dany system nie posiada wystarczającej wiedzy na temat tworzenia systemu, ale również nie posiada orientacji w kwestii projektowania takiej bazy danych. Właściwe zrozumienie środowiska informatycznego i wszelkich aspektów z nim związanych m.in. uwarunkowań prawnych, kwestii technologicznych czy relacji właścicielskich są niezbędne do podejmowania trafnych i przemyślanych decyzji do realizowania zamierzonych celów. System bazy danych posiada następujące funkcjonalności:

* Baza danych została utworzona na potrzeby wypożyczalni płyt DVD. Służy do gromadzenia najważniejszych danych, dotyczących filmów (produkcja filmowa, gatunki filmowe, role aktorów w filmach, dostępność filmów oraz cena) oraz klientów (imię, nazwisko, telefon, miejsce zamieszkania, adres e – mail oraz data wpisu do bazy danych).
* Poza wypożyczaniem oraz oddawaniem danego filmu, system umożliwia przeglądanie bazy filmów, które będą uporządkowane według wszystkich możliwych kryteriów.
* System pozwala na wyświetlenie wszystkich istniejących kont, jakie są aktywne w wypożyczalni.
* System pozwala na dodawanie, edycję lub usuwanie danych, dotyczących klientów, pracowników oraz filmów.
* Płatność obliczana jest tylko w przypadku, gdy klient odda film do wypożyczalni. Po sprawdzeniu danych klienta oraz tytułu filmu, system wyliczy opłatę jaką musi ponieść klient za każdy wypożyczony film.

### 2.1.2. Szczegółowe określenie funkcji systemu

W zależności od miejsca zastosowania systemu oraz jego rodzaju zmieniają się również zakres i opis jego funkcjonalności. System baz danych dla wypożyczalni płyt DVD będzie systemem do zarządzania bazą filmów oraz posiadający funkcje zapisu następujących danych:

* Filmy, które będą dostępne do wypożyczenia lub będą wypożyczone przez konkretnych klientów,
* Klienci, którzy mają lub nie mają w danej chwili wypożyczonego filmu czy filmów.

Funkcje systemów informatycznych są ściśle powiązane z operacjami, które są wykonywane na informacjach podawanych procesowi cyfryzacji. Do podstawowych funkcji systemu dotyczących danych zaliczamy m.in. gromadzenie, przechowywanie czy przetwarzanie danych. Funkcje, jakie zostały przewidziane w celu spełnienia wymagań systemu baz danych dla wypożyczalni płyt są następujące:

* Magazynowanie filmów – w bazie danych będą się znajdowały filmy różnego gatunku oraz różnych producentów wraz z wieloma rolami, które będą grane przez wielu artystów.
* Magazynowanie klientów – w bazie będą się znajdowali klienci, którzy mogą mieć wypożyczone kilka filmów jednocześnie lub nie mają żadnego wypożyczonego filmu. Każdy klient w bazie musi podać swoje dane osobowe oraz adres zamieszkania w celu poprawnego działania systemu.
* Wypożyczanie filmów – każdy klient zapisany w bazie może wypożyczyć film (lub filmy), który będzie dostępny w magazynie.
* Oddawanie filmów – każdy klient, który wypożyczył film lub filmy, musi je oddać.
* Magazynowanie filmów wypożyczonych – w bazie danych znajdują się filmy, które zostały wypożyczone przez klientów. Każdy film wypożyczony zawiera informacje o kliencie, który wypożyczył dany film, datę wypożyczenia i oddania filmu oraz należność, jaką klient musi uregulować.
* Blokowanie wypożyczenia filmów, których nie ma w magazynie – system anuluje każdą próbę wypożyczenia filmu, który aktualnie, nie jest dostępny w magazynie.
* Wyszukiwanie filmów oraz klientów według różnych kryteriów – filmy oraz klientów możemy wyszukać w bazie danych według wszystkich możliwych kryteriów.

### 2.1.3. Uwarunkowania prawne systemu

W współczesnym świecie troska przedsiębiorstw o bezpieczeństwo systemów informatycznych nie jest dobrowolną praktyką, lecz koniecznością i obowiązkiem. Obowiązek ten nakładają na nie m.in. odpowiednie przepisy prawne, które obligują przedsiębiorstwa do zapewnienia ochrony zgromadzonym w systemach danym oraz przewidują odpowiedzialność karną za ich nieprzestrzeganie. Do podstawowych aktów prawnych, mających wpływ na bezpieczeństwo i ochronę danych w systemach informatycznych polskich przedsiębiorstw należą ustawy: Kodeks karny [k.k. 1997], o rachunkowości [1994], o ochronie danych osobowych [ustawa ODO 1997]. Gdy weźmiemy pod uwagę uwarunkowania prawne, tworzony system baz danych będzie zapewniał:

* Bezpieczeństwo i ochronę danych osobowych – jednym z ważniejszych aspektów prawnych będzie zapewnienie w naszym systemie bezpieczeństwa danych osobowych, które będą zawarte w bazie danych.
* Wgląd użytkowników do własnych danych osobowych – zgodnie z ustawą o ochronie danych osobowych, osoby których dane osobowe są zawarte w bazie danych, będą miały prawo do odczytu swoich danych osobowych.
* Możliwość zmiany danych osobowych wyłącznie przez upoważnione osoby – zgodnie z ustawą o ochronie danych osobowych, osoby których dane są zawarte w bazie danych, mają prawo do edycji swoich danych osobowych, których może dokonać tylko osoba obsługująca system.
* Wykorzystywanie danych osobowych – zebrane dane osobowe są wykorzystywane wyłącznie do prawidłowej pracy systemu baz danych i nie będą rozpowszechniane na zewnątrz innym instytucjom prywatnym i państwowym.

## 2.2. Wymagania systemu baz danych

Podczas projektowania i wdrażania systemu informatycznego jednym z najistotniejszych aspektów jest szczegółowa specyfikacja wymagań funkcjonalnych i niefunkcjonalnych, jakie ma spełniać. Wymaganiami funkcjonalnymi nazywamy konkretne aspekty funkcjonalności aplikacji, które definiują działania, które mogą być udostępniane dla użytkownika czy samego oprogramowania. Specyfikacja tych wymagań jest jednym z kluczowych punktów inicjonowania projektu, ponieważ dobrze zdefiniowane i opisane wymagania mogą w znaczny sposób uprościć implementację oraz analizę wprowadzanych modyfikacji. System do zarządzania wypożyczalnią płyt DVD, posiada następujące funkcjonalności, takie jak:

* Klient może przeglądać filmy, które są dostępne do wypożyczenia,
* Pracownik ma prawo wyświetlić wszystkich klientów, którzy są członkami wypożyczalni,
* Pracownik ma prawo dodać film do bazy danych, podając jego unikalną nazwę oraz jego atrybuty,
* Pracownik może edytować film, w przypadku podania błędnych danych, podając identyfikator filmu oraz wszystkie jego atrybuty,
* Pracownik ma prawo usunąć film z bazy danych, tylko w przypadku, gdy film nie jest wypożyczany przez klientów. Do usunięcia filmu z bazy danych, pracownik podaje identyfikator filmu, który został wybrany do usunięcia,
* Klient ma obowiązek dodać swoje konto, podając jego unikalną nazwę oraz atrybuty takie jak m.in. Imię, Nazwisko czy Adres zamieszkania,
* Klient ma prawo do edycji swoich danych osobowych, podając identyfikator klienta oraz swoje najważniejsze dane,
* Klient ma prawo do usunięcia swojego konta, tylko w przypadku zrezygnowania z usług wypożyczalni,
* Klient może wypożyczyć filmy, które będą dostępne w magazynie. Płatność za wypożyczony film jest osobna dla każdego z filmów. Po zapłaceniu za film, jest on aktywny przez 7 dni,
* Klient ma obowiązek oddać filmy, które zostały przez niego wypożyczone. Jeżeli się okaże, że klient zwleka z oddaniem filmów, to po upływie 7 dni od daty wypożyczenia klient będzie zmuszony do poniesienia dodatkowej opłaty za nieoddany film,
* Pracownik może zablokować film wybrany przez klienta, gdzie film nie jest aktualnie dostępny w magazynie. W takiej sytuacji system będzie za każdym razem anulował każdą próbę wypożyczenia filmu.

Każda funkcjonalność powinna być skonkretyzowana i opisana w sposób umożliwiający jej implementację podczas kodowania. Może się to okazać przydatnym rozwiązaniem przy późniejszej analizie gotowego systemu.

Do wymagań niefunkcjonalnych zaliczamy wszystkie wymagania, które nie dotyczą bezpośrednio konkretnych funkcji systemu, które mogą być związane z właściwościami systemu takich jak czas reakcji, niezawodność i zajętość pamięci. Mogą one definiować ograniczenia systemu takie jak możliwości urządzeń wejścia – wyjścia i reprezentacje danych używane przez interfejsy. Zwykle dotyczą systemu jako całości, a nie poszczególnych cech systemu tak jak wymagania funkcjonalne. Wymagania mogą dotyczyć nie utworzonego systemu, lecz mogą ograniczać proces tworzenia systemu np. specyfikacja standardów, które należy użyć w procesie, metodyka, narzędzia, itd. Wymaganiami niefunkcjonalnymi dla systemu wypożyczalni płyt DVD będą następujące funkcje:

* System będzie umożliwiał utrzymanie danych w spójności, ponieważ nie można dopuścić do wystąpienia anomalii w bazie danych,
* System powinien posiadać jak najkrótszy dostęp do danych, gdzie wyższym priorytetem jest spójność danych,
* System ma umożliwiać pracę w oparciu o bazę danych Microsoft SQL Server,
* Serwery bazodanowe mogą używać tego systemu w środowisku maszyn wirtualnych Oracle Visual Box czy VMWare,
* System powinien być w stanie obsłużyć dziesięć tysięcy użytkowników,
* System powinien wykonywać dla każdego elementu systemu kopie bezpieczeństwa,
* Użytkownik tego systemu powinien zapewnić migrację danych z obecnych systemów,
* System powinien kontrolować integralność danych,
* System powinien umożliwiać zabezpieczenie danych przed nieautoryzowanym dostępem.

## 2.3. Opracowanie modelu konceptualnego

Prawidłowo zaprojektowana baza danych zapewnia dostęp do aktualnych i dokładnych informacji. Dane przechowywane w bazie mogą być dokładne tylko wtedy, gdy każda informacja jest zapisana dokładnie jeden raz. W przypadku edycji jednej informacji wystarczy modyfikacja jednego wpisu z bazy. Projekt wykonany prawidłowo jest niezbędny do osiągnięcia założonych celów podczas pracy z bazą danych, dlatego czas, który potrzeba na naukę zasad dobrego projektowania ma sens. Jeżeli dane zostaną zmagazynowane lub wyświetlone razem w naszym systemie, jego ważną częścią jest struktura danych. Musimy się zastanowić nad tym, w jaki sposób dane będą wykorzystywane w jednym konkretnym systemie, ale w jaki sposób będą korzystali inni z utworzonego systemu wypożyczalni płyt. Odwołanie się do utworzonych procesów biznesowych oraz różnego rodzaju schematów pomoże nam określić jakie dane chcemy przechowywać w bazie danych.

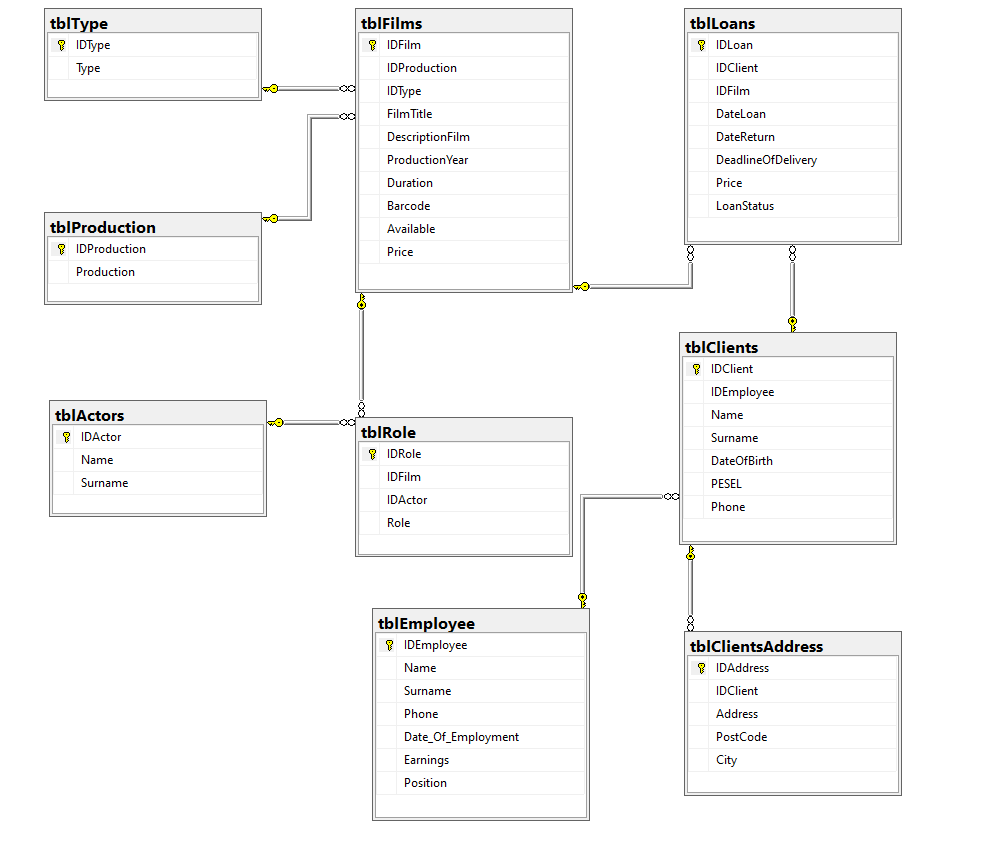
### 2.3.1. Model relacyjny danych

Jak mogliśmy się dowiedzieć z rozdziału pierwszego model relacyjny pozwala na przechowanie powiązanych ze sobą elementów danych i zapewnienie do nich dostępu. Na modelu relacyjnym są oparte relacyjne bazy danych, gdzie jest to prosty i intuicyjny sposób przedstawiania danych w tabelach. W relacyjnej bazie danych każdy wiersz tabeli jest rekordem z unikatowym identyfikatorem nazywanym kluczem. Kolumny tabeli zawierają atrybuty danych, a każdy rekord zawiera zwykle wartość dla każdego atrybutu, co ułatwia ustalenie relacji między poszczególnymi elementami rekordu. Czyli możemy powiedzieć, że relacyjny model danych składa się z trzech podstawowych elementów:

* relacyjna struktura danych,
* operatory relacyjne umożliwiające tworzenie, przeszukiwanie i modyfikację informacji zawartych w bazie danych,
* więzy integralności, które określają wartości danych.

### 2.3.2. Schemat relacyjnej bazy danych

Schemat bazy danych określa zbiór pojęć używanych do opisywania własności konkretnego wycinka świata rzeczywistego, które są istotne z punktu widzenia danego zastosowania. Baza danych to model logicznie spójny, służący określonemu celowi. W związku z powyższym baza danych nie powinna przyjąć, takiego stanu, który nigdy nie jest osiągany w modelowanej rzeczywistości. Na Rysunku 2.1, został przedstawiony diagram związków encji, który przedstawia konceptualny model danych relacyjnej bazy danych dla wypożyczalni płyt DVD.



Rysunek 2.1. Diagram związków encji przedstawiający relacyjną bazę danych dla potrzeb wypożyczalni płyt DVD.

Źródło: opracowanie własne

W poniższej tabeli 1.1 przedstawiono typy danych występujące w relacyjnej bazie danych. Dane które zostaną użyte w systemie wypożyczalni płyt DVD utworzono za pomocą języka SQL w środowisku Microsoft SQL Server.

Tabela .1. Lista typów danych użytych w systemie wypożyczalni płyt DVD.

Źródło: opracowanie własne

|  |  |  |
| --- | --- | --- |
| **Typ danych** | **Zakres wartości** | **Wielkość** |
| INT | – 231 (– 2 147 483 648) do 231 (2 147 483 647) | 4 bajty |
| VARCHAR | Dane tekstowe o zmiennej długości, maksymalnie 8000 znaków | 1 bajt na znak |
| CHAR | Dane tekstowe o stałej długości, maksymalnie 8000 znaków | 1 bajt na znak |
| MONEY | Od – 263 do 263 – 1 z dokładnością jednej dziesięciotysięcznej jednostki | 8 bajtów |
| DATE | Od 1 stycznia 0001 do 31 grudnia 9999 | 3 bajty |

Poniżej została przedstawiona lista typów danych, które występują w bazie danych wraz z określeniem ich użycia:

* **INT** – krotki kolumn identyfikujące wszystkie relacje w bazie danych, oraz wartości określające rok produkcji i dostępność filmu.
* **VARCHAR** – opisujący większość atrybutów encji, takich jak tytuł filmu, imię i nazwisko klienta oraz opisy filmów, które różnią się od pozostałych atrybutów znacznie większą liczbą znaków.
* **CHAR** – opisuje atrybuty encji, posiadające stałą długość znaków takich jak numer telefonu czy kod pocztowy.
* **MONEY** – służący do przechowywania ceny za każdy film oraz zarobków dla poszczególnych pracowników wypożyczalni.
* **DATE** – automatycznie wykorzystywany w chwili wpisywania nowych klientów do bazy, wykonywania procedur związanych z opłatą za wypożyczony film oraz obliczania terminu oddawania filmu.

Jak możemy zauważyć na Rysunku 2.1 zostało zaprojektowanych osiem encji o następujących nazwach: *tblType*, *tblProduction*, *tblActors*, *tblFilms*, *tblRole*, *tblEmployee*, *tblLoans*, *tblClients* i *tblClientsAddress*. Encje te zawierają wszystkie potrzebne informacje na temat filmów, klientów, pracowników, filmów aktualnie wypożyczonych i stanów filmów w danym czasie, czyli atrybuty wraz z ich typami danych oraz klucze główne. W tym modelu mamy styczność z wieloma związkami (relacjami) między encjami i są to:

1. Produkcja filmu:

*Opis:* przynależność produkcji do filmu;

Liczebność: jeden do wielu (1:N);

Jedna produkcja musi być przypisana do wielu filmów, natomiast jeden film musi być przypisany tylko do jednej produkcji;

*Uczestnictwo:* *tblFilms* – wymagane; *tblProduction* – wymagane;

Każdy film musi być przypisany do producenta oraz każdy producent musi być przypisany do filmu.

1. Gatunek filmu:

*Opis:* przynależność gatunku do filmu;

*Liczebność:* jeden do wielu (1:N);

Jeden gatunek musi być przypisany do wielu filmów, natomiast jeden film musi być przypisany tyko do jednego gatunku;

*Uczestnictwo:* *tblFilms* – wymagane; *tblType* – wymagane;

Każdy film musi być przypisany do gatunku oraz każdy gatunek musi być przypisany do filmu;

1. Wypożyczony film:

*Opis:* wypożyczenie filmu;

*Liczebność:* jeden do wielu (1:N);

Jeden film może być wypożyczony tyle razy jednocześnie, ile jest dostępnego filmu na stanie, natomiast jeden film wypożyczony musi być przypisany tylko do jednego filmu.

*Uczestnictwo:* *tblFilms* – wymagane; tblLoans – opcjonalne;

Każdy wypożyczony film musi być przypisany do filmu, ale nie każdy film musi być w danym czasie wypożyczony.

1. Klient wypożyczył:

*Opis:* klient wypożycza film;

*Liczebność:* jeden do wielu (1:N);

Jeden klient może wypożyczyć wiele filmów, natomiast jeden wypożyczony film może być przypisany tylko do jednego klienta.

*Uczestnictwo:* *tblClients* – opcjonalne; *tblLoans* – wymagane;

Każdy wypożyczony film może być przypisany tylko do jednego klienta, ale jeden klient może mieć wypożyczonych kilka filmów lub wcale.

1. Rola aktora filmowego:

*Opis:* przypisanie aktora do wykonywanej roli;

*Liczebność:* jeden do wielu (1:N);

Jeden aktor może być przypisany do wielu ról, natomiast jedna rola musi zostać przypisana tylko do jednego aktora;

*Uczestnictwo:* *tblActors* – wymagane; *tblRole* – opcjonalne;

Każda rola w filmie musi zostać przypisana do aktora oraz każdy aktor może zostać przypisany do roli w filmie.

1. Rola aktora w filmie:

*Opis:* aktor gra różne role w filmie;

*Liczebność:* jeden do wielu (1:N);

Jeden film może być przypisany do wielu ról granych przez aktorów, natomiast jedna rola musi być przypisana tylko do jednego filmu;

*Uczestnictwo:* *tblFilms* – wymagane; *tblRole* – opcjonalne;

Każda rola w filmie musi zostać przypisana do filmu oraz każdy film może zostać przypisany do roli, grającego przez aktora;

1. Obsługa klienta przez Pracownika:

*Opis:* pracownik obsługuje klienta w wypożyczalni;

*Liczebność:* jeden do wielu (1:N);

Jeden pracownik musi obsłużyć wielu klientów w wypożyczalni, natomiast jeden klient, może zostać obsłużony tylko przez jednego pracownika;

*Uczestnictwo:* *tblEmployee* – opcjonalne; *tblClients* – wymagane;

Każdy klient świadczący usługi w wypożyczalni może zostać obsłużony przez pracownika oraz pracownik musi obsłużyć jednego klienta lub wcale.

1. Adres Klienta:

*Opis:* klient podaje dane adresowe pracownikowi wypożyczalni;

*Liczebność:* jeden do jednego (1:1);

Jeden klient może posiadać tylko jeden adres, natomiast jeden adres, musi zostać przypisany tylko do jednego klienta.

*Uczestnictwo:* *tblClients* – wymagane; *tblClientsAddress* – opcjonalne;

Każdy klient może zostać przypisany do swoich danych adresowych oraz każdy adres musi być przypisany do klienta.

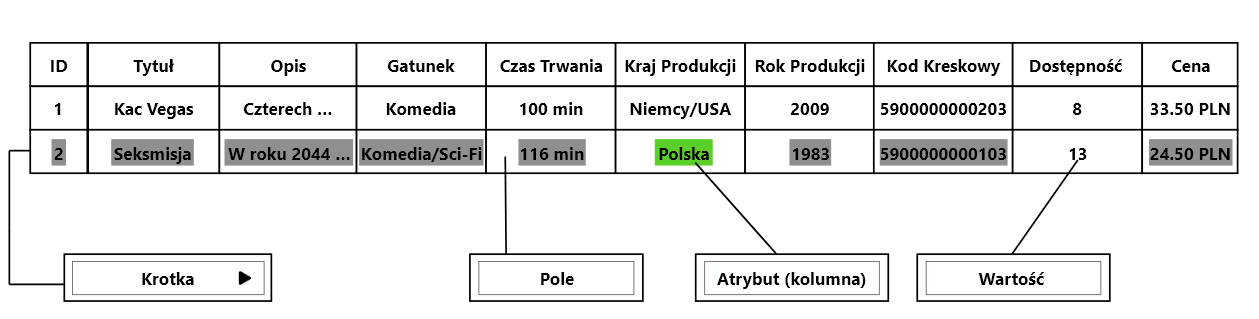
### 2.3.3. Struktura danych

Podstawową strukturą danych w systemie jest relacja będąca podzbiorem iloczynu kartezjańskiego dwóch zbiorów przedstawiających dopuszczalne wartości. W bazie danych relacja przedstawiana jest w postaci tabeli jako zbiór krotek, które posiadają taką samą strukturę, ale różne wartości. Każda krotka odpowiada jednemu wierszowi tablicy oraz posiada co najmniej jeden atrybut odpowiadający pojedynczej kolumnie tablicy. Każdą relację (tablicę) określa kilka następujące wartości:

* krotki (wiersze) są unikalne,
* atrybuty (wiersze) są unikalne,
* kolejność krotek (wierszy) nie ma znaczenia,
* kolejność atrybutów (kolumn) nie ma znaczenia,
* wartości atrybutów (pól) są atomowe.

Poprzez strukturę danych rozumiemy także związki między relacjami oraz ograniczenia.

Żeby lepiej zrozumieć w jaki sposób jest zbudowana struktura danych na Rysunku 2.2 została przedstawiona przykładowa tabela zawierająca dane filmów, jaka znajduje się w bazie danych dla potrzeb wypożyczalni DVD. Na poszczególnych elementach tabeli, zostały pokazane dokładne położenia atrybutów.



Rysunek 2.2. Przykładowa tabela zawierająca dane filmów wraz z atrybutami znajdująca się w bazie danych.

Źródło: opracowanie własne

### 2.3.4. Uwarunkowania technologiczne

Z punktu widzenia uwarunkowania technologicznego baza danych dla wypożyczalni płyt DVD jest przewidziana do adaptacji w systemie bazodanowym Microsoft SQL Server. Jest to relacyjna baza danych, która poprzez relacje encji umożliwia nam pozyskiwanie i tworzenie spójnych danych. Bazę tworzą encje, odpowiednio będące ze sobą w relacji. Przykładowymi encjami w systemie będą: filmy, klienci, pracownicy, gatunki filmów, produkcja filmów, role oraz aktorzy. Każda tych encji będzie posiadała wszystkie wymagane atrybuty oraz klucze główne, kandydujące i obce.

W bazie danych zaimplementowanych zostało wiele funkcji, m.in. takich jak:

* + 1. Ograniczenia:
* NOT NULL – oznaczające, że krotka nie może być pusta;
* DEFAULT – pozwalająca definiować wartość domyślną;
* CHECK – który wymusza wprowadzenie poprawnych danych w odpowiednim formacie;
  + 1. Perspektywy (widoki):
* VIEW – czyli wirtualne tabele wywiedzione z innych tabel;
  + 1. Indeksy:
* INDEX – dzięki któremu dane nie są wyszukiwane kolejno, tylko po indeksach;
  + 1. Procedury przechowywane:
* PROCEDURE – pozwalające na częste wykonywanie zaawansowanych zapytań bez obciążania serwera;
  + 1. Nadawanie praw użytkownikom:
* GRANT, DENY, REVOKE – pozwalające na określenie praw użytkownikom do danych oraz do operacji wykonywanych na tych danych;
  + 1. Zaawansowane wyszukiwanie danych:
* SELECT, ORDER BY, JOIN, LIKE, HAVING, WHERE – umożliwiające wyszukiwanie według różnych kryteriów;
  + 1. Funkcje agregujące:
* AVG, COUNT, SUM, MIN, MAX – obliczające średnią arytmetyczną, zliczającą, sumującą wartości minimalne i maksymalne;
  + 1. Replikacja danych:
* Nie możemy dopuścić do utraty danych, więc baza każdej nocy, po dniu, w którym była modyfikowana, zostanie automatycznie replikowana.

## 2.4. Modelowanie danych

Modelowanie danych jest procesem szczególnej analizy danych, która ma na celu stworzenie modelu systemu informacyjnego, który pozwala operować informacjami dotyczącymi określonego celu projektu fragmentu rzeczywistości, ale niejako z góry, a nie od podstaw, jak w przypadku normalizacji. Normalizacja zaczyna się zebraniem danych, które mają być przechowywane w bazie danych, podczas kiedy modelowanie danych polega na określeniu tematów mających znaczenie dla projektu oraz charakterystycznych dla nich cech. W efekcie powstaje konceptualny model danych CDM (ang. Conceptual Data Model), czyli zbiór specyficznych dla danego projektu wymagań dotyczących danych. Model ten jest następnie przekształcany w fizyczny model danych PDM (ang. Physical Data Model). Normalizacja może posłużyć dla sprawdzenia otrzymanego fizycznego modelu danych, a przynajmniej jego krytycznych części.

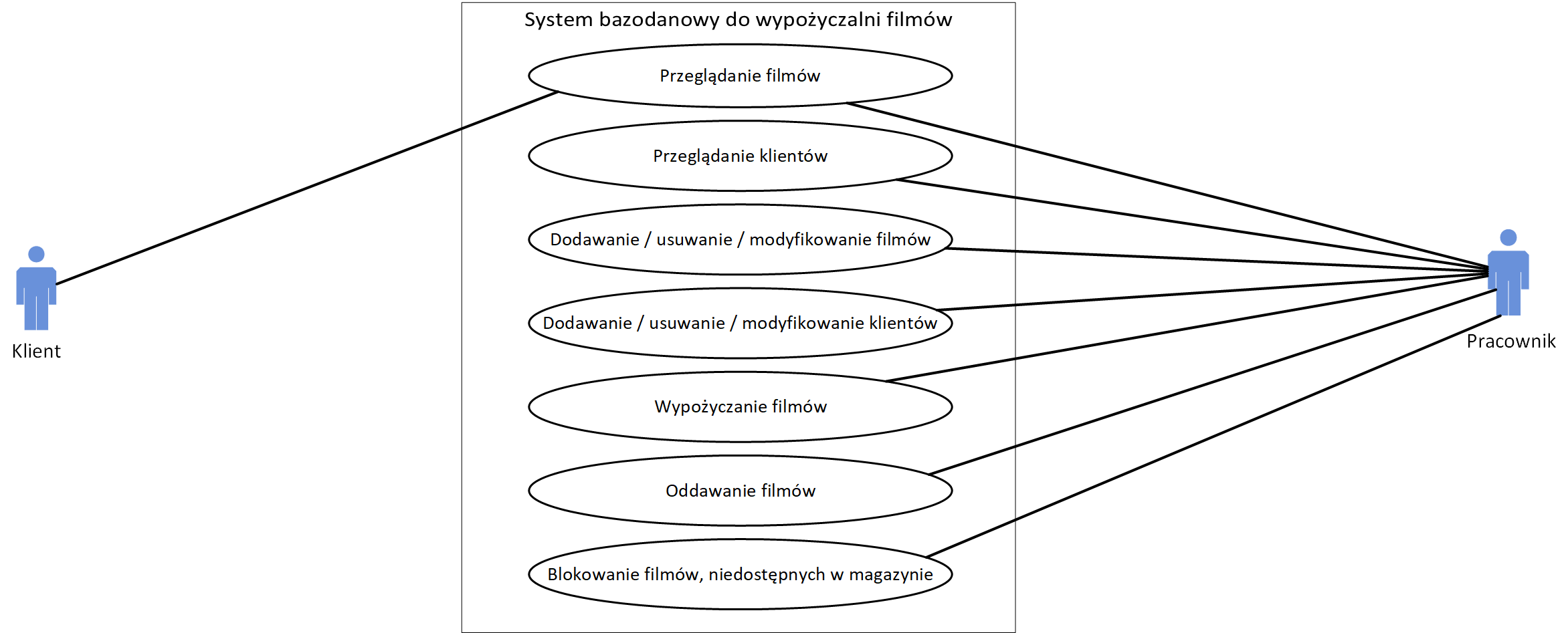
Modelowanie danych ułatwia nam na wprowadzenie zmian do projektu w każdej chwili, bez potrzeby ponownego przeprowadzania analizy danych.

Aby dogłębnie zrozumieć, przeanalizować oraz rozwiązać problem, w dalszej części pracy przygotowałem klika diagramów, które zostały opisane w poniższych podrozdziałach.

### 2.4.1. Model przypadków użycia

Skuteczność tworzenia systemu informatycznego jest ściśle powiązana z możliwością jednoznacznej specyfikacji i dokumentowania wymagań użytkowników wobec przyszłego systemu. W przypadku języka UML i metodyk RUP rolę metody definiowania metodyk systemowych pełnią diagramy przypadków użycia. Diagram przypadków użycia definiuje wszelkie interakcje zachodzące pomiędzy użytkownikami, a systemem w konkretnym scenariuszu, czyli przebiegu wydarzeń, nie wchodząc w szczegóły implementacji. Schemat będzie przedstawiał wszelkie czynności, jakie będzie wykonywała osoba, która będzie zarządzać bazą danych. Pokazuje on wymagania jakie osoby narzucają korzystające z tego systemu. Diagram przypadków użycia składa się z aktorów, przypadku użycia i komunikacji (przedstawionych w formie kresek pomiędzy aktorami, a przypadkami). Możemy również wiązać ze sobą przypadki użycia, łącząc include, extend i generalizację. Związek <<include>> przedstawia powiązanie zachodzące pomiędzy bazowym przypadkiem użycia, a przypadkiem zawieranym. Natomiast związek <<extend>> rozszerza przypadek użycia, który łączy się z przypadkiem rozszerzającym. Zatem możemy powiedzieć, że diagram przypadków użycia jest używany do określania wymagań systemowych, komunikacji z użytkownikami końcowymi i ekspertami z określonej dziedziny oraz testowania systemu.

Na poniższym Rysunku 2.3 został przedstawiony diagram przypadków użycia dla systemu baz danych do zarządzania wypożyczalnią płyt DVD. Jak możemy zauważyć w systemie określono dwóch aktorów *Pracownik* oraz *Klient*. Aktor o nazwie *Pracownik* będzie mógł skorzystać ze wszystkich funkcjonalności związanych z przeglądaniem klientów, modyfikacją filmów i klientów, zatwierdzaniem wypożyczonych filmów przez klienta, oddawaniem i blokowaniem filmów, które nie są dostępne w magazynie. Natomiast aktor o nazwie *Klient* będzie posiadał jedynie uprawnienia do przeglądania baz filmów, które są dostępne w bazie danych. Aktor *Pracownik* może dodać klienta wtedy, gdy chce on wypożyczyć film, ale nie jest dostępny w bazie danych. Dane mogą zostać zmienione w sytuacji, gdy klient przeprowadzi się w inne miejsce zamieszkania lub zmieni się jego stan cywilny w przypadku, gdy klientem jest kobieta. Dane klienta mogą zostać usunięte przez aktora *Pracownik* tylko wtedy, gdy klient go powiadomi o zrezygnowaniu z dalszych usług. Gdy aktor *Klient* wybierze film do wypożyczenia, aktor *Pracownik* zatwierdza film, który został wypożyczony przez klienta i umieszcza jego dane w tabeli „Wypożyczone”. Po wybraniu filmu przez klienta, *Pracownik* dodaje film do tabeli oraz ustawia datę oddania na termin 7 dni od dnia wypożyczenia. Gdyby się okazało, że *Klient* chcę wypożyczyć film, który nie jest dostępny w bazie danych, to system zablokuje możliwość wypożyczenia takiego filmu. Dodatkowo aktor *Pracownik* może przyjmować filmy, które *Klient* chce oddać. W przypadku, gdy aktor *Klient* odda film do wypożyczalni, *Pracownik* sprawdza dane klienta, oraz czy film został oddany w terminie. Jeżeli warunki zostaną spełnione przez klienta to w bazie danych zmieniony zostanie status wypożyczenia na „Zwrócono” oraz nie musi ponosić dodatkowych opłat. W przypadku, gdy aktor *Klient* nie odda filmu w terminie, to system uruchomi czynności związane z poniesieniem dodatkowej opłaty przez klienta. Aktor o nazwie *Pracownik* będzie także posiadał funkcjonalności związane z przeglądaniem klientów oraz wykonywanie operacji związane z modyfikacją filmów. Pracownik będzie mógł przejrzeć listę klientów, którzy świadczą usługi dla systemu wypożyczalni płyt DVD. Aktor o nazwie *Pracownik* dodaje lub modyfikuje w bazie danych w sytuacji, gdy film nie jest jeszcze dostępny w bazie danych i wypożyczalnia wykupiła filmy, które są popularne wśród klientów. W sytuacji, gdy filmy są rzadko lub nie są wypożyczane przez klientów, aktor *Pracownik* ma prawo usunąć wybrany film z bazy danych. Z kolei aktor o nazwie *Klient* będzie mógł skorzystać tylko z funkcjonalności, związanej z przeglądaniem filmów. Aktor *Klient* może wtedy zobaczyć listę wszystkich filmów dostępne w wypożyczalni oraz wybrać intersujący go film, gdzie może zgłosić aktorowi *Pracownik* chęć wypożyczenia wybranego przez siebie filmu.



Rysunek 2.3. Diagram przypadków użycia dla systemu bazy danych zarządzającą wypożyczalnią DVD.

Źródło: opracowanie własne

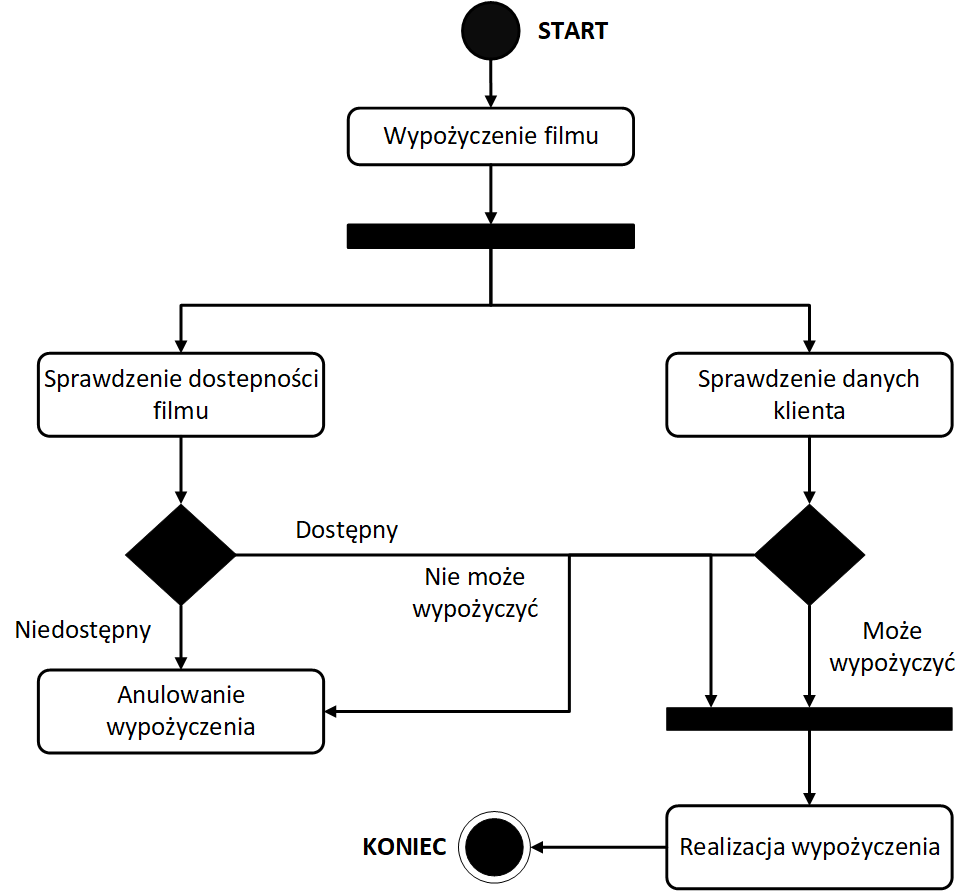
### 2.4.2. Diagram aktywności

Jak można było się dowiedzieć z pierwszego rozdziału, diagram czynności jest przedstawieniem w postaci graficznej procesów, na które składają się obiekty tworzące całą aplikację, bądź interesującą jej część. Za pomocą tego typu diagramów możemy z łatwością stworzyć aspekty programowania wielowątkowego, jak i dogłębnie je przeanalizować. Diagram aktywności składa się z kilku podstawowych pojęć, takich jak:

* aktywności – którą definiujemy jako funkcje wykonywane przez określone obiekty systemu. Ten rodzaj funkcjonalności może mieć również charakter sekwencyjny, który obejmuje kilka obiektów.
* przejść – występują one po wykonaniu funkcji oraz zrealizowaniu wszystkich warunków, które zostały nałożone na określone przejście.
* synchronizacji – polega ona na równorzędnym wykonywaniu aktywności przez kilka obiektów.

Z powodu złożoności systemu zostanie przedstawiona tylko jedna funkcjonalność (spośród siedmiu), która jest związana z wypożyczaniem filmów. Funkcjonalność ta została przedstawiona na Rysunku 2.4, która podpowie klientowi jakie są zasady wybierania filmów do wypożyczenia. Pierwszą czynnością, która zostanie wykonana, to *Wypożyczenie filmu*, gdzie pracownik odbywa osobistą rozmowę z klientem wypożyczalni. Klient po przejrzeniu listy wszystkich filmów, może wybrać filmy do wypożyczenia, pod warunkiem, że będą one dostępne w magazynie. Jeżeli klient zgłosi pracownikowi jakie chce wypożyczyć filmy oraz w jakich ilościach, pracownik wykonuje zapytanie do bazy danych, które umożliwi dodanie najważniejszych informacji takich jak, m.in. imię i nazwisko klienta, tytuł wypożyczonego filmu, czy datę wypożyczenia i oddania filmu. Zanim pracownik wykona zapytanie w bazie danych, to jednocześnie należy wykonać dwie czynności. Pierwszą czynnością jaką należy wykonać to *Sprawdzenie dostępności filmu*, która pozwoli pracownikowi sprawdzić, czy film jest dostępny w bazie. Druga czynność o nazwie *Sprawdzenie danych klienta*, sprawdza, czy dane klienta są zgodne z tymi jakie podał podczas rejestracji. Jeżeli się okaże, że film nie jest dostępny w bazie danych, to wykonana zostanie czynność *Anulowanie wypożyczenia*, pozwalająca pracownikowi anulować wypożyczenie filmu. W przeciwnym wypadku, gdy film jest dostępny w bazie danych wykonywana jest czynność o nazwie *Realizacja wypożyczenia*, zezwalająca pracownikowi wykonać zapytanie dodające najważniejsze dane dotyczące wypożyczonego filmu przez klienta. Jednocześnie z pierwszą czynnością jest wykonywana druga czynność *Sprawdzenie danych klienta*, pozwalająca pracownikowi sprawdzić poprawność danych. W przypadku, gdy po poprawnym zweryfikowaniu danych przez pracownika klient może wypożyczyć film jest wykonywana czynność *Realizacja wypożyczenia*. W innym wypadku, kiedy klient nie będzie mógł z danego powodu wypożyczyć filmu, będzie wykonywana czynność *Anulowanie wypożyczenia*.

Jeżeli chodzi o pozostałe funkcjonalności, jakie zostały wykonane w diagramie aktywności, to będą analogiczne, jak dla tej funkcji.



Rysunek 2.4. Diagram aktywności dla funkcji wypożyczenia filmów w systemie wypożyczalni płyt DVD.

Źródło: opracowanie własne

### 2.4.3. Diagram sekwencji

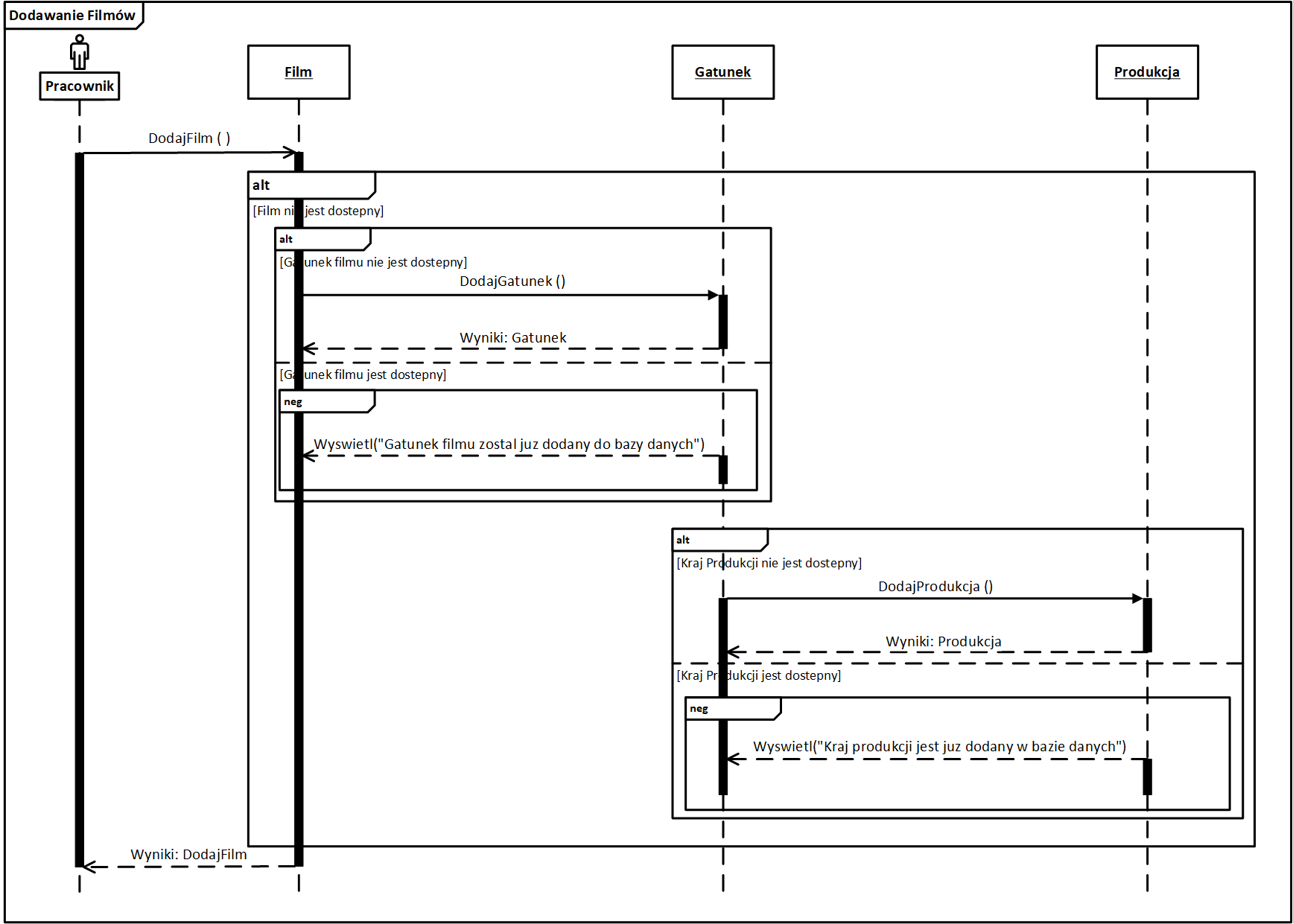
Jak już wiemy z pierwszego rozdziału diagram sekwencji jest jednym z modeli zachowania, który opisuje system od strony jego dynamiki. Ponadto można za jego pomocą przedstawić w sposób graficzny procesy, jakie zachodzą w systemie. Diagram sekwencji ilustruje wszystkich aktorów oraz obiekty, które partycypują w interakcji, jak i wszystkie wygenerowane przez nie zdarzenia, które są przedstawione w sposób chronologiczny. Jest on odwzorowaniem zdarzeń, które można przedstawić na diagramie przypadków użycia, z tą różnicą, że na dwóch płaszczyznach: poziomej i pionowej, odpowiadających odpowiednio obiektom i czasowi.

Schemat tworzą podstawowe elementy takie jak:

* aktor – który na diagramie jest oznaczonym symbolem człowieka,
* obiekt – jest on przedstawiony za pomocą prostokąta,
* linia życia – pionowa, przerywana linia, przedstawiająca czas istnienia danego elementu,
* aktywacja – prostokąt na linii życia, ilustrujący czas wykonania danej operacji,
* komunikaty – są reprezentowane na diagramie za pomocą różnego rodzaju strzałek. W diagramie sekwencji można wyróżnić cztery rodzaje komunikatów:
  + komunikat synchroniczny – oznacza przekazanie sterowania do klasyfikatora – odbiorcy. Kiedy zostanie on przesłany aktualny przepływ sterowania klasyfikatora – nadawcy ulega przerwaniu. Jest on wznawiany dopiero po wykonaniu przez klasyfikator – odbiorcę operacji inicjonowania przez komunikat.
  + komunikat asynchroniczny – nie powoduje przerwania aktualnego przepływu sterowania nadawcy. Instancja klasyfikatora, która wysyła komunikat nie oczekuje odpowiedzi, zarazem pozostając w stanie aktywności, co może umożliwić dalsze przetwarzanie bądź wysłanie komunikatów.
  + komunikat zwrotny – wskazuje na powrót sterowania do instancji klasyfikatora po wykonaniu komunikatu synchronicznego. Ten rodzaj komunikatu nie jest umieszczany w diagramach sekwencji obligatoryjnie. Funkcją komunikatu zwrotnego nie jest tylko przekazanie sterowania do danej instancji, ale też zainicjonowanie jej określonej operacji.
  + komunikat wewnętrzny – jest on rodzajem komunikatu, który obiekt wysyła sam do siebie. Tego rodzaju komunikatów używa się w sytuacji, gdy obiekt wywołuje swoją własną metodę.

Z powodu złożoności systemu zostanie przedstawiona tylko jedno zdarzenie, które będzie związane z dodawaniem filmów. Funkcjonalność dodawania filmów do bazy danych zarządzającej wypożyczalnią płyt DVD, została przedstawiona na Rysunku 2.5. Na samym początku aktor *Pracownik* przekazuje komunikat o nazwie *Dodaj Film*, gdzie wykonuje on zapytanie do bazy, pozwalające dodać nowy film do bazy danych. Od obiektu *Film* do obiektu *Produkcja* jest wykonywany operator alternatywy (alt), oznaczający możliwość wyboru jednego i tylko jednego spośród wszystkich operatorów danego wyodrębnionego fragmentu. Wybór ten dokonuje się na podstawie warunku, który jest przypisany do operandu. Za pomocą bloku alternatywy sprawdzamy, czy film nie jest dostępny w magazynie. Ten warunek jest zawsze spełniony tylko, kiedy się okaże, że film nie jest aktualnie dostępny na stanie. Wewnątrz pierwszego operatora alt, w obiekcie *Film* jest wykonywany kolejny operator alternatywny, który pozwoli zweryfikować czy gatunek filmu nie jest dostępny w bazie danych. Jeżeli warunek ten jest spełniony to z obiektu *Film* jest przekazywany do obiektu *Gatunek* komunikat *Dodaj Gatunek*. Aktor *Pracownik* wykonuje wtedy odpowiednie zapytanie do bazy, pozwalające na dodanie gatunku filmu do tabeli. Po dodaniu nowego rekordu do tabeli wszystkich gatunków do obiektu *Film* jest wysyłany komunikat zwrotny, gdzie system wyświetli pracownikowi komunikat o pomyślnym dodaniu gatunku do bazy. Wtedy nowy dodany rekord do tabeli *Gatunek* zostanie przypisany do identyfikatora gatunku, który będzie numerowany od ostatniej pozycji wstawionego wcześniej rekordu zwiększanego o wartość 1. W przeciwnym wypadku, kiedy gatunek filmu jest dostępny w bazie danych, to zostanie wykonany operator funkcjonalności nieprawidłowej (neg). Operator funkcjonalności nieprawidłowej, inaczej negacji, jest używany w przypadku działań nieprawidłowych. Takiego rodzaju operatory wskazują na wyjątki, które muszą zostać obsłużone. Z obiektu *Gatunek* zostanie przekazana do obiektu *Film* funkcja, która pozwoli wyświetlić komunikat o dodanym już gatunku filmu do bazy danych. Komunikat ten zostanie wyświetlony aktorowi *Pracownik* tylko wtedy, gdy spróbuje wykonać zapytanie do bazy wstawiając do kolumny Gatunek, taką samą wartość jaka została wcześniej dodana do tabeli gatunków. W tej sytuacji, kiedy wstawiony rekord już istnieje w bazie, to identyfikator gatunku zostanie przypisany do takiego ID, odpowiadającej wartości wstawionego wcześniej rekordu. Następnie do obiektu *Gatunek* jest wykonywany kolejny operator alternatywy, który sprawdzi, czy kraj produkcji nie jest dostępny w bazie danych. Jeżeli warunek ten jest spełniony, to z obiektu *Gatunek* jest przekazywany do obiektu *Produkcja* komunikat *Dodaj Produkcja*. *Pracownik* wykona wtedy zapytanie do bazy, pozwalające dodać nowy kraj produkcji do tabeli. Po wykonaniu takiego zapytania przez pracownika zostanie dodany nowy rekord w tabeli, zawierającą nową wartość. Wtedy do obiektu *Gatunek* jest wysyłany komunikat zwrotny, pozwalający wyświetlić komunikat o dodanym kraju produkcji do tabeli. Nowy dodany rekord obiektu *Produkcja*, tak jak było w przypadku obiektu *Gatunek* zostanie przypisany do identyfikatora produkcji. W przeciwnym wypadku, kiedy kraj produkcji jest dostępny w bazie danych, to zostanie wykonany operator negacji. Wtedy z obiektu *Produkcja* zostanie wysłany komunikat zwrotny do obiektu *Gatunek*, który poinformuje *Pracownika* o dodanym już wcześniej kraju produkcji do bazy danych. Wstawiona wcześniej wartość kraju produkcji zostanie przypisana do takiej wartości identyfikatora, która będzie odpowiadać wcześniej dodanej wartości. Kiedy nowe wartości gatunku oraz produkcji zostaną pomyślnie dodane do bazy danych, *Pracownik* może wykonać pełne zapytanie do bazy, które pozwoli dodać nowy film. Wtedy nowy dodany film zostanie dodany do bazy, do którego zostaną przypisane odpowiednie wartości identyfikatora gatunku oraz produkcji. W tym samym czasie z obiektu *Film* zostanie wysłany komunikat zwrotny, który wyświetli pracownikowi komunikat o pomyślnym dodaniu filmu do bazy danych.

Jeżeli chodzi o pozostałe zdarzenia systemu, jakie zostały wykonane w diagramie sekwencji, będą analogiczne, jak dla tej funkcjonalności.



Rysunek 2.5. Diagram sekwencji dla funkcji dodawania filmów w systemie wypożyczalni płyt DVD.

Źródło: opracowanie własne

### 2.4.4. Diagram klas UML

Utworzenie diagramu klas UML jest pierwszym etapem procesu budowy systemu informatycznego, gdzie należy ustalić za pomocą elementów graficznych, jakie informacje mają się znaleźć w systemie (atrybuty) i operacje które mogą być wykonane. Utworzone klasy muszą się ze sobą komunikować za pomocą powiązań, gdzie przekazywane informacje są zrozumiałe dla odbiorcy. Zatem możemy powiedzieć, że diagram klas jest graficznym przedstawieniem statycznych, abstrakcyjnych elementów z dziedziny przedmiotowej oraz związków jakie występują pomiędzy nimi. Celem tego etapu jest opracowanie diagramu klas, gdzie należy określić ich atrybuty i operacje, jakie mają zostać wykonywane. Dodatkowo do diagramu klas musimy utworzyć wszelkie związki zachodzące pomiędzy klasami, których wymaga system, aby mógł spełniać założone funkcjonalności.

Diagram klas pomaga analizować system informatyczny poprzez analizy strukturalne, a także wdrożenia za pomocą języka UML. UML pozwala na zdefiniowanie struktur danych oraz ich powiązań w sposób niezależny od docelowego diagramu związków encji (modelu konceptualnego).

#### 2.4.4.1. Tworzenie diagramu klas

Aby rozpocząć pracę nad tworzeniem systemu do zarządzania wypożyczalnią płyt DVD na naszym komputerze, należy uruchomić środowisko PowerDesigner, służący do modelowania systemów informatycznych w tym baz danych. Program został szczegółowo omówiony w rozdziale 1 (punkt 1.9). Zanim utworzymy diagram klas UML, w środowisku PowerDesigner należy utworzyć nowy model zorientowany obiektowo (PowerDesigner, 2011). Aby utworzyć nowy model w PowerDesignerze, z menu wybieramy opcję *File | New Model*. Po wybraniu opcji *New Model* pojawi się okno dialogowe, w którym po lewej stronie znajdują się trzy opcje.

Opcje są podzielone na następujące kategorie:

* *Categories* – zawiera zestaw predefiniowanych modeli i diagramów posortowanych w konfigurowalnej strukturze kategorii.
* *Model types* – zawiera klasyczną listę typów modeli i diagramów programu PowerDesigner.
* *Template files* – zawierają zestaw szablonów i modeli posortowanych według typu modelu [16].

Z opcji znajdujących się po lewej stronie okna *New Model*, klikamy przycisk *Model type*, klikamy opcję *Object – Oriented Model*. Dalej z panelu znajdującego się po prawej stronie wybieramy element o nazwie *Class Diagram*. W zależności od tego, jak skonfigurowano okno dialogowe *New Model*, elementy te mogą być pierwszymi diagramami lub szablonami, na których należy oprzeć tworzenie modelu. Możemy użyć przycisk *Views*, znajdujący się w prawym górnym rogu, aby sterować wyświetlaniem elementów. Następnie należy wprowadzić nazwę modelu. Kod modelu jest używany do generowania skryptu lub kodu i jest wprowadzany z tej nazwy za pomocą konwencji nazewnictwa modelu. Następnie wybieramy docelowy język obiektowy, który dostosowuje domyślne środowisko modyfikujące program PowerDesigner za pomocą określonych właściwości, obiektów i szablonów generowania. Domyślnie PowerDesigner tworzy łącze w modelu dla określonego pliku. Aby skopiować zawartość zasobu i zapisać go w pliku modelu, należy kliknąć przycisk *Embed Resource in Model*, znajdujący się po prawej stronie opcji *Object language*. Osadzenie pliku w ten sposób umożliwia wprowadzenie zmian specyficznych dla modelu bez wpływu na inne modele, odwołujące się do udostępnionego zasobu. Na koniec klikamy przycisk OK, aby utworzyć i otworzyć model zorientowany obiektowo. Model zorientowany obiektowo jest zapisywany do pliku o rozszerzeniu OOM (polecenie *File | Save*).

Aby utworzyć diagram klas UML w istniejącym modelu zorientowanym obiektowo w obszarze roboczym należy kliknąć prawym przyciskiem myszy i wybrać opcję *New | Class Diagram*, a następnie kliknąć przycisk OK.

#### 2.4.4.2. Określanie atrybutów i operacji klasy

**Klasa**

Klasa (PowerDesigner, 2011) jest przedstawieniem zbioru obiektów, które posiadają podobną strukturę i zachowanie oraz mają te same atrybuty, operacje, relacje i semantykę. Strukturę klasy opisują jej atrybuty i skojarzenia, a operacje wykonywane w danej klasie opisują jej zachowanie. Klasy i relacje, które tworzymy, tworzą podstawową strukturę modelu zorientowanego obiektowo [16].

Aby dodać nową klasę do naszego diagramu UML w środowisku PowerDesigner z menu *Toolbox* znajdującego się po prawej stronie programu, klikamy lewym przyciskiem myszy na sekcję o nazwie *Class Diagram*. Po rozwinięciu tej sekcji klikamy na opcję *Class* i najeżdżamy myszką na obszar roboczy, gdzie należy kliknąć na dany obszar lewym przyciskiem myszy. W naszym przypadku na obszar roboczy należy kliknąć 9 razy. Wtedy należy utworzyć klasy o następujących nazwach: *Production*, *Type*, *Actors*, *Films*, *Role*, *Employee*, *Loans*, *Clients* oraz *ClientsAddress*.

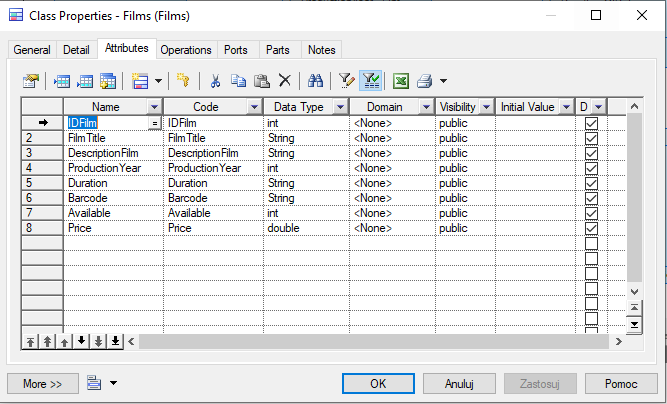
**Atrybuty**

Atrybut (PowerDesigner, 2011) jest właściwością klasy opisująca zakres wartości, jakie możemy przypisać do poszczególnych obiektów danej klasy. Klasa może posiadać kilka atrybutów lub nie posiadać żadnego. Każdy obiekt w klasie ma takie same atrybuty, ale ich wartości mogą być różne. Nazwy atrybutów w klasie muszą być unikalne. Możemy nadać identyczne nazwy dwóm lub więcej atrybutom tylko w sytuacji, gdy istnieją w różnych klasach [16].

Aby dodać atrybuty dla wybranej klasy, to w diagramie klas z obszaru roboczego klikamy dwukrotnie lewym przyciskiem myszy na wybraną klasę. Wtedy powinno się wyświetlić okno, zawierające właściwości klasy. Z menu wybieramy opcję o nazwie *Attributies*, gdzie następnie wybieramy opcję *Add a Row*. Kliknięcie tej opcji pozwoli nam na wprowadzenie wszystkich zmiennych, jakie chcemy umieścić w klasie i nadać im odpowiednie właściwości. Na potrzeby tworzonego systemu na rysunku 2.6 zostały przedstawione atrybuty, jakie się znajdą w klasie *Films*. Dla poszczególnych atrybutów klasy zostały ustawione modyfikatory dostępu na *public*, co oznacza, że będą one dostępne dla wszystkich obiektów. Modyfikator dostępu w diagramie klas, możemy ustawić w atrybutach w kolumnie *Visibility*. Kolumna ta pozwala na określenie widoczności obiektu, w sposób jaki on jest widziany poza otaczającą przestrzenią nazw. W sytuacji, gdy klasa jest widoczna dla innego obiektu, może to wpłynąć na strukturę lub zachowanie obiektu lub podobnie, inny obiekt może wpłynąć na właściwości klasy. W kolumnie *Code* zostały wpisane takie same nazwy, jakie znajdują się w kolumnie *Name*. Nazwy w kolumnie Code zostaną użyte do generowania kodu podczas przekształcenia modelu klas na konceptualny model danych. Kod jest generowany na podstawie nazwy przez zastosowanie konwencji nazewnictwa, które zostały określone w opcjach modelu. We wszystkich atrybutach nie zostały określone domeny o czym wskazuje kolumna *Domain* oraz nie zostały przypisane wartości początkowe dla poszczególnych atrybutów (kolumna *Initial Value*). Klasa Films będzie zawierała 8 atrybutów o następujących nazwach:

* + 1. *IDFilm* – pole jest liczbą całkowitą typu int, która identyfikuje wybrany film w systemie.
    2. *FilmTitle* – pole typu string, które w pamięci systemu przechowuje dane na temat tytułu filmu.
    3. *DescriptionFilm* – atrybut typu string, zawierający szczegółowy opis filmu.
    4. *ProductionYear* – atrybut typu int, zawierający rok produkcji wybranego filmu.
    5. *Duration* – pole przechowujące wartość czasu trwania filmu, która zawiera tekst typu string.
    6. *Barcode* – pole zawiera kod kreskowy, jaki jest przypisany do odpowiedniego filmu. Atrybut przechowuje wartości typu string.
    7. *Available* – wartość przypisana do tego atrybutu przechowuje liczbę dostępnych filmów w magazynie. Atrybut ten przyjmuje wartości typu int.
    8. *Price* – pole typu double, które przechowuje cenę filmu.

Procedura dodawania atrybutów do innych klas w systemie będzie wyglądała analogicznie do tej, jak to zostało przedstawione poniżej.

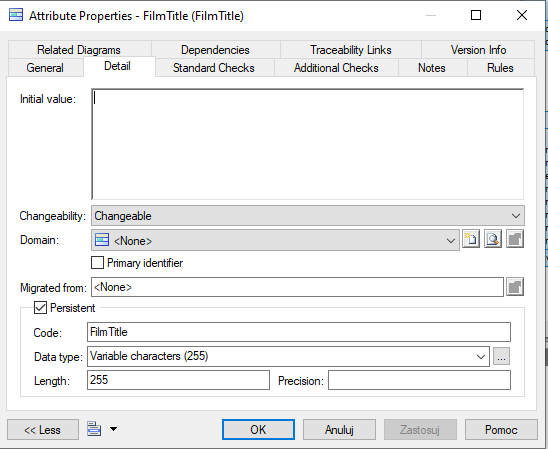


Rysunek 2.6. Lista atrybutów zdefiniowanych w klasie Films.

Źródło: opracowanie własne

Następnie musimy zdefiniować jakie pola będą zawierały poszczególne atrybuty po przekształceniu modelu klas na model konceptualny. Można to zrobić klikając we właściwościach klasy zakładki *Attributies* dwukrotnie lewym przyciskiem myszy na zdefiniowany atrybut. Wtedy powinno się wyświetlić osobne okno zawierające właściwości atrybutu. Z menu znajdującego się u góry wybieramy opcję *Detail*, aby określić nazwę pola oraz typ danych, jaki będzie posiadał atrybut klasy po wygenerowaniu diagramu do modelu konceptualnego. Na Rysunku 2.7 zostały przedstawione właściwości atrybutu *FilmTitle*, znajdujący się w klasie *Films*. Po przekształceniu na model konceptualny atrybut ten będzie posiadał taką samą nazwę oraz będzie polem tekstowym typu *Variable* *characters* o długości 255 znaków.

Zaznaczone pole *Persistent* oznacza, że atrybut będzie utrwalany i przechowywany w bazie danych. W sytuacji, gdy atrybut klasy jest identyfikatorem, we właściwościach klasy należy zaznaczyć opcję *Primary* *identifier*. Identyfikator jest podstawową częścią dla określonego atrybutu. Identyfikatory podstawowe są konwertowane na klucze główne po przekształceniu diagramu klas na model fizyczny PDM.

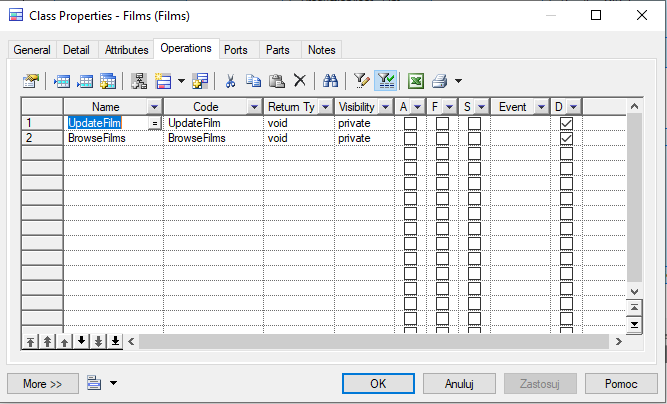


Rysunek 2.7. Właściwości atrybutu FilmTitle w klasie Films.

Źródło: opracowanie własne

**Operacje**

Operacja (PowerDesigner, 2011) jest właściwością klasy, która określa, w jaki sposób wywołać określone zachowanie. Jest to specyfikacja zapytania, do wykonania którego obiekt może zostać wywołany. Klasa może posiadać dowolną liczbę operacji lub nie mieć ich wcale. Operacja posiada nazwę oraz listę parametrów. Kilka operacji może mieć taką samą nazwę w ramach tej samej klasy w sytuacji, jeśli ich parametry są różne. [16] Aby dodać nową operację w klasie należy kliknąć na obszar roboczy dwukrotnie lewym przyciskiem myszy na daną klasę. Powinno się wyświetlić osobne okno zawierające właściwości klasy, gdzie w menu należy wybrać zakładkę *Operations*. Następnie w karcie operacji klikamy na jedno z następujących przycisków, gdzie w naszym przypadku wybieramy opcję *Add a Row / Insert a Row*. Po kliknięciu na tę opcję możemy wprowadzić nazwę operacji i inne odpowiednie właściwości. Przykładowo na rysunku 2.8 została przed-stawiona lista operacji, które zostały zdefiniowane dla klasy *Films*. Dla tej klasy zostały zdefiniowane dwie operacje *UpdateFilm* oraz *BrowseFilms*. Operacja *UpdateFilm* będzie funkcją, odpowiadająca za aktualizację filmów wykonywanych w bazie danych. Z użyciem tej funkcji będzie można dodać, edytować oraz usunąć film z bazy. Natomiast druga operacja o nazwie *BrowseFilms* będzie odpowiedzialna za wyszukanie filmów, które będą dostępne w bazie. Filmy będzie można wtedy wyszukiwać według określonych kryteriów. Operacje te nie będą zwracały żadnych wartości, gdzie obie funkcje zwracają typ *void*. Dodatkowo operacje będą widoczne tylko w odrębie danej klasy, do której należą, o czym mówi kolumna *Visibility* ustawione na wartość *private*. Dla pozostałych klas w naszym diagramie proces dodawania operacji będzie wyglądał analogicznie do tego, co zostało przedstawione na poniższym rysunku.



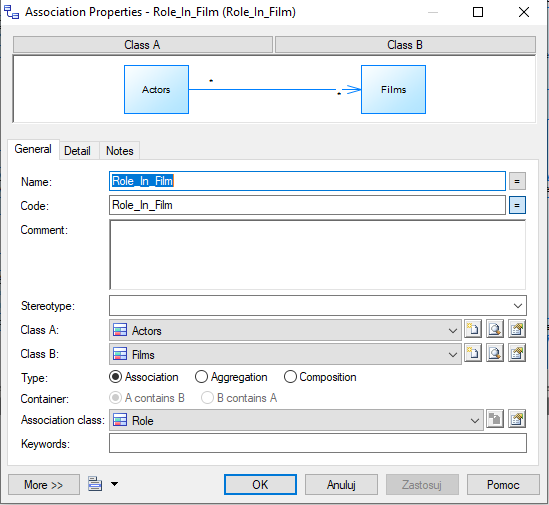
Rysunek 2.8. Lista operacji zdefiniowanych w klasie Films.

Źródło: opracowanie własne

#### 2.4.4.3. Określenie związków pomiędzy klasami

Asocjacja (PowerDesigner, 2011) jest związkiem pomiędzy dwoma lub więcej klasami lub między klasą a interfejsem, która opisuje powiązania występujące pomiędzy ich instancjami. Należy ona do silniejszych typów powiązań, oznaczająca dłuższy czas trwania związku pomiędzy dwiema klasami. Jest ona rysowana na diagramie jako linia ciągła między parą obiektów. Oprócz nazwania samego powiązania możemy określić nazwę roli dla każdego końca, aby opisać funkcję klasy z punktu widzenia klasy przeciwnej. [16]

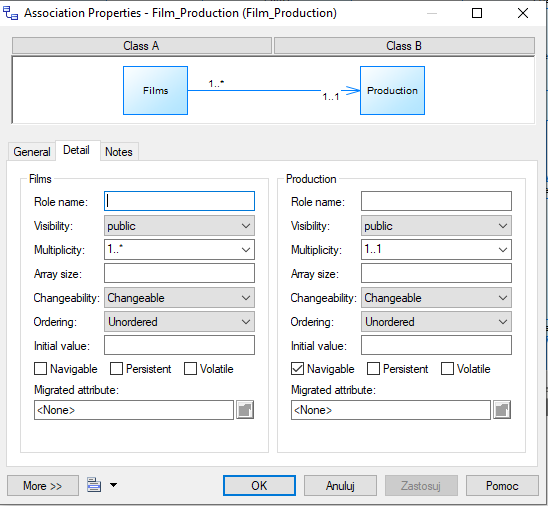
Na poniższym rysunku 2.9 zostały przedstawione ogólne właściwości asocjacji *Film\_Production*, jaka będzie zachodzić pomiędzy klasą *Films*, a klasą *Production*. Nową asocjację możemy utworzyć, wykorzystując sekcję *Toolbox* znajdującą się z prawej strony programu PowerDesigner. W tej sekcji po kliknięciu na opcję *Class Diagram* wybieramy przycisk *Association* i określamy relację klikając na wybraną klasę. W tej sytuacji, jak to zostało pokazane na poniższym przykładzie, klikamy w diagramie na klasę *Films* i przeciągamy myszką na klasę *Production*, aby utworzyć nową asocjację. Następnie klikamy dwukrotnie lewym przyciskiem myszy na utworzoną asocjację, aby ustawić nazwę oraz określić liczebność, jaka będzie zachodzić pomiędzy klasami. Po kliknięciu na asocjację wyświetli się okno zawierające właściwości asocjacji. W zakładce *General* należy określić nazwę asocjacji oraz nazwę jaka zostanie użyta do wygenerowania diagramu klas na model konceptualny. W naszym przypadku asocjacja będzie nosiła nazwę *Film\_Production*. Pola o nazwach *Class A* i *Class B* określają klasy na każdym końcu asocjacji. Możemy użyć narzędzi, które znajdują się po prawej stronie tych pól, aby tworzyć lub przeglądać właściwości aktualnie wybranej klasy. Dla asocjacji *Film\_Production* ustawiamy typ na opcję *Association* klikając we właściwościach asocjacji odpowiedni przycisk typu „radio”.



Rysunek 2.9. Właściwości ogólne asocjacji "Film\_Production".

Źródło: opracowanie własne

Dalej należy określić szczegóły asocjacji *Film\_Production* ustawiając liczebność, jaka będzie zachodzić pomiędzy klasami. We właściwościach asocjacji należy przejść do zakładki *Detail*, gdzie dla klasy *Films* w opcji *Multiplicity* wybieramy liczebność „1..\*”. Opcja *Multiplicity* określa dopuszczalną liczebność roli, która nazywana jest wielokrotnością. Wielokrotność wskazuje maksymalną i minimalną liczebność, jaka może mieć dana rola. Ustawienie dla klasy *Films* liczebności „1..\*” oznacza, że wartości w klasie mogą przyjmować jeden lub więcej obiektów. W klasie *Production* ustawiamy wartość opcji liczebności na „1..1”, pozwalająca przyjmować tylko jeden obiekt. Dla danej asocjacji możemy ustawić zmienność, o czym wskazuje opcja *Changeability*. Opcja ta pozwala określić, czy zestaw łączy związanych z obiektem może być modyfikowany po zainicjonowaniu obiektu. Dla obydwu klas opcja ta jest ustawiona na *Changeable*, która pozwala na dowolne dodawanie, edycję oraz usuwanie obiektów. Asocjacja pozwala użytkownikowi także na sortowanie obiektów, gdzie jest uwzględniana w porządkowaniu, które sortuje listę skojarzeń według kolejności ich tworzenia. Opcja ta we właściwościach asocjacji nosi nazwę *Ordering*, gdzie dla klas *Films* i *Production* została ustawiona na wartość *Unordered*. Wartość *Unordered* oznacza, że koniec powiązania nie jest posortowany ani uporządkowany. We właściwościach asocjacji w klasie *Production* należy zaznaczyć opcję *Navigable*, która określa, że informacje mogą być przesyłane między dwoma obiektami, które są połączone relacją. Szczegóły asocjacji dla właściwości *Film\_Production* zostały przedstawione na Rysunku 2.10.

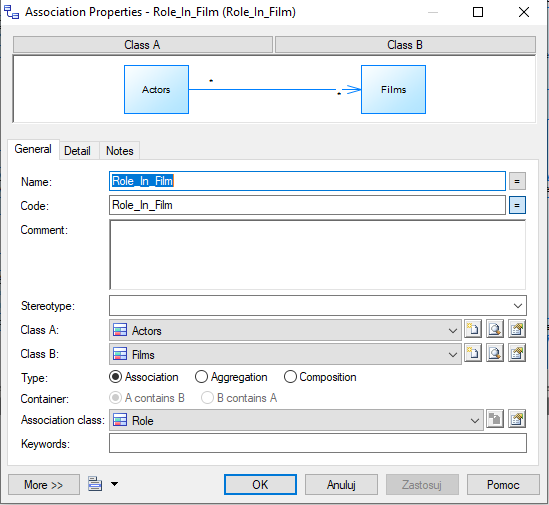


Rysunek 2.10. Szczegóły asocjacji dla właściwości "Film\_Production".

Źródło: opracowanie własne

W diagramie klas jest możliwość dodania właściwości do asocjacji między klasami, tworząc klasę asocjacyjną. Służy ona do dalszego definiowania właściwości asocjacji poprzez dodawanie atrybutów i operacji do asocjacji. Klasa asocjacyjna jest asocjacją, która ma właściwości klasy lub klasa właściwości asocjacji. Na diagramie symbol klasy asocjacyjnej jest połączeniem pomiędzy asocjacją, a klasą. Klasy asocjacyjne muszą się znajdować w tym samym pakiecie co asocjacja nadrzędna. Nie można użyć skrótu klasy do utworzenia klasy asocjacyjnej. Klasa, która została użyta do dodania nie może zostać ponownie wykorzystana dla innej klasy asocjacji.

Na poniższym rysunku 2.11 zostały przedstawione właściwości asocjacji dla *Role\_In\_Film*, określającą rolę jaką będzie aktor grał w filmie. Aby utworzyć klasę asocjacyjną należy w sekcji *Toolbox* znajdującej się z prawej strony programu PowerDesigner kliknąć na opcję *Class Diagram*, a następnie wybrać *Association*. Po kliknięciu na tą opcję w obszarze roboczym klikamy na opcję *Actors* i przeciągamy myszką na klasę *Films* w celu utworzenia nowej asocjacji. Następnie klikamy dwukrotnie na utworzoną asocjację i we właściwościach w opcji *Name* i *Code* wprowadzamy nazwę jaka została wymieniona powyżej. Dalej wskazujemy w opcji *Association* class klasę *Role*, która będzie powiązana z bieżącą asocjacją, która uzupełnia definicję asocjacji. Dodatkowo należy w zakładce *Detail* określić liczebność, jaka będzie zachodzić pomiędzy klasami. W tym przypadku zarówno dla klasy *Actors* i *Films* należy ustawić liczebność na wartość „\*”. Nadanie takiej wartości będzie oznaczało, że klasy będą mogły przyjąć nieograniczoną liczbę obiektów.

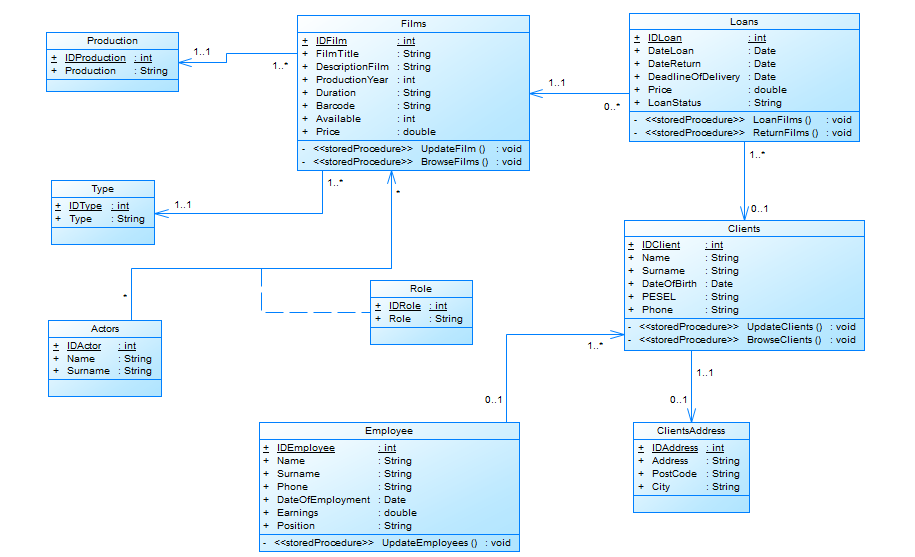


Rysunek 2.11. Właściwości ogólne asocjacji "Role\_In\_Film".

Źródło: opracowanie własne

#### 2.4.4.4. Diagram klas dla systemu wypożyczalni płyt DVD

Diagram klas dla bazy danych zarządzającej wypożyczalnią płyt DVD został przedstawiony na rysunku 2.12. Pracę nad modelem klas rozpoczyna się od utworzenia klas, które będą identyfikowały obiekty o wspólnym stanie i zachowaniu. Następnie do poszczególnych klas są przypisywane atrybuty charakteryzujące pojedyncze obiekty lub grupy obiektów, tworząc niezależną ich instancję. Jeżeli w danej klasie lub poza nią będą wykonywane poszczególne czynności, to należy utworzyć operacje, które będą przedstawiały określone zachowanie na poszczególnych obiektach. Na sam koniec na diagramie należy określić odpowiednie powiązania, jakie będą zachodziły między klasami.



Rysunek 2.12. Diagram klas UML dla systemu wypożyczalni płyt DVD.

Źródło: opracowanie własne

Rolą systemu wypożyczalni płyt DVD, który został pokazany na powyższym rysunku jest odpowiednie zarządzanie wypożyczalnią. Główną klasą, która jest niezbędna do działania systemu to klasa *Loans*. Klasa będzie przechowywała najważniejsze informacje dotyczące filmów wypożyczonych przez klientów. Każde wypożyczenie będzie się odwoływać do aktualnie wypożyczonego filmu z klasy *Films* oraz do wybranego klienta z klasy *Client*. Wypożyczenie musi dotyczyć tylko jednego filmu, natomiast wypożyczenie może być przypisane do żadnego lub jednego klienta. Klient będzie mógł wypożyczać filmy, które będą uporządkowane według produkcji (atrybut *Production* w klasie *Production*) i gatunku (atrybut *Type* w klasie *Type*). Następnie w ciągu 7 dni od wypożyczenia filmu, klient musi oddać filmy do wypożyczalni, gdzie zostanie naliczona opłata. W sytuacji, kiedy filmy zostaną oddane po określonym terminie (o czym wskazuje atrybut *DeadlineOfDelivery*), klient musi ponieść dodatkową opłatę za nieoddanie filmu na czas.

Wszystkie filmy zarejestrowane w systemie są reprezentowane przez obiekty klasy *Films*. W tej klasie będą przechowywane filmy, które będą dostępne w magazynie. Każdy film w systemie będzie zawierał tylko jedną produkcję oraz jeden gatunek. Natomiast jedna produkcja oraz gatunek muszą dotyczyć jednego lub więcej filmów. Jeden film, który zostanie wybrany do wypożyczenia, może zostać wypożyczony zero lub wiele razy. Za pomocą operacji *UpdateFilm* będzie można dodawać, modyfikować lub usunąć wybrany film z systemu, gdzie to będzie zależało od wykonywanej operacji. W przypadku wykonywania tej operacji należy wykorzystać atrybuty *Production* oraz *Type* z klas o takich samych nazwach. Do tych atrybutów możemy się odwołać za pomocą identyfikatorów, które wartości zostaną przypisane do danego obiektu. Za pomocą operacji *BrowseFilms* będzie można wyszukać film, według określonych kryteriów. Każdy klient będzie mógł wyszukać ulubiony film według atrybutów: *FilmTitle*, *Production* oraz *Type*.

Klasa *Employee* reprezentuje wszystkich pracowników, którzy pracują w wypożyczalni. Każdy pracownik będzie posiadał w systemie podstawowe dane takie jak m.in. imię i nazwisko (atrybuty *Name* i *Surname*), datę zatrudnienia (atrybut *DateOfEmployment*) czy zajmowane stanowisko (atrybut *Position*). Pracownik będzie zarabiał (atrybut *Earnings*) w zależności od zajmowanego stanowiska. Każdy pracownik może obsłużyć jednego lub więcej klientów w wypożyczalni, natomiast klient może zostać obsłużony przez zero lub jednego pracownika. Za pomocą operacji *UpdateEmployees*, będzie zapewniona możliwość dodawania, edytowania i usuwania pracowników w zależności od wykonywanej czynności przez administratora systemu. Pracownik zatrudniony przez właściciela będzie posiadał unikatowy numer (atrybut *IDEmployee*), który bardzo się przyda w czasie obsługi klienta.

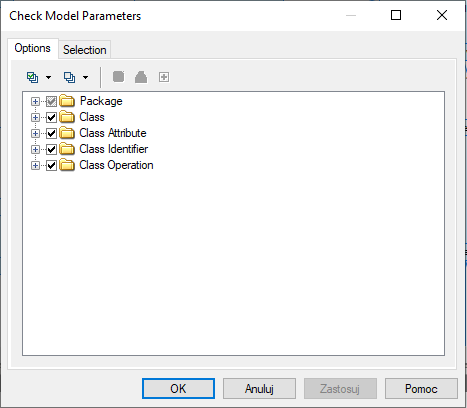
Wszyscy klienci zarejestrowani w systemie będą przechowywani w obiektach klas *Clients* oraz *ClientsAddress*. W tych klasach będą przechowywane wszystkie dane klientów wraz z danymi adresowymi, którzy korzystają z usług wypożyczalni. Dane adresowe z klasy *ClientsAddress* zostaną połączone przy pomocy asocjacji z klasą *Clients*, gdzie wartość atrybutu *IDAddress* będzie się odwoływać do konkretnego adresu klienta z klasy *Clients*. Klient może posiadać zero lub jeden adres, natomiast jeden adres musi być przypisany do jednego klienta. Klienci mogą zostać obsłużeni przez żadnego lub jednego pracownika, w zależności od tego, czy klient chce wypożyczyć film. Jeżeli klient jest zdecydowany na wypożyczenie, to może wybrać jeden lub wiele filmów do obejrzenia. W klasie *Clients* będzie możliwość wykonywania dwóch operacji. Operacja *UpdateClients* pozwala nam na dodawanie, edycję i usuwanie klientów w zależności od wykonywanej czynności. W przypadku dużej liczby klientów korzystających z usług wypożyczalni, bardzo przydatną operacją w systemie jest *BrowseClients*. Operacja pozwoli na wyszukanie klienta, który korzysta z wszystkich przysługujących usług wypożyczalni. Klienta będzie można wtedy wyszukać po atrybutach *Name* oraz *Surname* z klasy *Client*.

Na diagramie klas UML została utworzona klasa asocjacyjna *Role*, która dokumentuje asocjację pomiędzy klasami *Actors* a *Films*. Z jedną rolą może być związana nieograniczona liczba aktorów. Natomiast do jednej roli może być przypisane wiele filmów. W diagramie klas klasę asocjacyjną wprowadza się poprzez ścieżkę asocjacyjną, która jest przedstawiona w postaci linii przerywanej.

#### 2.4.4.5. Generowanie modelu konceptualnego

Gdy skończymy pracę nad diagramem klas UML, to należy sprawdzić, czy model został wykonany poprawnie. Środowisko PowerDesigner umożliwia automatyczne sprawdzanie poprawności modeli. Funkcja sprawdzania poprawności może wykryć błędy związane z użyciem poszczególnych obiektów w modelach. Podczas sprawdzenia poprawności modelu są wyświetlane ostrzeżenia, dotyczące sytuacji, które potencjalnie mogą prowadzić do zaistnienia błędu. Niektóre rodzaje błędu mogą się naprawić automatycznie. Sprawdzenie poprawności modelu uruchamiamy zawsze przed operacją generowania innego modelu lub kodu. Jeżeli zostaną znalezione błędy, to wówczas operacja zostanie anulowana. Aby dokonać sprawdzenia poprawności modelu klas UML, w programie PowerDesigner należy skorzystać z polecenia Tools | Check model lub nacisnąć klawisz F4.

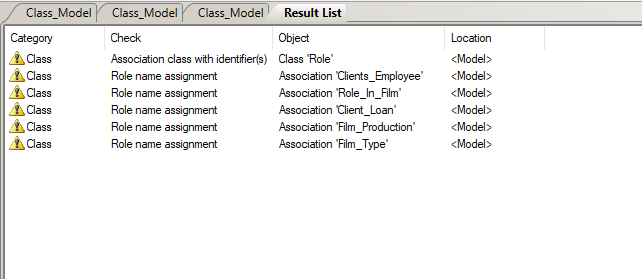
Po uruchomieniu procedury sprawdzania modelu wyświetlane jest okno parametrów z dwoma zakładkami (Rysunek 2.13). W zakładce Options można wybrać typy błędów i ostrzeżeń, które będą brane pod uwagę w trakcie sprawdzania. Natomiast w zakładce Selection można wskazać obiekty, które mają zostać sprawdzone. Domyślnie są tutaj zaznaczone wszystkie rodzaje błędów i obiekty istniejące w tym modelu.



Rysunek 2.11. Okno z parametrami sprawdzania modelu na przykładzie modelu klas UML.

Źródło: opracowanie własne

Po wykonaniu sprawdzenia poprawności modelu klas na liście wyników zostaną wyświetlone wszystkie znalezione błędy i ostrzeżenia (Rysunek 2.14). Błędy muszą zostać obowiązkowo usunięte, natomiast poprawki związane z ostrzeżeniami nie są wymagane. Ostrzeżenia, które są wyświetlane na liście wyników dotyczą sytuacji nie do końca poprawnych, ponieważ mogą spowodować zaistnienie błędu w dalszych etapach prac nad projektem bazy danych.



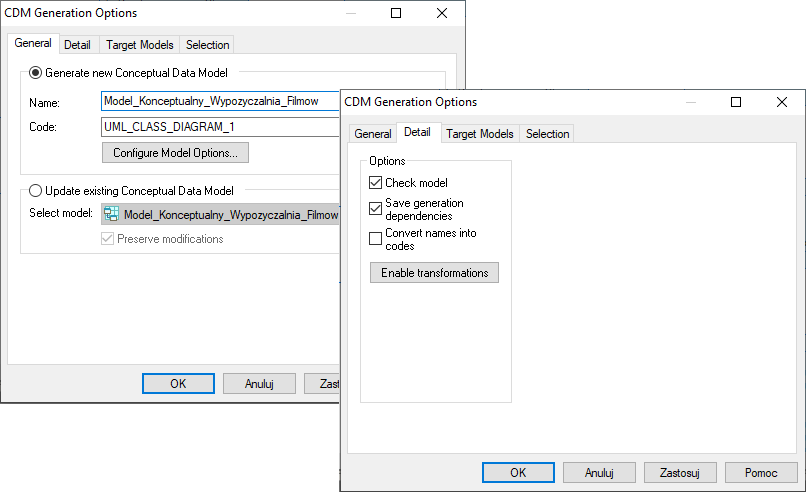
Rysunek 2.14. Lista wyników sprawdzania modelu na przykładzie diagramu klas.

Źródło: opracowanie własne

Aby uzyskać więcej informacji na temat konkretnego błędu lub ostrzeżenia, należy zaznaczyć odpowiedni element na liście rezultatów i wybrać polecenie *Help* z menu kontekstowego (otwieranego prawym przyciskiem myszy). Wyświetlony zostanie odpowiedni fragment dokumentacji z systemu pomocy. Jeżeli dla danego typu błędu możliwa jest korekta automatyczna, to można ją wykonać za pomocą polecenia *Automatic Correction*. W przypadku, gdy błąd wymaga poprawek ręcznych, należy wybrać polecenie *Correct*. Otwarte zostanie okno właściwości obiektu, którego dotyczy wybrany błąd.

Po sprawdzeniu poprawności modelu klas UML musimy przekształcić model klas UML na model związków encji (model konceptualny). Operacja generowania modelu zorientowanego obiektowo pozwala na tłumaczenie obiektów OOM na obiekty CDM. Po wygenerowaniu modelu klas na modelu konceptualny będziemy mogli bez problemu dalej projektować utworzony model i w ostateczności przekształcić model związków encji na model fizyczny PDM. W celu przekształcenia modelu klas, należy w programie PowerDesigner z menu górnego wybrać opcję *Tools | Generate Conceptual Data Model*. Po wybraniu tej opcji pojawi się okno generowania CDM (Rysunek 2.15). Za pomocą odpowiednich pól wybieramy opcję generowania nowego modelu konceptualnego (*Generate new Conceptual Data Model*) lub aktualizację modelu istniejącego (*Update existing Conceptual Data Model*). Nazwę nowego modelu należy wpisać w polu *Name*. Pole *Code* służy do generowania kodu i skryptów, które są generowane na podstawie nazwy przez zastosowanie konwencji nazewnictwa w określonych opcjach modelu. Po kliknięciu na przycisk *Configure Model Options...* możemy dokonać ustawień tworzonego modelu związków encji, które będą miały wpływ na działanie wszystkich obiektów. Domyślnie nie są skonfigurowane opcje modelu konceptualnego.

Na zakładkach okna opcji generowania modelu konceptualnego możemy znaleźć kilka ustawień. Możemy wymienić wiele z nich. W zakładce *Detail* sekcji *Options* możemy znaleźć pole wyboru *Check* *model*, za pomocą którego możemy określić, czy model klas UML ma zostać sprawdzony przed wygenerowaniem CDM. Opcja *Save generation dependencies* pozwala na zapisanie w generowanym modelu konceptualnym powiązań między każdym obiektem źródłowym, a odpowiednim obiektem docelowym. Umożliwi to późniejszą identyfikację obiektów, nawet jeśli zostały one zmienione. W zakładce *Selection* możemy wybrać klasy, które zostaną wzięte pod uwagę w trakcie generowania modelu związków encji. Domyślnie wybrane są wszystkie klasy, które zostały utworzone. [16]



Rysunek 2.15. Okno opcji generowania modelu konceptualnego.

Źródło: opracowanie własne

Szczegółowy opis pozostałych opcji generowania modelu konceptualnego możemy znaleźć w dokumentacji programu PowerDesigner. Zatwierdzenie wszystkich ustawień spowoduje uruchomienie sprawdzania poprawności modelu (zaznaczenie opcji *Check Model* w zakładce *Detail*) oraz wygenerowanie CDM. W przypadku, gdy model konceptualny CDM będzie zawierał błędy, operacja zostanie przerwana i pojawi się odpowiedni komunikat. Jeżeli generowanie modelu zakończy się pomyślnie, to wtedy otworzy się wygenerowany model konceptualny naszego systemu.

### 2.4.5. Diagram związków encji

Podstawowym zadaniem projektanta bazy danych podczas tworzenia modelu konceptualnego CDM jest zidentyfikowanie obiektów zwanych encjami oraz określenie związków, jakie występują między nimi. Model konceptualny pozwala na zdefiniowanie struktur danych oraz ich powiązań w sposób niezależny od docelowego SZBD.

Graficzną reprezentację elementów struktury danych w modelu konceptualnym stanowi diagram związków encji (ERD). Diagramy ERD umożliwiają spójne, efektywne i czytelne modelowanie danych oraz ułatwiają komunikację członków zespołu projektowego z przyszłymi użytkownikami bazy danych. Celem tego etapu jest zidentyfikowanie wszystkich danych, dla których zostały wygenerowane obiekty z diagramu klas. Gotowy model konceptualny możemy przekształcić na model fizyczny PDM, gdzie zostanie utworzony plan projektu relacyjnej bazy danych. Wynikiem będzie model fizyczny PDM, który zostanie zaimplementowany zgodnie ze schematem opracowanym w postaci diagramu CDM.

Aby utworzyć nowy model konceptualny w PowerDesignerze (PowerDesigner, 2005), to w menu znajdującym się u góry wybieramy opcję File | New. Po wybraniu opcji Conceptual Data Model otwarte zostanie puste okno diagramu oraz paleta narzędzi CDM, która będzie się znajdować w sekcji Toolbox. Nazwę oraz inne parametry modelu definiujemy w oknie właściwości modelu otwieranym poleceniem Model | Model Properties. Model konceptualny jest zapisywany do pliku o rozszerzeniu CDM (polecenie File | Save). [15] W naszym przypadku nie trzeba tworzyć nowego modelu konceptualnego, ponieważ został on wygenerowany na podstawie modelu klas UML (punkt 2.4.4.5).

#### 2.4.5.1. Encje

Encja (PowerDesigner, 2005) jest obiektem, który należy do modelowanego fragmentu świata rzeczywistego. Encja ma zostać uwzględniona w projektowanej bazie danych. Obiekt reprezentowany przez encję musi być jednoznacznie identyfikowany, czyli odróżniany od innych obiektów w danym fragmencie rzeczywistości.

Przy wyodrębnieniu encji wygodnie jest posługiwać się zasadą, która mówi, że budując model danych oparty na diagramach ERD, należy zdefiniować encje odpowiadające obiektom, dla których trzeba przechowywać dane na temat wielu ich własności. Przydatne jest także stwierdzenie, że encje często odpowiadają rzeczownikom występującym w wymaganiach funkcjonalnych dla projektowanej bazy danych.

Zdefiniowanie nowej encji w modelu konceptualnym wymaga:

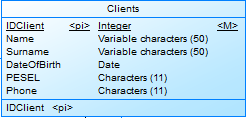
* Określenia atrybutów encji,
* Dokonania wyboru typu danych dla atrybutów,
* Zidentyfikowania atrybutów wymaganych i opcjonalnych,
* Wyboru atrybutu (lub atrybutów), który będzie stanowił identyfikator encji.

Dodanie nowej encji odbywa się przez wybór odpowiedniego narzędzia z opcji *Conceptual Diagram* z sekcji *Toolbox* lub za pomocą listy encji, która jest otwierana za pomocą *Model | Entities*. Po umieszczeniu opcji na diagramie wyświetlamy okno własności dwukrotnie klikając lewym przyciskiem myszy na jej symbolu (wskaźnik musi mieć postać strzałki) lub wybierając polecenie *Properties* z grupy *View* z paska menu lub też z menu kontekstowego.

W oknie właściwości encji na zakładce *General* można m.in. zmodyfikować nazwę encji (*Name*) oraz określić szacowaną liczbę wystąpień (*Number*). [15]

Na rysunku 2.16 przedstawiono przykład symbolu encji, który zostanie zastosowany w systemie bazy danych dla wypożyczalni płyt. Wgląd oraz szczegóły, które są wyświetlane wewnątrz symbolu mogą się różnić w zależności od ustawień wyświetlania (*Tools | Display Preferences*). Encja *Clients* będzie przechowywała dane klientów, którzy będą świadczyć usługi w wypożyczalni. Nazwa encji jest wyświetlana u góry symbolu, poniżej znajduje się lista atrybutów. Encja ta zawiera 6 atrybutów takich jak identyfikator klienta, imię i nazwisko, data urodzenia, PESEL i numer telefonu. Przy prawej stronie nazw atrybutów, są wyświetlane oznaczenia typów danych np. *Variable characters* (50) (Tabela 2.2), lub w przypadku, gdy atrybut należy do dziedziny, nazwa tej dziedziny np. *IDClient*. Na samym dole encji został wskazany atrybut o nazwie *IDClient*, który będzie identyfikatorem głównym oraz będzie jednoznacznie identyfikował wystąpienie encji. W encji przy nazwach atrybutów znajdują się oznaczenia, wskazujące na niektóre ich właściwości:

* <pi> – atrybut stanowi element identyfikatora głównego,
* <M> – atrybut wymagany.

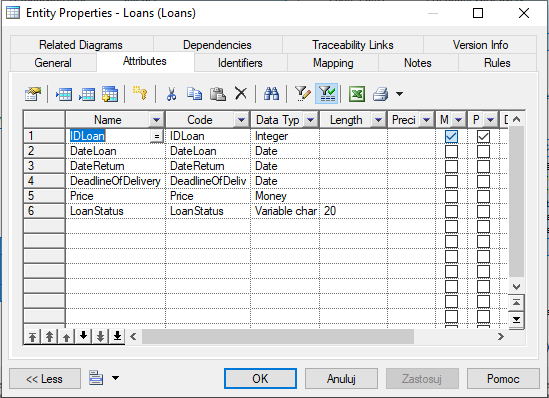


Rysunek 2.16. Symbol encji "Clients" występujący w modelu konceptualnym systemu wypożyczalni płyt DVD.

Źródło: opracowanie własne

**Atrybuty encji**

W zakładce *Attributies* znajduje się lista, w której definiujemy atrybuty encji. Na rysunku 2.17 została przedstawiona lista atrybutów encji, która została zdefiniowana dla encji *Loans* dla naszego diagramu konceptualnego. Atrybuty mogą być dodawane na dwa sposoby: bezpośrednio w liście atrybutów lub przez wybór z listy elementów danych. Pierwszy sposób polega na określeniu w liście własności nowego atrybutu – wypełnienie pustego wiersza tej listy.



Rysunek 2.17. Lista atrybutów encji Loans.

Źródło: opracowanie własne

Drugi sposób polega na przypisaniu do encji jednego bądź kilku elementów danych jako jej nowych atrybutów. W tym przypadku elementy danych muszą zostać wcześniej zdefiniowane, nowe atrybuty przyjmują własności elementów danych. Odpowiednie elementy danych wybieramy w oknie selekcji klikając na przycisk *Add Data Item*. Atrybuty encji możemy podzielić na wymagane oraz opcjonalne. Atrybuty wymagane, to takie, dla których dane muszą zostać wprowadzone – odpowiadające im pola tabeli w bazie danych nie mogą zostać niewypełnione. Natomiast wartość atrybutów opcjonalnych może pozostać nieokreślona (pusta).

W przedstawionej na rysunku 2.17 liście atrybutów encji poza nazwą atrybutu możemy określić następujące jego właściwości:

* *Data type* – typ danych,
* *Domain* – dziedzina,
* *M (mandatory)* – wskazanie atrybutu wymaganego,
* *P (primary identifier)* – wskazanie głównego identyfikatora encji,
* *D (displayed)* – wskazanie atrybutu wyświetlanego na diagramie (domyślnie wszystkie atrybuty). [15]

**Typy danych**

Dla każdego atrybutu należy zdefiniować typ danych w kolumnie *Data Type*. Można to zrobić wybierając jeden ze standardowych typów danych. Jeżeli atrybut został dodany przez wybór spośród dostępnych elementów danych, wówczas typ danych atrybutu odpowiada typowi wybranego elementu. Listę standardowych typów danych otwieramy za pomocą przycisku, który pojawi się po umieszczeniu kursora w komórce kolumny *Data Type*. [15] Zestawienie standardowych typów danych jakie zostały zastosowane w modelu konceptualnym dla bazy danych wypożyczalni płyt zamieszczono w tabeli 2.2. Typy posiadają dwie właściwości: długość (*Length*) – liczba znaków oraz precyzja (*Precision*) – liczba miejsc po przecinku dla danych, które przyjmują wartości dziesiętne. Właściwości, które zostały wymienione nie są dostępne dla wszystkich typów danych. Ponadto, w zależności od typu, długość może oznaczać maksymalną lub wymaganą liczbę znaków.

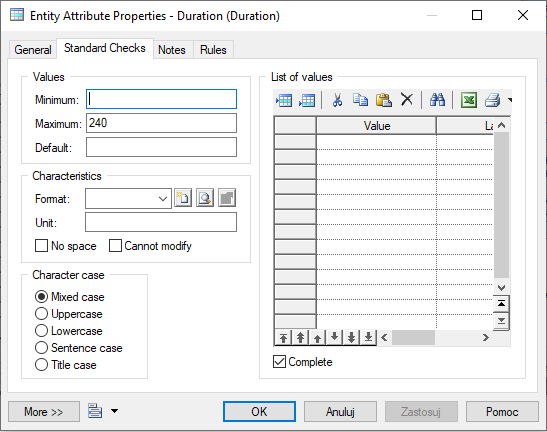
Tabela .2. Standardowe typy danych zastosowane w modelu konceptualnym dla systemu wypożyczalni płyt.

Źródło: opracowanie własne

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nazwa | Symbol | Opis | Długość | Precyzja |
| Integer | I | Liczba całkowita 32 – bitowa | NIE | NIE |
| Money | MN | Liczba rzeczywista o definiowanej precyzji | TAK | TAK |
| Characters | A | Łańcuch znaków | TAK | – |
| Variable characters | VA | Łańcuch znaków | MAX | – |
| Date | D | Data: dzień, miesiąc, rok | NIE | – |

**Parametry sprawdzające**

Parametry sprawdzające (*check parameters*) (PowerDesigner, 2005) stanowią zbiór warunków, które muszą zostać spełnione, aby wprowadzone dane zostały uznane za poprawne. Parametry sprawdzające poprawność danych mogą być definiowane dla pojedynczych atrybutów encji. W modelu konceptualnym mamy do dyspozycji standardową grupę parametrów sprawdzających (*standard check parameters*), które są typowymi ograniczeniami zakresu danych np. wartość minimalna i maksymalna. [15]



Rysunek 2.18. Okno definicji parametrów sprawdzających poprawność danych na przykładzie atrybutu Duration.

Źródło: opracowanie własne

Aby określić standardowe parametry sprawdzające dla atrybutu, należy przejść do okna właściwości atrybutu za pomocą przycisku *Properties*, który znajduje się w pasku narzędzi nad listą atrybutów encji. Na poniższym rysunku 2.18 został pokazany przykład na atrybucie *Duration* znajdujący się w encji *Films*. W tym atrybucie będą przechowywane wartości, informujące użytkownika o czasie trwania wybranego filmu. Na ten atrybut zostało nałożone ograniczenie, gdzie będą przechowywane wartości, dla których czas trwania filmu nie może być dłuższy niż 240 minut. Wartość tę należy zdefiniować w zakładce *Standard Checks*, gdzie w sekcji *Values* pola *Maximum* wprowadzamy wymienioną wcześniej liczbę. Pole *Maximum* określa najwyższą dopuszczalną wartość dla danych numerycznych.

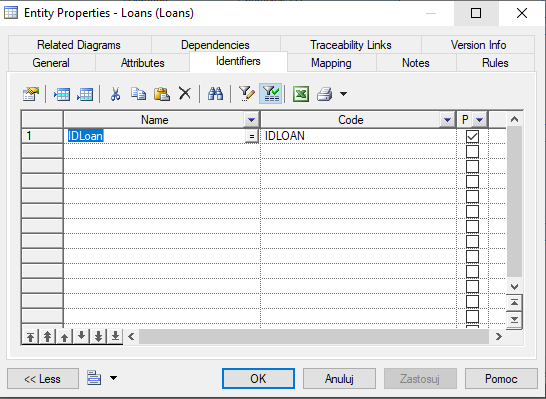
W standardowych parametrach możemy zdefiniować następujące właściwości sprawdzające poprawność danych:

* *Minimum* – najniższa dopuszczalna wartość dla danych numerycznych,
* *Default* – wartość domyślna przypisywana w przypadku braku bezpośrednio wprowadzonych danych,
* *Unit* – jednostka miary,
* *Format* – format danych,
* *Lowercase* – automatyczna zmiana na małe litery,
* *Uppercase* – automatyczna zmiana na duże litery,
* *Cannot modify* – blokada możliwości zmiany danych,
* *List of values* – lista dopuszczalnych wartości,
* *Label* – etykieta, która identyfikuje wartość dopuszczalną. [15]

**Identyfikatory**

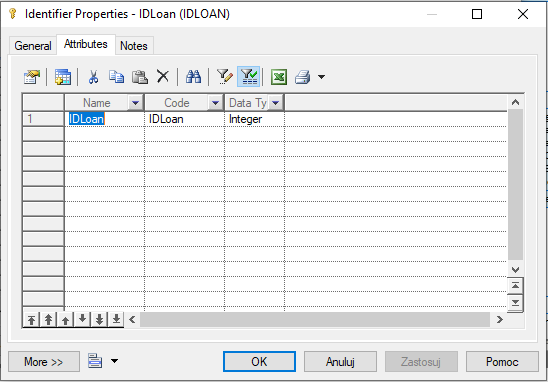
Każda encja w modelu konceptualnym musi być identyfikowana przez kombinację atrybutów (identyfikator) lub przez związki z innymi encjami (związki zależne) (PowerDesigner, 2005). Jeżeli encja posiada jeden identyfikator, to domyślnie jest on identyfikatorem głównym (primary identifier). Jeżeli identyfikatorów jest więcej, to jeden z nich stanowi identyfikator główny, a pozostałe są identyfikatorami alternatywnymi. Ze zbioru atrybutów danej encji wybiera się jeden lub więcej atrybutów jako identyfikator encji. Identyfikator jest atrybutem (lub grupą atrybutów), który jednoznacznie identyfikuje wystąpienie (instancję) encji. Oznacza to, że wartości identyfikatora nie mogą być takie same, dla różnych wystąpień encji. Ponadto atrybut lub atrybuty tworzące identyfikator muszą być atrybutami wymaganymi, ponieważ wartość identyfikatora musi być określona dla każdego wystąpienia encji. [15]

Identyfikatory definiujemy w zakładce *Identifiers* w oknie właściwości encji (Rysunek 2.19). Na poniższym rysunku została przedstawiona lista identyfikatorów, jakie zostały zdefiniowane dla encji *Loans*. Dla tej encji został zdefiniowany jeden identyfikator *IDLoan*, który będzie definiował numer identyfikacyjny wypożyczenia dla wybranego filmu stosowany w danym przedsiębiorstwie. Po określeniu w kolumnie *Name* nazwy identyfikatora przechodzimy do okna właściwości identyfikatora za pomocą przycisku *Properties* (Rysunek 2.20). W oknie właściwości na zakładce *Attributies* wskazujemy atrybuty tworzące identyfikator, wybieramy je posługując się listą, która otwiera przycisk *Add Attributies* z paska narzędzi. W kolumnie oznaczonej literą *P* na liście identyfikatorów wskazujemy główny identyfikator encji.



Rysunek 2.19. Lista identyfikatorów encji Loans w modelu konceptualnym.

Źródło: opracowanie własne



Rysunek 2.20. Okno właściwości identyfikatora IDLoan.

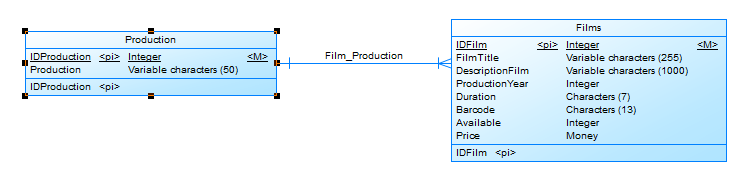
Źródło: opracowanie własne

Mniej skomplikowana metoda definiowania identyfikatora głównego encji polega na zaznaczeniu atrybutów, które mają tworzyć identyfikator główny w kolumnie *P* na liście atrybutów encji (Rysunek 2.17).

#### 2.4.5.2. Związki

Związek (PowerDesigner, 2005) jest połączeniem dwóch encji (niekoniecznie różnych), określającym wzajemne zależności występujące pomiędzy encjami. [15] Rozważmy przykład modelu konceptualnego dla systemu wypożyczalni płyt DVD, w którym istnieje związek pomiędzy encjami *Production* oraz *Films* (Rysunek 2.21), ponieważ film musi zawierać informację o tym, w jakim kraju film został wyprodukowany. Związek ten informuje, że kraj produkcji może być przypisany do wielu filmów, natomiast film musi być przypisany tylko do jednej konkretnej produkcji.

Definiowanie związku w modelu konceptualnym jest realizowane na diagramie za pomocą odpowiedniego narzędzia z opcji *Concpetual Diagrams* w sekcji *Toolbox* lub w liście związków, która otwiera polecenie *Model | Relationships* z paska menu. Aby zdefiniować związek, przede wszystkim należy wskazać encje, które mają w tym związku uczestniczyć. Korzystając z narzędzi graficznych na diagramie, wskazujemy pierwszą encję klikając w obszarze jej symbolu, następnie przeciągamy wskaźnik myszy do symbolu drugiej encji uczestniczącej w związku i zwalniamy przycisk myszy. Natomiast na liście związków definiujemy encje uczestniczące podając ich nazwy w kolumnach *Entity 1* oraz *Entity 2*.

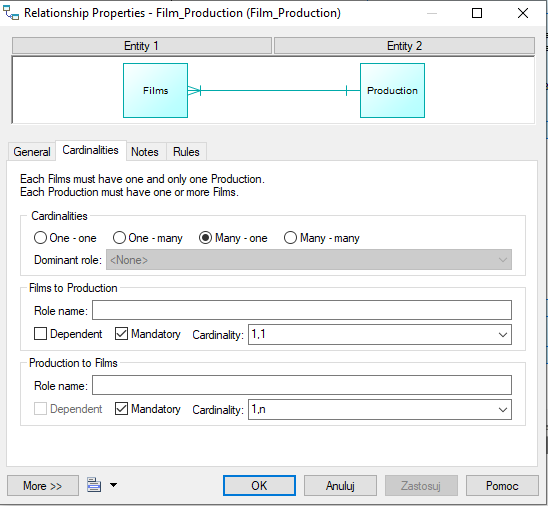


Rysunek 2.21. Symbol związku encji Production -> Films w systemie wypożyczalni płyt DVD.

Źródło: opracowanie własne

W oknie właściwości związku, na zakładce *Detail* (Rysunek 2.22) należy wybrać typ związku. Do wyboru mamy cztery następujące typy:

* Jeden do jednego (*one – one*) – jedno wystąpienie encji pierwszej może odpowiadać tylko jednemu wystąpieniu encji drugiej,
* Jeden do wielu (*one – many*) – jedno wystąpienie encji pierwszej może odpowiadać więcej niż jednemu wystąpieniu encji drugiej,
* Wiele do jednego (*many – one*) – więcej niż jedno wystąpienie encji pierwszej może odpowiadać jednemu (temu samemu) wystąpieniu encji drugiej,
* Wiele do wielu (*many – many*) – więcej niż jedno wystąpienie encji pierwszej może odpowiadać więcej niż jednemu wystąpieniu encji drugiej. [15]



Rysunek 2.22. Okno właściwości związku Films->Production.

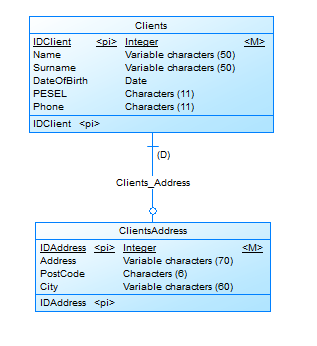
Źródło: opracowanie własne

Jak możemy zauważyć na powyższych rysunkach (Rysunek 2.21 i 2.22) dla przykładowego związku uczestnictwo encji *Production* i *Films* jest wymagane. Oznacza to, że produkcja musi zostać przypisana do filmu. Krotność „1, n” oznacza, że z wystąpieniem encji *Production* może być powiązane jeden bądź wiele wystąpień encji *Films*. Natomiast krotność „1, 1” przy wystąpieniu encji *Films*, będzie powiązane tylko jedno wystąpienie encji *Production*. Innymi słowy, produkcja musi być przypisana do jednego lub więcej filmów. Natomiast film musi zostać przypisany tylko do jednej produkcji. Należy zauważyć, że minimalna krotność jest związana z wymagalnością: jeżeli uczestnictwo encji w związku jest wymagane, to minimalna krotność wynosi jeden, natomiast gdy uczestnictwo encji w związku jest opcjonalne, wówczas minimalna krotność jest równa zeru.

W modelu konceptualnym nie zaleca się stosowania związków wymaganych w obydwu kierunkach. W przypadku tego typu związków utworzenie nowego wystąpienia jednej encji wymaga jednoczesnego utworzenia związanego z nim wystąpienia drugiej encji. Taka zależność może spowodować praktyczne problemy przy wprowadzaniu danych do bazy danych. [15]

**Związki z rolą dominującą**

W przypadku definiowania związków typu jeden do jednego istnieje możliwość określenie roli dominującej (*Dominant role*). Odpowiednia opcja znajduje się w oknie własności związku (Rysunek 2.22). Domyślnie rola dominująca jest nieokreślona (*None*), co oznacza, że dla wystąpień obydwu encji uczestniczących w związku dostępna jest informacja wskazująca wystąpienia encji z nimi powiązanych. W przeciwnym wypadku, gdy zdefiniujemy rolę dominującą, wówczas tylko dla wystąpień jednej z encji uczestniczących w związku określone będą wskazania na powiązane z nimi wystąpienia encji drugiej.



Rysunek 2.23. Związek encji z rolą dominującą Clients-> ClientsAddress.

Źródło: opracowanie własne

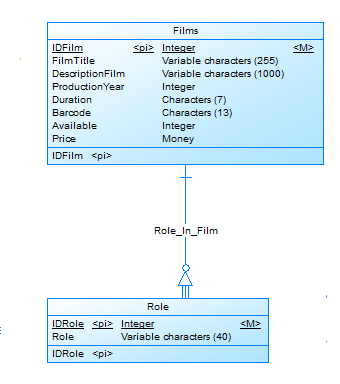
Na rysunku 2.23 przedstawiono przykładowy związek encji z rolą dominującą *Clients -> ClientsAddress*. Rolę dominującą wskazuje symbol (D) na diagramie. W tym przykładzie informacja o związanym wystąpieniu encji będzie dostępna tylko dla encji *ClientsAddress*. Zatem budując model możemy założyć, że każdy adres, który jest zapisany w bazie danych musi posiadać informację o kliencie, który mieszka pod danym adresem. Natomiast klient nie musi potrzebować wskazania na adres, pod którym mieszka. Dla przeciwnie skierowanej roli dominującej (*ClientsAddress -> Clients*) symbol (D) na diagramie znajdzie się po drugiej stronie związku przy encji *ClientsAddress*. W takim przypadku dla każdego wystąpienia encji *Clients* będzie dostępne wskazanie na związane z nim wystąpienie encji *ClientsAddress*, a wpis dotyczący adresu klienta nie będzie zawierał informacji umożliwiającej jego dostęp do danych.

Gdy się zdarzy sytuacja, że rola dominująca nie jest określona (*None*), wówczas wśród danych dotyczących adresu, znajdzie się informacja wskazująca na danego klienta, a także dane adresowe będą zawierały informację o kliencie, mieszkającym pod podanym adresem. [15]

**Związki zależne**

W związku zależnym wystąpienie jednej encji jest identyfikowane przez wystąpienie drugiej encji. Każda encja modelu konceptualnego musi posiadać identyfikator, jednak w pewnych przypadkach atrybuty danej encji nie wystarczają, aby jednoznacznie zidentyfikować jej wystąpienia. W takiej sytuacji identyfikator tej encji musi zawierać identyfikator pochodzący z innej encji, z którą występuje związek zależny. Związki zależne mogą posiadać typ jeden do jednego lub jeden do wielu (wiele do jednego). Związki zależne typu wiele do wielu nie występują. [15]

Przykład związku zależnego został przedstawiony na rysunku 2.24. Na poniższym przykładzie przyjęto założenie, że każdy film może posiadać kilka ról, granych przez jednego aktora. Role o takiej samej nazwie mogą być grane w różnych filmach. Aby zidentyfikować wystąpienie encji *Role*, należy użyć identyfikatora głównego z atrybutu *IDRole* encji *Role* oraz z atrybutu *IDFilm* z encji *Films*.

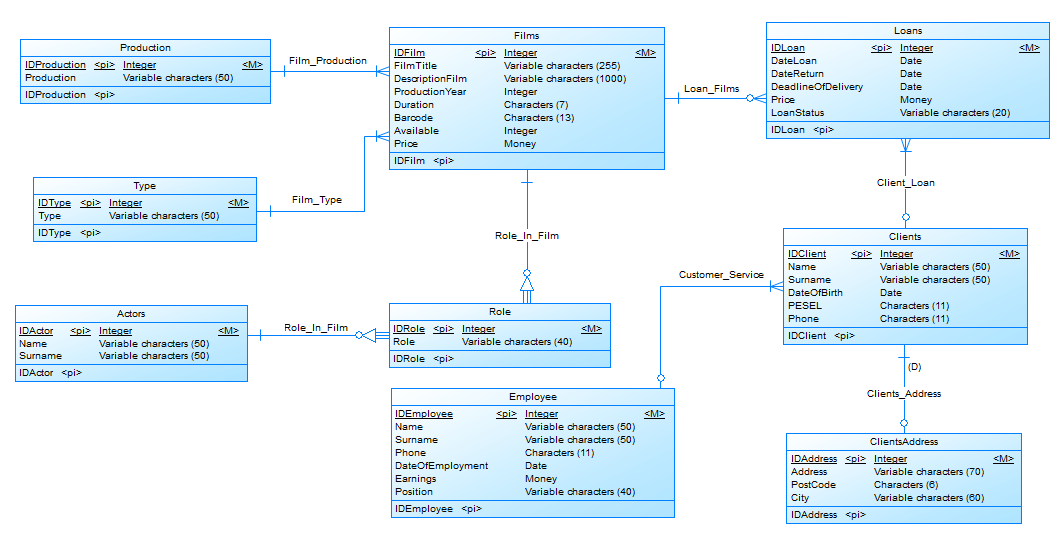


Rysunek 2.24. Związek zależny Films -> Role w systemie wypożyczalni płyt DVD.

Źródło: opracowanie własne

#### 2.4.5.3. Model konceptualny dla systemu wypożyczalni płyt DVD

Jeżeli zakończyliśmy pracę nad modelem konceptualnym, to możemy bez żadnego problemu przejść do krótkiego omówienia utworzonego diagramu. Model konceptualny dla systemu wypożyczalni płyt DVD został opracowany uwzględniając dane, które określiliśmy w wymaganiach funkcjonalnych. Model konceptualny został wygenerowany na podstawie modelu klas UML (punkt 2.4.4). Kolejnym krokiem jest zidentyfikowanie encji i przypisanie do nich atrybutów, czyli odpowiednich elementów danych (punkt 2.4.5.1). Odpowiedź na pytanie, jakie encje są potrzebne, wynika z zakresu danych, jakie zostały uwzględnione w projekcie.



Rysunek 2.25. Diagram konceptualny dla systemu wypożyczalni płyt DVD.

Źródło: opracowanie własne

Należy pamiętać, że każda encja może posiadać tylko takie atrybuty, które bezpośrednio jej dotyczą. Nie można umieszczać w jednej encji atrybutów opisujących różne obiekty. Na przykład, dane *PESEL* i *PostCode* muszą zostać umieszczone jako atrybuty w dwóch odrębnych encjach, ponieważ dotyczą dwóch różnych obiektów – klienta i adresu klienta. Kierując się tą zasadą oraz biorąc pod uwagę zbiór danych, które określiliśmy w wymaganiach funkcjonalnych zdefiniowano encje, które zostały przedstawione w modelu konceptualnym (Rysunek 2.25).

Encje zostały połączone za pomocą związków zgodnie z założeniami, które zestawiono w tabeli 2.3. W modelu zostały wprowadzone dwa związki zależne łączące encję *Role* z encjami *Actors* i *Films*. Identyczne przykłady zależnych związków encji zostały omówione w punkcie 2.4.5.2.

Tabela 2.. Związki w modelu związków encji dla systemu wypożyczalni płyt DVD.

Źródło: opracowanie własne

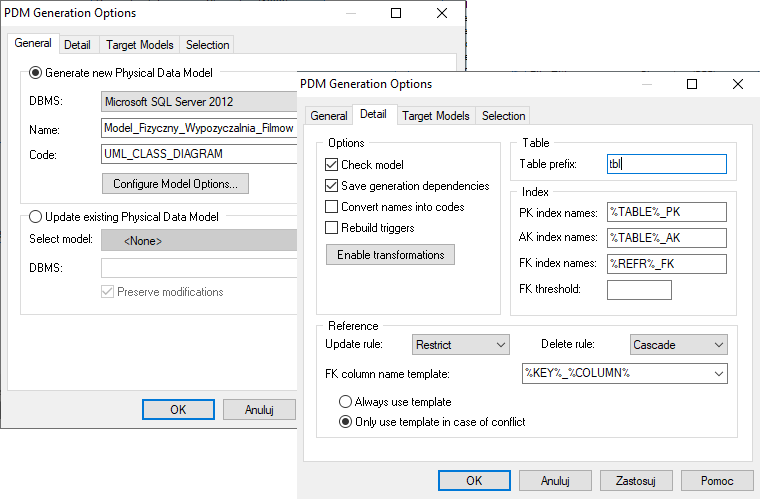
|  |  |
| --- | --- |
| **Związek** | **Opis** |
| Production – Films | Produkcja musi być przypisana do wielu filmów. Film musi być przypisany tylko do jednej produkcji. |
| Type – Films | Gatunek musi być przypisany do wielu filmów. Film musi być przypisany tylko do jednego gatunku. |
| Films – Loans | Film może być wypożyczony wiele razy. Film wypożyczony musi dotyczyć tylko jednego filmu. |
| Loans – Clients | Wypożyczony film może być przypisany tylko do jednego klienta. Klient może wypożyczyć wiele filmów. |
| Employee – Clients | Pracownik musi obsłużyć wielu klientów. Klient może zostać obsłużony tylko przez jednego pracownika. |
| Clients – ClientsAddress | Klient może posiadać tylko jeden adres. Adres musi zostać przypisany tylko do jednego klienta. |
| Actors – Role | Aktor może grać wiele ról w filmie. Rola musi zostać zagrana tylko przez jednego aktora |
| Role – Films | Rola musi być przypisana tylko do jednego filmu. Film może zostać przypisany do wielu ról. |

#### 2.4.5.4. Generowanie modelu fizycznego

Następnie po krótkim omówieniu diagramu konceptualnego musimy przekształcić utworzony diagram na model fizyczny. Operacja generowania modelu fizycznego oznacza automatyczną transformację obiektów CDM na odpowiednie obiekty w PDM. Należy podkreślić, że model fizyczny jest generowany dla konkretnego SZBD.

Aby wygenerować model fizyczny, należy wybrać polecenie *Tools | Generate Physical Data Model*. Otwarte zostanie okno opcji generowania PDM (Rysunek 2.26). Za pomocą odpowiednich pól wybieramy generowanie nowego modelu fizycznego (*Generate new Physical Data Model*) lub aktualizację istniejącego (*Update existing Physical Data Model*). W polu *DBMS* musimy wybrać docelowy system zarządzania bazą danych. Nazwę nowego modelu wpisujemy w polu *Name*.

Na zakładkach okna opcji generowania modelu fizycznego możemy znaleźć wiele innych ustawień, gdzie możemy wymienić kilka z nich. Zakładka *Detail* zawiera m.in. pole wyboru *Check model*, za pomocą którego określamy, czy model konceptualny ma zostać sprawdzony przed wygenerowaniem PDM. Pole *Table prefix* pozwala na użycie przedrostka dla nazw tabel wygenerowanych z podmiotów w modelu CDM. Na zakładce *Selection* możemy wybrać obiekty, które zostaną uwzględnione w trakcie generowania modelu fizycznego. Domyślnie wybrane są wszystkie obiekty. [15]



Rysunek 2.26. Okno opcji generowania modelu fizycznego.

Źródło: opracowanie własne

Szczegółowy opis pozostałych opcji generowania modelu fizycznego możemy znaleźć w dokumentacji programu PowerDesigner. Jeżeli ustawienia zostaną zatwierdzone, to zostanie uruchomiony proces sprawdzania modelu oraz wygenerowanie PDM. W sytuacji, gdy model związków encji, będzie zawierał błędy, to proces zostanie przerwany i pojawi się odpowiedni komunikat. W przeciwnym wypadku operacja zakończy się powodzeniem i wówczas otwarty zostanie diagram PDM. Zasady transformacji obiektów CDM na obiekty PDM zostaną omówione w kolejnym podrozdziale, który będzie dotyczył modelu fizycznego.

### 2.4.6. Model fizyczny

Model fizyczny PDM pozwala na zdefiniowanie struktury bazy danych z uwzględnieniem szczegółów jej implementacji. Szczegóły te dotyczą m.in. wyboru docelowego SZBD oraz organizacji zapisu danych w pamięci. Opracowanie modelu fizycznego jest ostatnim etapem procesu projektowania bazy danych. Celem tego etapu jest określenie sposobu implementacji obiektów modelu konceptualnego. Na podstawie gotowego modelu fizycznego możemy wygenerować skrypt SQL, który utworzy w bazie danych wszystkie elementy, jakie umieściliśmy w projekcie. Rezultatem będzie baza danych zaimplementowana zgodnie ze schematem opracowanym w postaci diagramu PDM.

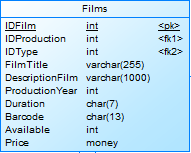
W PowerDesignerze istnieje możliwość budowy modelu fizycznego od postaw, z pominięciem etapu projektowania konceptualnego. Jednak korzystniejszym i prostszym rozwiązaniem jest zastosowanie funkcji automatycznego generowania PDM na podstawie opracowanego wcześniej modelu konceptualnego (punkt 2.4.5.4.). Utworzenie nowego, pustego modelu jest realizowane za pomocą polecenie *File | New* z menu głównego. W oknie, które zostanie otwarte, należy wybrać w odpowiednich polach rodzaj modelu (*Physical Data Model*), docelowy SZBD oraz typ diagramu (*Physical Diagram*). Po zatwierdzeniu wyboru zostanie otwarte puste okno diagramu oraz paleta narzędzi modelu fizycznego [14].

W dalszej części podrozdziału omówiono poszczególne typy obiektów, jakie zostały zastosowane w modelu fizycznym dla tworzonego systemu oraz podano zasady transformacji CDM na PDM. Przedstawiono również ostateczną wersję modelu fizycznego, który został opracowany dla systemu wypożyczalni filmów.

#### 2.4.6.1. Tabele

Tabele (PowerDesigner, 2004) to struktury składające się z wierszy i kolumn, które są wykorzystywane do przechowywania danych w relacyjnej bazie danych. W modelu fizycznym PDM tabele mogą zawierać następujące elementy:

* Kolumny (*columns*), z których każda gromadzi dane jednego typu i reprezentuje jedną właściwość obiektów opisywanych w wierszach tabeli,
* Indeksy (*indexes)*, czyli struktury umożliwiające uporządkowanie (posortowanie) wartości znajdujących się w kolumnach tabeli,
* Klucze (*keys*), czyli pojedyncze kolumny lub zbiory kolumn pozwalające na jednoznaczne identyfikowanie wierszy tabeli,
* Wyzwalacze (*triggers*), czyli zestawy poleceń SQL, które będą uruchamiane w momencie wykonywania określonych operacji na danych w tabeli. [14]



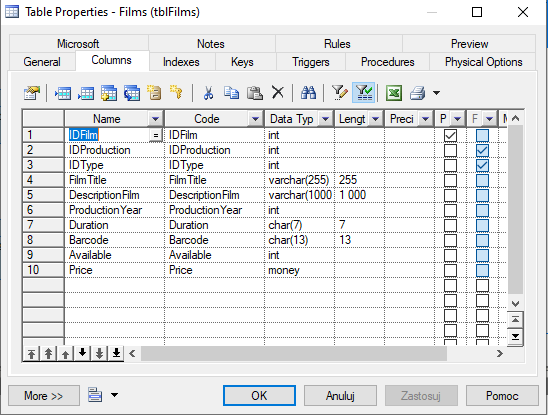
Rysunek 2.27. Symbol tabeli w modelu fizycznym PDM, na przykładzie tabeli Films.

Źródło: opracowanie własne

Na rysunku 2.27 przedstawiono symbol tabeli stosowany w diagramach PDM, na przykładzie tabeli *Films*. W górnej części tego symbolu umieszczona jest nazwa tabeli, poniżej znajduje się lista kolumn z ich nazwami oraz oznaczonymi typami danych. Skrót „<pk>” wskazuje na kolumnę bądź kolumny klucza głównego (*Primary key*). Natomiast skrót o nazwie „<fk>” wskazuje na klucz obcy.

Wszystkie właściwości tabeli są dostępne w oknie, które zostało przedstawione na rysunku 2.28. W zakładce *General* możemy zdefiniować m.in. nazwę tabeli (*Name*), komentarz stanowiący opis tabeli (*Comment*) oraz szacunkową liczbę wierszy (*Number*). Ponadto, za pomocą pola wyboru *Generate* możemy określić, czy tabela zostanie utworzona podczas generowania bazy danych. Domyślnie pole *Generate* jest zaznaczone dla każdej tabeli. Właściwość *Number* jest wykorzystywana w obliczeniach dotyczących rozmiarów pamięci potrzebnej dla projektowanej bazy danych.

Lista kolumn tabeli znajduje się w oknie właściwości na zakładce *Columns*. Za pomocą narzędzi dostępnych na tej zakładce możemy edytować nazwy kolumn (*Name*) i wybrać dla nich typ danych. W polach „P” zaznaczone są kolumny tworzące klucz główny tabeli. Pola wyboru „F” wskazują na klucze obce. Natomiast pole oznaczone literą „M” wskazuje na kolumny wymagane, czyli takie, dla których dane muszą zostać wprowadzone w każdym wierszu tabeli. Oznacza to, że w kolumnach wymaganych nie mogą występować wartości „puste” (*null*).



Rysunek 2.28. Okno właściwości tabeli "Films" w modelu fizycznym PDM.

Źródło: opracowanie własne

Na kolejnych zakładkach znajdują się listy obiektów, które zostały zdefiniowane dla rozpatrywanej tabeli lub są z nią powiązane. Obiekty, o których mowa, to indeksy (zakładka *Indexes*), klucze (zakładka *Keys*), wyzwalacze (zakładka *Triggers*) oraz procedury (zakładka *Procedures*). Zakładka *Preview* zawiera podgląd poleceń SQL, które w trakcie generowania bazy danych utworzą zaprojektowaną tabelę wraz z towarzyszącymi obiektami. [14]

**Kolumny**

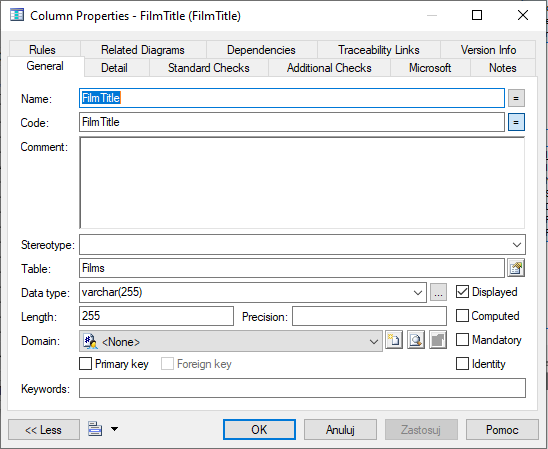
Jak wspomniano wyżej, lista kolumn tabeli jest dostępna w oknie jej właściwości (Rysunek 2.28). Lista zawiera podstawowe parametry kolumn oraz narzędzia umożliwiające dodawanie nowych kolumn do tabeli (*Add a Row*), utworzenie kopii (*Add a Column*), oraz repliki istniejącej kolumny (*Replicate Columns*). Replika posiada identyczne właściwości jak kolumna oryginalna. Kopiując kolumnę tworzymy nową, niezależną kolumnę, która posiada właściwości zdefiniowane tak samo jak kolumna oryginalna.

Wyświetlenie okna umożliwiającego edycję szczegółów definicji wybranej kolumny jest możliwe za pomocą przycisku *Properties*. W oknie właściwości kolumny (Rysunek 2.29) dostępne są m.in. następujące parametry:

* *Name* – nazwa kolumny,
* *Table* – nazwa tabeli, w której znajduje się kolumna,
* *Data type* – typ danych, przechowywanych w kolumnie,
* *Length* – długość, czyli maksymalna liczba znaków,
* *Precision* – precyzja, czyli maksymalna liczba cyfr po przecinku,
* *Identity* – opcja, która oznacza automatyczne wstawianie w kolumnie numerów wierszy,
* *Computed* – opcja, która oznacza, że kolumna jest kolumną wyliczaną,
* *Primary key* – opcja, która oznacza, że kolumna należy do klucza głównego,
* *Foreign key* – opcja, która oznacza, że kolumna należy do klucza obcego,
* *Mandatory* – opcja, która oznacza, że w kolumnie nie mogą się znajdować wartości null (pola puste).

Część z wymienionych powyżej parametrów jest dostępna tylko dla niektórych SZBD.

Typ danych dla kolumny możemy zdefiniować w sposób bezpośredni lub pośredni, przypisując kolumnę do odpowiedniej dziedziny. W naszym modelu typ danych został określony w sposób bezpośredni. Oznacza to, że musimy dokonać odpowiedniego wyboru w polu *Data type* lub skorzystać z listy standardowych typów danych, która otwierana jest za pomocą przycisku z symbolem znaku zapytania. Definicję typu danych możemy uzupełnić o parametry sprawdzające, które ograniczają zakres dopuszczalnych wartości. Parametry sprawdzające możemy określić dla dziedziny lub kolumny w oknie właściwości tych obiektów, na zakładce *Standard Checks*. Dostępne parametry obejmują wartość maksymalną, minimalną oraz listę dopuszczalnych wartości (punkt 2.4.5.1).



Rysunek 2.29. Okno właściwości kolumny "FilmTitle" w tabeli "Films".

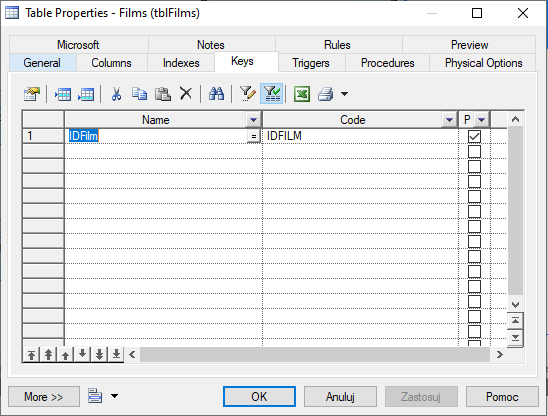
Źródło: opracowanie własne

Wartość domyślna jest zapisywana automatycznie w danej kolumnie, jeżeli użytkownik nie określił wartości dla tej kolumny podczas wstawiania nowego wiersza. Wartość domyślną definiujemy na zakładce *Standard Checks*, w polu *Default*. Możemy w tym polu wpisać wartość stałą lub wybrać jedną z funkcji dostępnych na liście rozwijalnej. Na przykład, jeżeli chcemy, aby w projektowanej kolumnie automatycznie była wstawiana bieżąca data, to wybieramy z listy rozwijanej funkcję *„current date”*. Przydatną funkcją może być również *„current time”* wyświetlająca aktualny czas. [14]

**Klucze**

Każdy wiersz w tabeli musi być unikatowy, czyli musi się różnić od pozostałych wierszy. To oznacza, że w tabeli nie mogą istnieć dwa takie wiersze, które będą posiadały identyczne wartości we wszystkich kolumnach. Zatem, każda tabela musi posiadać przynajmniej jeden klucz, czyli kolumnę lub zbiór kolumn, w których wartości się nie powtórzą. Klucz w tabeli może tworzyć jedna kolumna (np. *IDFilm* w tabeli *Films*) lub wiele kolumn (np. *IDActor* i *IDFilm* w tabeli *Role*). W pierwszym przypadku mówimy o kluczu prostym, a w drugim przypadku o kluczu złożonym.

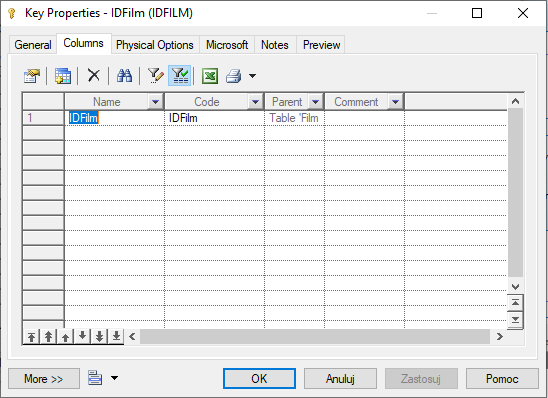
W każdej tabeli może istnieć wiele kluczy. Ze zbioru kluczy w danej tabeli wybieramy jeden klucz główny. Pozostałe klucze są kluczami alternatywnymi. Oprócz klucza głównego i kluczy alternatywnych mogą występować klucze obce. Klucz obcy jest kolumną (lub zbiorem kolumn), która jest skopiowana z innej tabeli, w której pełni ona rolę klucza głównego lub alternatywnego. Klucze obce są stosowane w celu umożliwienia zapisu powiązań, jakie występują pomiędzy wierszami w różnych tabelach.



Rysunek 2.30. Lista kluczy tabeli "Films".

Źródło: opracowanie własne

Najprostszą metodą utworzenia klucza głównego dla tabeli jest zaznaczenie w oknie właściwości danej tabeli, na zakładce *Columns* (Rysunek 2.28) pól wyboru „P” dla tych kolumn, które mają tworzyć klucz główny. Zarówno klucze alternatywne, jak i klucz główny mogą być definiowane w oknie właściwości tabeli, w zakładce *Keys* (Rysunek 2.30). Za pomocą pól wyboru oznaczonych literą „P” wybieramy klucz główny tabeli. Nowy klucz możemy utworzyć używając przycisku *Add a Row*. Właściwości klucza należy zdefiniować w oknie, które otwiera przycisk *Properties* (Rysunek 2.31). Na zakładce *Columns* tego okna określamy, jakie kolumny tworzą dany klucz. Kolumny można dodać do listy z użyciem przycisku *Add Columns*. [14]



Rysunek 2.31. Okno właściwości klucza "IDFilm" w tabeli Films".

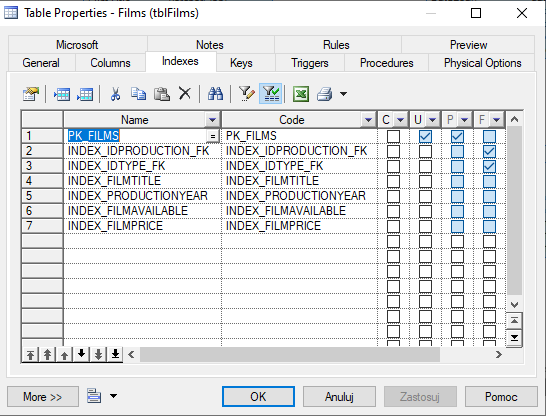
Źródło: opracowanie własne

**Indeksy**

Indeks (PowerDesigner, 2004) jest dodatkową strukturą fizyczną, która jest powiązana z tabelą i umożliwia przyspieszenie operacji sortowania i wyszukiwania danych w tej tabeli. Indeksy umożliwiają uporządkowanie (posortowanie) wierszy tabeli według wybranej kolumny. Są one definiowane dla tych kolumn, według których będziemy często przeszukiwali tabelę. Najbardziej efektywnym rozwiązaniem są indeksy stosowane dla kolumn, które zawierają unikalne wartości.

Podczas generowania modelu fizycznego z modelu konceptualnego indeksy są tworzone automatycznie dla wszystkich kluczy głównych, obcych i alternatywnych. Jeżeli modyfikujemy diagram PDM lub tworzymy go od podstaw, wówczas indeksy dla wszystkich kluczy możemy w prosty sposób utworzyć lub uaktualnić za pomocą polecenia *Tools | Rebulid objects | Rebulid indexes*. [14]

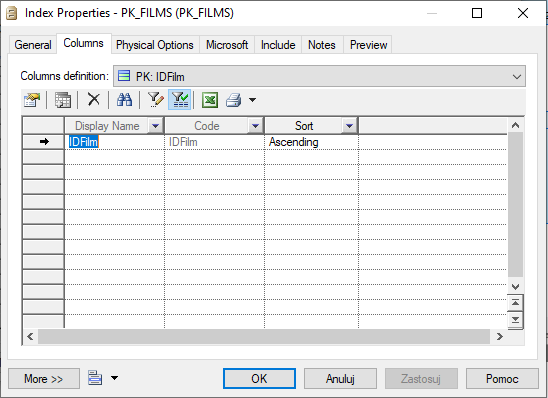
Na poniższym rysunku 2.32 została zdefiniowana lista indeksów jakie się znajdą w tabeli *Films*. Aby wejść do listy indeksów po kliknięciu na wybraną tabelę we właściwościach tabeli należy przejść do zakładki *Indexes*. Jeżeli chcemy dodać nowy indeks, należy użyć przycisku *Add a Row* i następnie wyświetlić okno właściwości nowego indeksu (przycisk *Properties*). W zakładce *General* określamy nazwę indeksu i zaznaczamy pole *Unique*, jeżeli wartości indeksowane nie mogą się powtarzać.



Rysunek 2.32. Lista indeksów zdefiniowanych w tabeli "Films".

Źródło: opracowanie własne

Następnym krokiem jest określenie kolumny lub zbioru kolumn, dla którego ma być utworzony indeks. W tym celu należy przejść na zakładkę *Columns* w oknie właściwości indeksu (Rysunek 2.33) i wybrać odpowiednią kolumnę lub kolumny za pomocą przycisku (*Add Columns*). Jeżeli chcemy utworzyć indeks dla klucza tabeli, wówczas odpowiedni klucz należy wybrać z listy dostępnej w polu *Columns definition*. W polu *Sort* określamy sposób sortowania wartości indeksu: opcja *Ascending* oznacza sortowanie w kolejności rosnącej, natomiast *Descending* odpowiada kolejności malejącej.



Rysunek 2.33. Okno właściwości indeksu PK\_FILMS dla tabeli Films, lista kolumn.

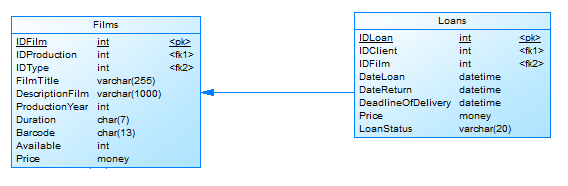
Źródło: opracowanie własne

#### 2.4.6.2. Referencje

Referencja (*Reference*) (PowerDesigner, 2004) jest obiektem modelu fizycznego PDM, który reprezentuje zależności występujące pomiędzy tabelą nadrzędną (rodzicem) oraz tabelą podrzędną (potomkiem). Każda referencja zawiera co najmniej jedno powiązanie kolumn (*Join*). Powiązanie zawsze dotyczy pary kolumn, przy czym jedna z tych kolumn należy do tabeli nadrzędnej, a druga do tabeli podrzędnej.

Przykład referencji, jaki jest użyty w naszym modelu fizycznym został przedstawiony na rysunku 2.34. Symbolem referencji w diagramie PDM jest strzałka, której grot wskazuje tabelę nadrzędną. W przedstawionym przykładzie referencja zawiera powiązanie kolumny klucza głównego *IDFilm* z tabeli *Films* z kolumną klucza obcego *IDFilm* w tabeli *Loans*.

Każdej referencji odpowiada ograniczenie, które dotyczy wartości występujących w powiązanych kolumnach. Dla analizowanego przykładu (Rysunek 2.34) ograniczenie można intepretować następująco: każdy wypożyczony film przechowywany w tabeli podrzędnej musi dotyczyć filmu, który w rzeczywistości istnieje, czyli jest zapisany w tabeli nadrzędnej. Przekładając ten warunek na wartości występujące w powiązanych kolumnach należy stwierdzić, że w kolumnie *IDFilm* w tabeli *Loans* dopuszczalne są tylko takie wartości identyfikatora filmu, który występuje w kolumnie *IDFilm* tabeli *Films*. W zaimplementowanej bazie danych kontrola spełnienia tego warunku i niedopuszczenie do zapisu w związanych kolumnach niepoprawnych danych będzie realizowane przez SZBD.

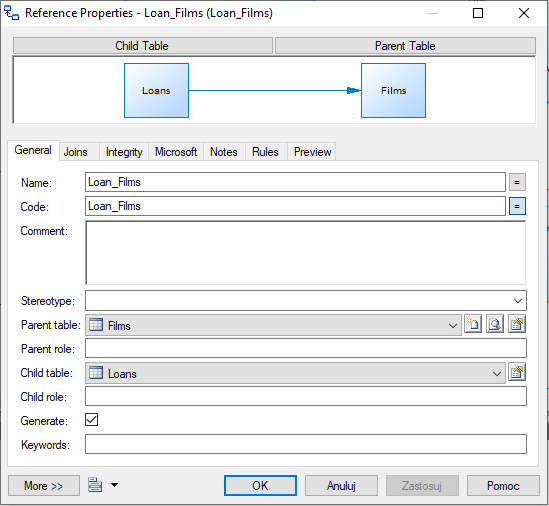


Rysunek 2.34. Referencja zachodząca pomiędzy tabelami "Loans" i "Films".

Źródło: opracowanie własne

Referencję możemy utworzyć w prosty sposób przez narysowanie jej na diagramie. W tym celu należy użyć odpowiedniego narzędzia, które jest dostępne na pasku palety w sekcji *Toolbox* w programie PowerDesigner po naciśnięciu przycisku *Physical Diagram*. Utworzenie referencji powoduje automatyczne wstawienie klucza obcego i powiązanie odpowiednich kolumn. To automatyczne wspomaganie definiowania referencji jest realizowane przez PowerDesignera, jeżeli zaznaczone są właściwe pola wyboru w opcjach modelu.

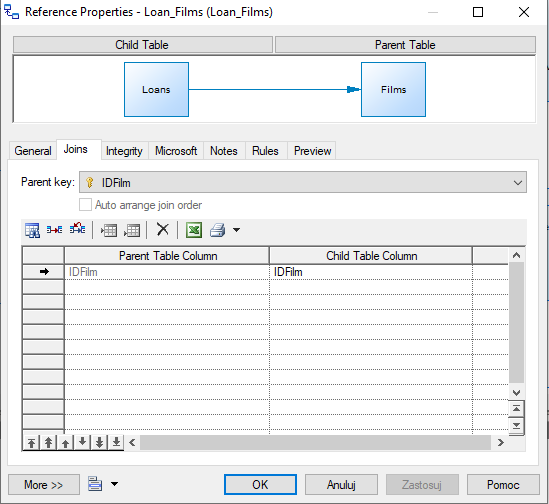
Wszystkie szczegóły dotyczące referencji możemy przeglądać i edytować w oknie właściwości referencji. Na poniższym rysunku 2.35 zostały przedstawione właściwości referencji *Loans\_Films* jaka zachodzi pomiędzy tabelą *Loans*, a tabelą *Films*. Podstawowe właściwości referencji obejmują jej nazwę (*Name*), nazwę tabeli nadrzędnej (*Parent table*) oraz nazwę tabeli podrzędnej (*Child table*). Role to opisy, które mogą być wyświetlane na diagramie w celu ułatwienia interpretacji referencji (Rysunek 2.34). Podobne zastosowanie mają role dla związków w diagramie CDM (punkt 2.4.5).



Rysunek 2.35. Okno właściwości referencji Loans\_Films.

Źródło: opracowanie własne

Wykaz powiązań kolumn zdefiniowanych dla referencji jest dostępny w oknie właściwości na zakładce *Joins* (Rysunek 2.36). Każde powiązanie dotyczy pary kolumn, z których jedna należy do tabeli nadrzędnej (*Parent Table Column*), a druga do tabeli podrzędnej (*Child Table Column*). W ramach referencji możemy zdefiniować powiązanie dla kolumn tworzących klucz główny lub alternatywny tabeli nadrzędnej. W tym celu należy wybrać nazwę klucza z listy dostępnej w polu *Parent key*. Istnieje również możliwość definiowania powiązań dla dowolnych kolumn, które niekoniecznie muszą należeć do klucza tabeli nadrzędnej. W takim przypadku należy w polu *Parent key* wybrać opcję *None*. Podczas definiowania powiązań kolumn mamy do dyspozycji narzędzia, które mogą ułatwić nam pracę. Przycisk *Reuse Columns* umożliwia automatyczne powiązanie wybranych kolumn tabeli nadrzędnej z kolumnami o identycznych nazwach tabeli podrzędnej (jeżeli takie istnieją). Przycisk *Migrate Columns* kopiuje wybrane kolumny tabeli nadrzędnej do tabeli podrzędnej i tworzy odpowiednie powiązania. Wycofanie operacji kopiowania i wiązania kolumn jest możliwe za pomocą przycisku *Cancel Migration*.



Rysunek 2.36. Okno właściwości referencji "Loans\_Films, lista powiązań kolumn.

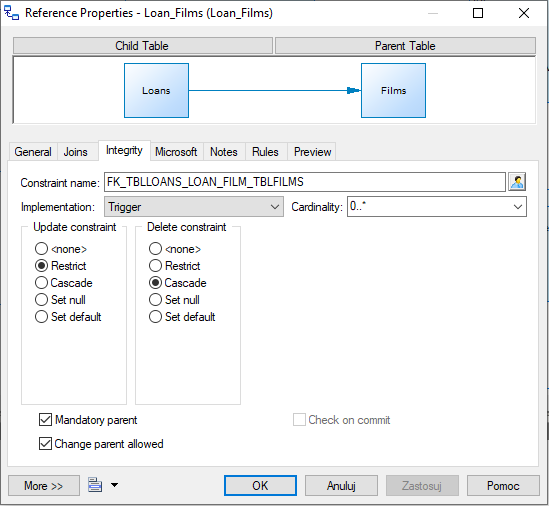
Źródło: opracowanie własne

W zakładce *Integrity* we właściwościach referencji są zawarte wszystkie opcje dotyczące więzów integralności (Rysunek 2.37). Reguły integralności definiowane są osobno dla operacji modyfikacji wartości klucza obcego (*Update*) i usuwania wiersza w tabeli nadrzędnej (*Delete*). Dla tych dwóch rodzajów operacji możliwy jest wybór jednej z pięciu opcji:

* *None* – po zmianach w tabeli nadrzędnej nie są wykonywane żadne zmiany w tabeli podrzędnej (brak kontroli integralności).
* *Restrict* – nie można wykonywać zmian wiersza tabeli nadrzędnej, jeżeli istnieją jakiekolwiek powiązane z nim wiersze tabeli podrzędnej.
* *Cascade* – zmiany dokonane w tabeli nadrzędnej są powielane w tabeli podrzędnej (zmiana wartości klucza obcego lub kasowanie wierszy w tabeli podrzędnej).
* *Set* *null* – po zmodyfikowaniu wartości klucza głównego lub usunięciu wiersza z tabeli nadrzędnej automatycznie wstawiona zostaje wartość null (pole puste) w kolumnie klucza obcego, w powiązanych wierszach tabeli podrzędnej.
* *Set Default* – po zmodyfikowaniu wartości klucza głównego lub usunięciu wiersza z tabeli nadrzędnej automatycznie wstawiona zostaje wartość domyślna w kolumnie klucza obcego, w powiązanych wierszach tabeli podrzędnej. W tym przypadku kolumna klucza obcego musi posiadać zdefiniowaną wartość domyślną.

Wybór opcji *Mandatory parent* dla więzów integralności powoduje, że każdy wiersz w tabeli podrzędnej musi posiadać związany wiersz w tabeli nadrzędnej. Oznacza to, że w kolumnie klucza obcego w tabeli podrzędnej nie mogą występować wartości null. Zastosowanie tej opcji dla przykładu z rysunku 2.34 będzie oznaczało, że każdy wypożyczenie musi być związane z konkretnym filmem. Podczas zapisywania danych dotyczących wypożyczonego filmu będzie wymagane podanie numeru identyfikacyjnego filmu, który został wybrany do wypożyczenia przez klienta.

Ustawienie pola wyboru *Change parent allowed* w oknie więzów integralności (Rysunek 2.37) określa, czy dozwolone sa modyfikacje wartości w kolumnie klucza obcego w tabeli podrzędnej. Jeżeli pole wyboru nie będzie zaznaczone, wówczas nie będzie można zmieniać wartości klucza obcego. Oznacza to, że raz ustalone powiązania wierszy w tabelach nie będą mogły być modyfikowane. Na przykład do wypożyczenia, które zostało zapisane w bazie danych, niedozwolona będzie zmiana numeru identyfikacyjnego filmu. Takie ograniczenie może być zaimplementowane tylko za pomocą wyzwalaczy – aby je zastosować, należy wybrać opcję *Trigger* w polu *Implementation*. [14]



Rysunek 2.37. Okno właściwości referencji "Loans\_Films", opcje więzów integralności.

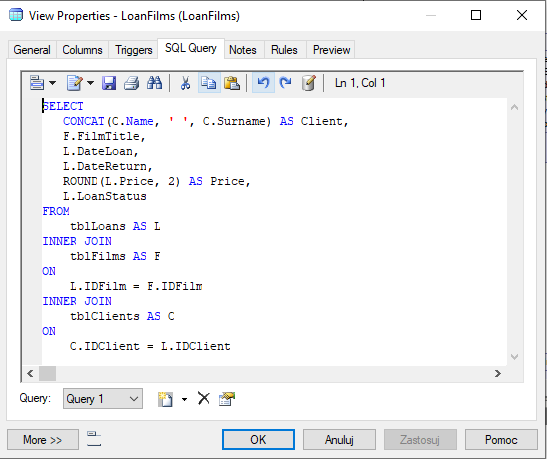
Źródło: opracowanie własne

#### 2.4.6.3. Widoki

Widok (*view*) (PowerDesigner, 2004) jest postrzegany przez użytkownika bazy danych jako zwykła tabela ze zbiorem kolumn i wierszy. Widok nie przechowuje danych, ale udostępnia określony podzbiór danych pochodzący z tabeli lub innych widoków. Widoki są nazywane również perspektywami lub tabelami wirtualnymi. Zastosowanie widoków jest wygodnym i bezpiecznym sposobem udostępniania danych użytkownikom, którzy nie muszą posiadać uprawnień do korzystania z tabel bazy danych, mogą natomiast mieć zezwolenie na korzystanie z widoków zawierających takie zestawy danych, które są potrzebne.

Stosowany na diagramach PDM symbol widoku zawiera trzy elementy: nazwę widoku, listę kolumn oraz listę tabel, do których się odwołuje (Rysunek 2.39). Aby utworzyć nowy widok w modelu fizycznym należy z sekcji *Toolbox* kliknąć przycisk *Physical Diagram*, a następnie kliknąć na element o nazwie *View*. Po kliknięciu na tą opcję dodajemy element do naszego diagramu klikając lewym przyciskiem myszy. We właściwościach widoku na zakładce *General* nadajemy odpowiednią nazwę widoku (pole *Name*) oraz nazwę obiektu, która zostanie użyta w skryptach generowanych na poziomie bazy danych (pole *Code*). Następnie musimy utworzyć zapytanie SQL, które będzie odpowiedzialne za wyświetlenie zawartości widoku. Aby edytować zapytanie SQL wyświetlające zawartość widoku, to we właściwościach widoku należy przejść do zakładki *SQL Query* (Rysunek 2.38). W tej zakładce możemy wpisać zawartość zapytania, które ma zwrócić baza danych w formie widoku. [14]

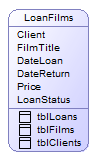
Na poniższym rysunku został przedstawiony jeden (z czternastu) widoków, które utworzyłem w modelu fizycznym. Widok będzie miał za zadanie wyświetlić dane wszystkich klientów, którzy wypożyczyli filmy. W zapytaniu typu SELECT służący do wybierania odpowiednich danych z bazy, będziemy wyświetlali następujące dane: imię i nazwisko klienta, tytuł filmu, datę wypożyczenia i oddania filmu, cenę oraz status wypożyczenia. W zapytaniu SQL do wyświetlenia danych została użyta funkcja CONCAT, która odpowiada za złączenie dwóch lub więcej ciągów znaków. W tej sytuacji będziemy z użyciem tej funkcji łączyli kolumny *Name* oraz *Surname* z tabeli *Clients*. Druga funkcja wbudowana, która została użyta w zapytaniu to ROUND pozwalająca na zaokrąglanie liczby do określonej liczby po przecinku. W tym przypadku użyjemy tej funkcji do wyświetlenia ceny, jaką klient zapłacił za wypożyczony film, gdzie cena zostanie podana do dwóch miejsc po przecinku. Inne kolumny, jakie zostaną użyte w zapytaniu to tytuł filmu, gdzie jest on zapisany w tabeli *Films* oraz data wypożyczenia, data oddania i status znajdujący się w tabeli *Loans*. Do złączenia tabel została użyta funkcja INNER JOIN, która wybierze rekordy pasujące do odpowiednich wartości tabel *Films* oraz *Clients*. Jeżeli uzupełniliśmy zawartość zapytania, to klikamy przycisk OK, aby utworzyć widok z danymi.



Rysunek 2.38. Okno właściwości widoku "LoanFilms" zawierający zapytanie SQL.

Źródło: opracowanie własne

Wygląd widoku w modelu fizycznym na przykładzie *LoansFilms* został przedstawiony na poniższym rysunku. W górnej części tego symbolu zamieszczona jest nazwa widoku, poniżej znajduje się lista kolumn jakie zostaną wyświetlone w wirtualnej tabeli wraz z ich nazwami. Na samym dole symbolu zostaną wyświetlone wszystkie tabele, jakie zostaną użyte do wyświetlenia perspektywy.



Rysunek 2.39. Symbol widoku w modelu fizycznym na przykładzie "LoanFilms".

Źródło: opracowanie własne

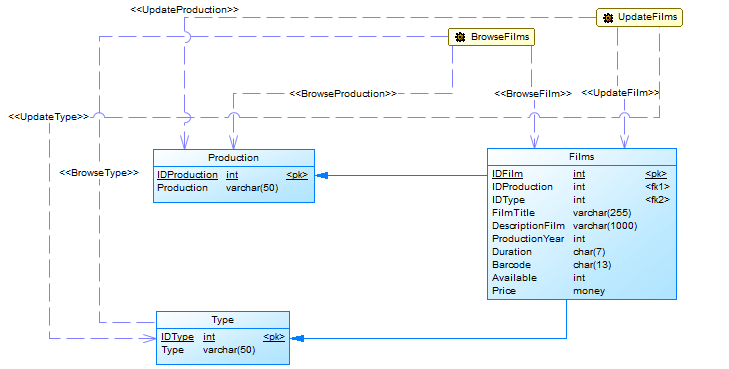
#### 2.4.6.4. Procedury

Procedury wbudowanie (*Stored Procedures*) (PowerDesigner, 2004) są zestawami poleceń języka SQL, które sa przechowywane w bazie danych. Procedury są tworzone specjalnie dla konkretnej bazy danych. Dzięki użyciu procedur różne aplikacje i użytkownicy mogą korzystać z bazy danych i wykonywać różne operacje posługując się jedynie nazwami procedur, bez konieczności używania języka SQL.

Zaletą stosowania procedur jest standaryzacja operacji wykonywanych na bazie danych przez różnych użytkowników i aplikacje, które się odwołują do tych samych procedur. Gdy baza danych jest zainstalowana na serwerze sieciowym, wówczas zastosowanie procedur wykonywanych na komputerze serwera ogranicza potrzebę komunikacji przez sieć i pozwala poprawić efektywność pracy. Kolejną zaletą jest poprawa bezpieczeństwa danych dzięki możliwości nadawania uprawnień do korzystania z procedur tylko wybranym użytkownikom. [14]

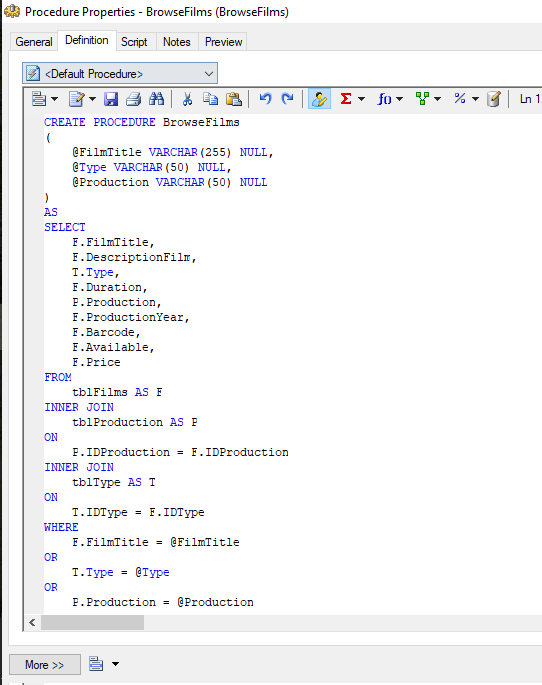
Na Rysunku 2.40 są zawarte symbole procedur, jakie są widoczne na diagramie modelu fizycznego. Jeżeli opracujemy procedurę, w której kodzie znajdą się odwołania do tabel, widoków lub innych procedur, to na diagramie automatycznie zostaną wstawione odpowiednie symbole zależności (strzałki rysowane linią przerywaną). Na fragmencie tworzonego modelu fizycznego dla systemu wypożyczalni filmów są widoczne dwie procedury. Z diagramu możemy wywnioskować, że procedury *BrowseFilms* i *UpdateFilms* odwołują się do tabel *Films*, *Production* oraz *Type*.

Aby utworzyć nową procedurę składowaną w modelu fizycznym, należy skorzystać z palety *Toolbox*, a następnie po kliknięciu na przycisk *Physical Diagram* wybieramy narzędzie o nazwie *Procedure*. Po dodaniu procedury do odpowiedniego miejsca na diagramie, klikamy dwukrotnie lewym przyciskiem myszy na dodany element, gdzie w zakładce *General* w oknie właściwości wykonujemy takie same czynności, jakie wykonywaliśmy przy dodawaniu nowego widoku (punkt 2.4.6.3). W kolejnym kroku należy uzupełnić definicję procedury w zakładce *Definition* (Rysunek 2.41). Za pomocą listy wyboru dostępnej w górnej części okna wybieramy opcję utworzenia procedury (*Default Procedure*) lub funkcji (*Default Function*). Po dokonaniu tego wyboru w polu edytora SQL pojawi się szablon procedury lub funkcji, który należy uzupełnić właściwymi instrukcjami.



Rysunek 2.40. Symbol procedur w diagramie PDM.

Źródło: opracowanie własne



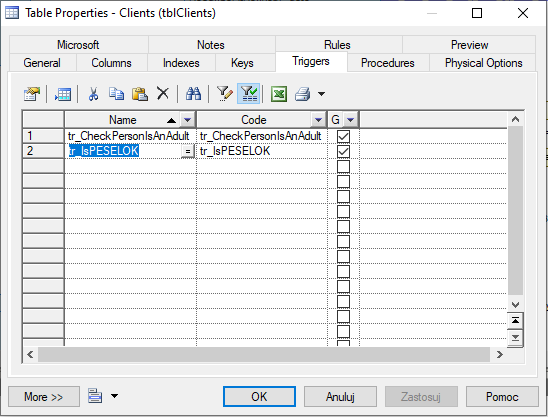
Rysunek 2.41. Definicja procedury "BrowseFilms".

Źródło: opracowanie własne

Jak możemy zauważyć na powyższym rysunku została zdefiniowana w systemie jedna (spośród siedmiu) procedur o nazwie *BrowseFilms*, która odpowiada za wyszukiwanie filmów w bazie danych według określonych kryteriów. Dostęp do tej procedury będą mieć zarówno klienci i pracownicy wypożyczalni. Wywołując tę procedurę możemy podać jeden z trzech parametrów: tytuł filmu, gatunek oraz kraj produkcji, gdzie wszystkie parametry są opcjonalne. Jeżeli podczas wywołania procedury nie podamy jednego z parametrów, to musimy w dane miejsce wstawić wartość NULL. Procedura ta odczyta za pomocą polecenia SELECT wszystkie filmy, które sa dostępne w wypożyczalni. Z tabeli *Films* zostaną pobrane rekordy z następujących kolumn: tytuł filmu, opis, czas trwania, rok produkcji, kod kreskowy, dostępność oraz cena za pojedynczy film. Następnie za pomocą instrukcji INNER JOIN zostaną pobrane z tabeli *Production* wszystkie rekordy zawierające kraj produkcji, gdzie wartość kolumny *IDProduction* z tabeli *Production* jest identyczna z wartością kolumny *IDProduction* w tabeli *Films*. Dalej z tabeli *Type* pobieramy wszystkie rekordy zawierające gatunek filmu, gdzie wartość identyfikatora z tabeli *Type* jest taka sama jak wartość kolumny *IDType* w tabeli *Films*. Za pomocą warunku WHERE określamy, czy tytuł filmu, gatunek lub kraj produkcji jest równy filmowi, gatunkowi lub krajowi produkcji, jaki został wpisany przez użytkownika. Przykładowo, jeżeli wyświetlimy filmy, które są komediami lub zostały wyprodukowane w Polsce, to wtedy, zostaną wyświetlone wszystkie filmy spełniające pierwszy jak i drugi warunek.

#### 2.4.6.5. Wyzwalacze

Wyzwalacz (*trigger*) (PowerDesigner, 2004) to procedura wbudowana uruchamiana automatycznie w momencie wystąpienia określonego zdarzenia związanego z tabelą. Wyzwalacz może zostać przypisany do zdarzenia polegającego na wykonaniu operacji wstawienia nowego wiersza do tabeli (*Insert*), modyfikacji (*Update*) lub usunięcia wiersza istniejącego (*Delete*). Uruchomienie wyzwalacza przez SZBD, w zależności od wybranych ustawień, następuje przed (*Before*) lub po (*After*) zajściu określonego zdarzenia. Dzięki wyzwalaczom możliwe jest zastosowanie dodatkowych mechanizmów kontroli integralności danych, nadzorowanie modyfikacji wprowadzanych w bazie danych i wykonywanie czynności porządkujących. Za pomocą wyzwalaczy, możemy również ograniczać dostęp do pewnych podzbiorów danych.

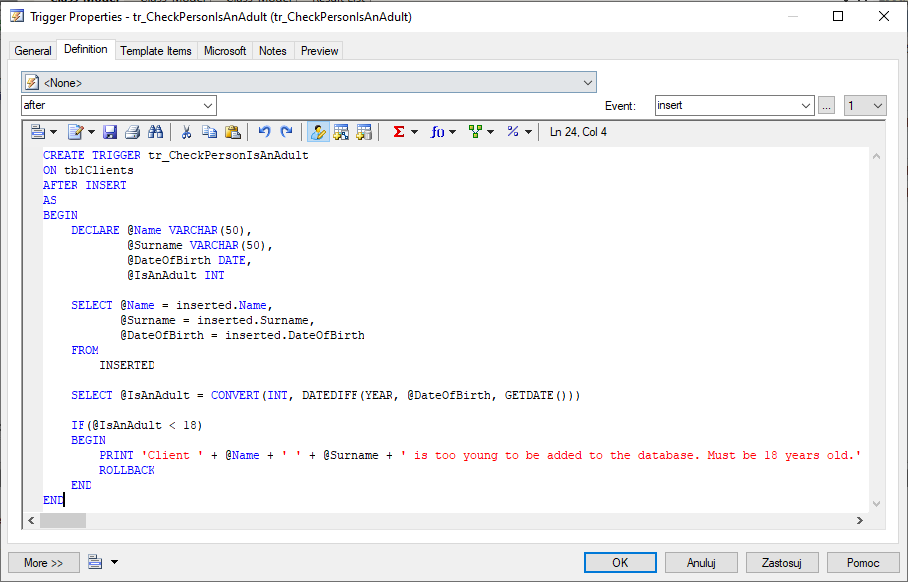


Rysunek 2.42. Okno właściwości tabeli "Clients", lista wyzwalaczy.

Źródło: opracowanie własne

W modelu fizycznym możemy definiować wyzwalacze dla poszczególnych tabel. Aby przejść do listy wszystkich wyzwalaczy dla wybranej tabeli w obszarze roboczym klikamy dwukrotnie lewym przyciskiem myszy na wybraną tabelę. Pojawi się okno właściwości tabeli. Następnie przechodzimy do zakładki *Triggers*, gdzie możemy zobaczyć listę wszystkich wyzwalaczy zdefiniowanych dla wybranej tabeli. Na powyższym rysunku został przedstawiona lista wyzwalaczy, jakie zostały zdefiniowane w tabeli *Clients* na potrzeby tworzonego systemu.

Nowy wyzwalacz możemy dodać za pomocą przycisku *Add a Row*. Okno właściwości wyzwalaczy otwiera przycisk *Properties*. W oknie właściwości wyzwalacza, na zakładce *Definition* (Rysunek 2.43) należy wybrać z listy *Trigger Template* szablon wyzwalacza dla odpowiedniego zdarzenia. Rodzaj zdarzenia (*insert*, *update* lub *delete*) wyświetlony zostaje w polu *Event*. Natomiast w polu *Time* wskazana jest opcja, która określa moment uruchomienia wyzwalacza (*before* lub *after*). Jeżeli dla danego zdarzenia istnieje kilka wyzwalaczy, wówczas kolejność ich uruchamiania określamy za pomocą opcji *Order number*. Wybrana wartość oznacza numer wyzwalacza w kolejności uruchamiania. Po dokonaniu ustawień dla wymienionych wyżej opcji, w polu edytora SQL wyświetlony zostanie szkielet kodu wyzwalacza. [14]



Rysunek 2.43. Definicja wyzwalacza sprawdzającego pełnoletniość klienta.

Źródło: opracowanie własne

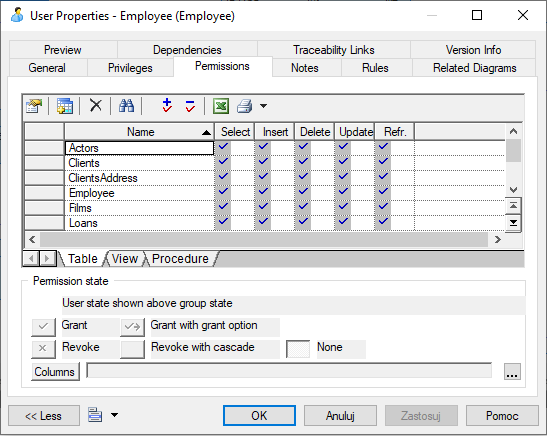
Na powyższym rysunku przedstawiono przykład definicji wyzwalacza jaki został zastosowany dla tabeli *Clients*. Wyzwalacz uruchamiany jest po wstawieniu nowego wiersza do tabeli (*after insert*) czyli w momencie próby zapisu nowego klienta do bazy danych. Zadaniem tego wyzwalacza jest sprawdzenie czy klient w chwili rejestracji ma ukończone 18 lat. Wewnątrz wyzwalacza zostały zadeklarowane cztery zmienne: imię (*Name*), nazwisko (*Surname*), data urodzenia (*DateOfBirth*) oraz wiek klienta (*IsAnAdult*). Następnie w tabeli INSERTED do zmiennych *Name*, *Surname* oraz *DateOfBirth* zostaną przypisane wartości, które określiliśmy w instrukcji INSERT dla tych kolumn. Do zmiennej *IsAnAdult* zostanie przypisany wynik funkcji DATEDIFF, który zwróci różnicę pomiędzy dwoma datami. Jako pierwszy parametr podajemy część daty jaką chcemy zwrócić, gdzie w naszym przypadku będziemy zwracali rok urodzenia. Następnie w drugim parametrze podajemy datę urodzenia klienta, a jako trzeci parametr należy podać dzisiejszą datę, która zostanie pobrana z systemu za pomocą funkcji GETDATE. Funkcja DATEDIFF zwraca wynik typu *„datetime”*, więc aby rzutować wynik do typu int musimy użyć funkcji CONVERT. W instrukcji warunkowej IF należy sprawdzić, czy wiek klienta jest mniejszy od 18 lat. Jeżeli warunek zostanie spełniony to za pomocą instrukcji PRINT należy wyświetlić komunikat, że klient o takim imieniu i nazwisku jest za młody, aby dodać do korzystania z usług wypożyczalni i musi mieć ukończone 18 lat. Wtedy jednocześnie wyzwalacz wygeneruje odpowiedni wyjątek i wstrzyma operację zapisu danych do tabeli.

#### 2.4.6.6. Użytkownicy i role

Kontrolę i sterowanie dostępem do bazy danych umożliwiają mechanizmy autoryzacji użytkowników, definiowania ról, praw właściwości do obiektów oraz uprawnień. Do autoryzacji poszczególnych użytkowników wykorzystywane są nazwy oraz hasła przydzielane przez administratora bazy danych. Każdy użytkownik posiada określony zakres uprawnień. SZBD sprawdza uprawnienia użytkownika przed wykonaniem na bazie danych każdej żądanej przez niego operacji.

Podczas pracy z modelem fizycznym w PowerDesignerze listę użytkowników projektowanej bazy danych możemy wyświetlić za pomocą polecenia *Model | Users and Roles | Users* z menu głównego. W oknie listy dostępne są narzędzia umożliwiające dodanie nowego użytkownika oraz wyświetlenie okna jego właściwości. W oknie właściwości, na zakładce *General* możemy edytować nazwę (*Name*) oraz określić hasło użytkownika (*Password*), które umożliwia zalogowanie do bazy danych. Przydział uprawnień dla użytkownika jest realizowany za pomocą narzędzi dostępnych na zakładce *Permissions* (Rysunek 2.44). Uprawnienia do tabel, widoków i procedur nadawane są oddzielnie. Przycisk *Add Objects* umożliwia wybór obiektów, których mają dotyczyć uprawnienia nadawane użytkownikowi. Do dyspozycji mamy następujące rodzaje uprawnień:

* *Alter* – zezwolenie na zmiany struktury tabeli,
* *Delete* – zezwolenie na usuwanie wierszy w tabeli lub widoku,
* *Insert* – zezwolenie na dodawanie wierszy do tabeli lub widoku,
* *References* – zezwolenie na tworzenie indeksów i kluczy obcych odnoszących się do tabeli,
* *Select* – zezwolenie na odczyt danych z tabeli lub widoku,
* *Update* – zezwolenie na modyfikację danych w tabeli lub widoku,
* *Exec* – zezwolenie na uruchamianie procedury.

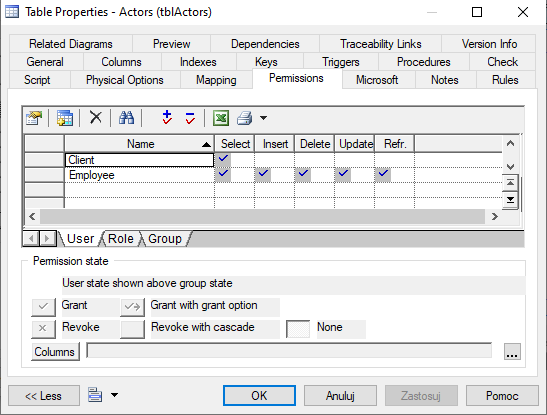


Rysunek 2.44. Okno właściwości użytkownika "Employee", uprawnienia dotyczące tabel.

Źródło: opracowanie własne

Jak możemy zauważyć na powyższym rysunku zostały zdefiniowane uprawnienia dla tabel, jakie posiada użytkownik *Employee*. Dostęp do wykonywanych operacji na tym koncie posiadają tylko i wyłącznie pracownicy wypożyczalni. W przypadku widoków i procedur składowanych pracownik także posiada wszystkie uprawnienia dla tych czynności. Nadawanie uprawnień dla tego użytkownika polega na zaznaczeniu odpowiednich pól w oknie właściwości, tak jak to przedstawia przykład zamieszczony na rysunku 2.44 oraz 2.45. Do oznaczenia stanu uprawnień stosowane są następujące symbole:

* *Grant* – uprawnienie zostało nadane użytkownikowi,
* *Grant with grant options* – uprawnienie zostało nadane użytkownikowi i dodatkowo ma on możliwość nadawania tego uprawnienia innym użytkownikom,
* *None* – uprawnienie nie zostało nadane użytkownikowi.

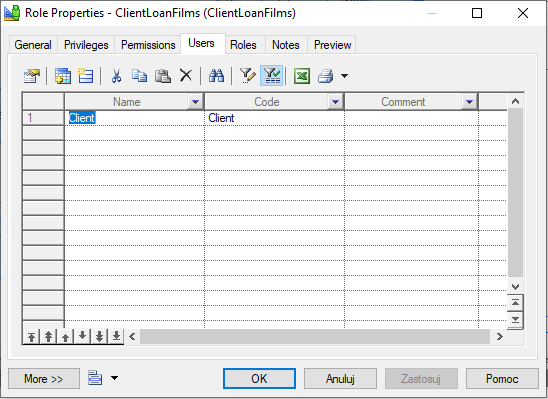


Rysunek 2.45. Uprawnienia użytkowników na przykładzie tabeli Actors w oknie właściwości tabeli.

Źródło: opracowanie własne

Uprawnienia do wykonywania operacji na wykonywanym obiekcie (tabeli, widoku lub procedurze) mogą być edytowane również w oknie właściwości obiektu, na zakładce *Permissions*. Aby zakładka *Permissions* była dostępna, należy wcześniej nacisnąć przycisk *More*. Na powyższym rysunku zostało przedstawione okno właściwości tabeli *Actors* z ustawieniami uprawnień, które dotyczą operacji wykonywanych wyłącznie na tej tabeli. [14] W przypadku tabeli *Actors* dostęp do tej tabeli będą mieli użytkownicy *Clients* oraz *Employee*. Użytkownik *Client* dysponuje uprawnieniami tylko do odczytu (zaznaczony symbol *Grant* dla operacji SELECT), natomiast użytkownik *Employee* posiada wszystkie uprawnienia, jakie można wykonać w tej tabeli.

Aby utworzyć nową rolę w modelu fizycznym PDM należy przejść do górnego menu, gdzie należy wybrać polecenie *Model | Users and Roles | Roles*. Po wybraniu polecenia otworzy się okno zawierające listę wszystkich ról zdefiniowanych w systemie. Aby dodać nową rolę, w oknie *List of Roles* należy kliknąć przycisk *Add a Row*. Następnie nadajemy odpowiednią nazwę roli (pole *Name*) oraz możemy nadać odpowiednie uprawnienia czy dodać użytkownika do roli. Jeżeli chcemy dodać użytkownika do roli, to jest to możliwe w oknie właściwości roli w zakładce *Users* (przycisk *Add Objects*) (Rysunek 2.46). Uprawnienia nadajemy rolom w taki sam sposób jak w przypadku użytkownika, wykorzystując w tym celu opcje dostępne na zakładce *Permissions*. Na poniższym rysunku zostały przedstawione właściwości roli *ClientsLoanFilms*. Dla tej roli został przypisany użytkownik *Client*, który posiada uprawnienia tylko dla tabel: *Production*, *Type*, *Actors*, *Role* i *Films*. Każdy klient, który będzie przeglądał filmy, będzie mógł tylko odczytać ich zawartość. W przypadku widoków klient posiada uprawnienia do 5 widoków, które będą dostępne tylko do odczytu: *ActorsPlayingRolesAtLeastTwoFilms*, *FilmActor*, *FilmMagazine*, *ListComendyAndScienceFictionFilms* i *ProductionOfFilmsInUSAAfter2000*. Dodatkowo klienci mogą wyszukiwać filmy według tytułu filmu, gatunku oraz kraju produkcji posiadając dostęp do procedury *BrowseFilms*.



Rysunek 2.46. Okno właściwości roli "ClientLoanFilms", lista użytkowników przypisanych do roli.

Źródło: opracowanie własne

#### 2.4.6.7. Transformacja CDM na PDM

Środowisko PowerDesigner umożliwia automatyczne generowanie PDM na podstawie istniejącego modelu konceptualnego (punkt 2.4.5.4). W tym punkcie, za pomocą przykładów, zostaną wyjaśnione zasady transformacji obiektów modelu konceptualnego na obiekty modelu fizycznego.

Podczas generowania modelu fizycznego każdy obiekt w CDM zostaje przekształcony na odpowiedni obiekt lub kilka obiektów PDM. Podstawowe zasady tej operacji zostały przedstawione w tabeli 2.4. Encje zostaną przekształcone na tabele, atrybuty na kolumny itd. Szczegółowego wyjaśnienia wymagają zasady transformacji związków oraz pozostałych obiektów CDM, które nie zostały uwzględnione w tabeli 2.4.

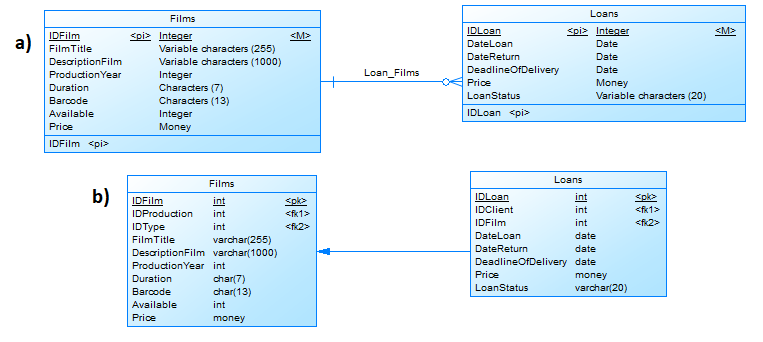
Tabela 2.. Transformacja obiektów CDM na PDM.

Źródło: opracowanie własne

|  |  |
| --- | --- |
| **Obiekt CDM** | **Obiekt PDM** |
| Encja | Tabela |
| Atrybut | Kolumna |
| Identyfikator | Klucz |
| Związek | Referencja, klucz obcy |

**Związek jeden do wielu**

Związki z modelu konceptualnego przekształcane są na referencje w modelu fizycznym. Utworzeniu referencji towarzyszy wstawienie do odpowiedniej tabeli klucza obcego. W przypadku związku typu jeden do wielu kolumna pełniąca rolę klucza obcego wstawiana jest do tabeli, która powstaje z przekształcenia encji stojącej w związku po stronie wiele. Kolumna ta zostaje powiązana z kolumną klucza głównego tabeli, która odpowiada encji stojącej w związku po stronie jeden. [14] Przykład takiego przekształcenia został pokazany na Rysunku 2.47. W tabeli *Loans* zaznaczono za pomocą strzałki kolumnę klucza obcego (Rysunek 2.47 b). Możemy zauważyć na poniższym rysunku, że do tabeli *Loans* został dodany klucz obcy *IDFilm*, który będzie się odwoływał do nazwy atrybutu swojego odpowiednika w tabeli *Films*.



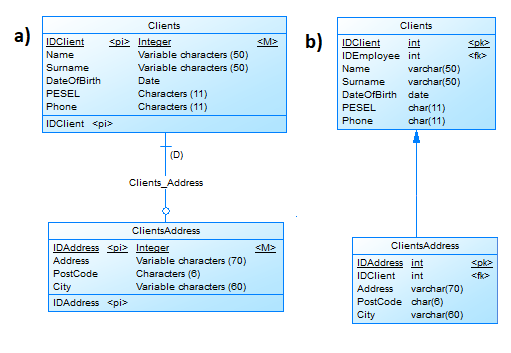
Rysunek 2.47. Transformacja związku jeden do wielu: a) CDM, b) PDM.

Źródło: opracowanie własne

Wymaganie uczestnictwa encji w związku przekłada się w modelu fizycznym na niedopuszczenie wartości NULL w kolumnie klucza obcego. I tak, w przykładzie przedstawionym na Rysunku 2.47 uczestnictwo encji *Films*, stojącej po stronie jeden związku jest wymagane. Zatem wartość klucza obcego *IDFilm* wstawionego do tabeli *Loans* nie może przyjmować wartości NULL. Aby uwzględnić ten warunek, odpowiednio ustawiane są właściwości referencji – zaznaczona zostanie opcja *Mandatory Parent*.

**Związek jeden do jednego**

W przypadku przekształcania związku typu jeden do jednego, który nie posiada zdefiniowanej roli dominującej, w modelu fizycznym zostaną utworzone dwie referencje. Klucze obce są wstawiane do obydwu tabel powstałych z encji uczestniczących w takim związku. Jeżeli natomiast związek jeden do jednego ma określoną rolę dominującą, to wówczas zostanie utworzona tylko jedna referencja i jeden klucz obcy. [14] Możemy przyjąć na przykładzie, który został przedstawiony na Rysunku 2.48 a) rolę dominującą zdefiniowaną w kierunku *Clients -> ClientsAddress*. Wtedy w modelu fizycznym pojawi się referencja, dla której tabelą nadrzędną jest tabela *Clients*, a klucz obcy umieszczony zostanie w tabeli *ClientsAddress* (Rysunek 2.48 b). Kolumny kluczy obcych powstające w wyniku transformacji związku jeden do jednego posiadają indeksy unikatowe, które nie dopuszczają powtórzeń wartości wpisywanych w tych kolumnach.



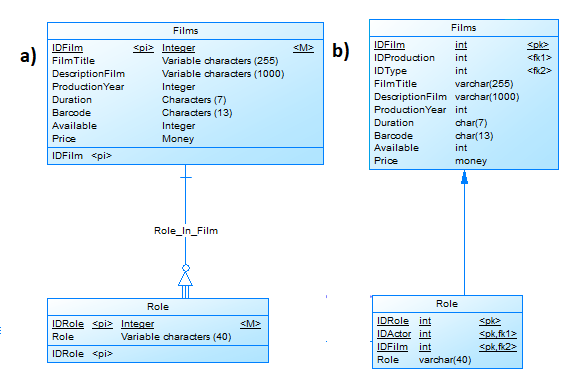
Rysunek 2.48. Transformacja związku jeden do jednego dla roli dominującej: a) CDM, b) PDM.

Źródło: opracowanie własne

**Związek zależny**

Transformacja związku zależnego realizowana jest podobnie jak w przypadku związku niezależnego. Różnica polega na tym, że klucz obcy utworzony w tabeli odpowiadającej encji zależnej staje się częścią klucza głównego tej tabeli. [14]

Na rysunku 2.49 został przedstawiony przykład transformacji związku zależnego. Encją zależną w tym związku jest encja *Role*. Podczas generowania modelu fizycznego PDM utworzona zostanie referencja, a klucz obcy umieszczany jest w tabeli *Role*. Jednocześnie kolumna klucza obcego zostaje wykorzystana jako element klucza głównego. Zatem możemy stwierdzić, że klucz główny tabeli Zadanie jest kluczem złożonym – składa się z trzech kolumn: *IDRole*, *IDActor* i *IDFilm*.



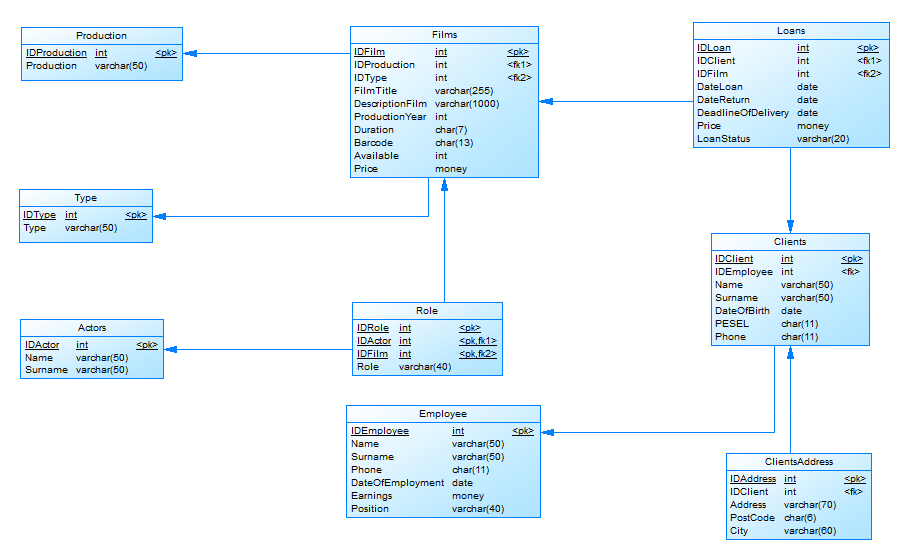
Rysunek 2.49. Transformacja związku zależnego: a) CDM, b) PDM.

Źródło: opracowanie własne

#### 2.4.6.8. Model fizyczny dla systemu wypożyczalni płyt DVD

Model fizyczny dla systemu wypożyczalni płyt DVD został wygenerowany na podstawie modelu konceptualnego. Kolejny krok polegał na rozszerzeniu modelu fizycznego o dodatkowe obiekty: widoki, procedury oraz wyzwalacze. Większość tych obiektów została już przedstawiona w ramach przykładów, które zostały zamieszczone w poprzednich punktach tego podrozdziału. Diagram modelu fizycznego został przedstawiony na rysunku 2.50.

Celem rozwiązań zastosowanych na etapie modelowania fizycznego było uwzględnienie w projekcie wszystkich wymagań funkcjonalnych. Aby zapewnić czytelność omawianego modelu, został przedstawiony sam model wraz z utworzonymi tabelami i relacjami zachodzącymi pomiędzy nimi. Zakres utworzonego projektu pozwolił na wyjaśnienie stosowania wszystkich typów obiektów, które są dostępne w modelu fizycznym danych.



Rysunek 2.50. Model fizyczny dla systemu wypożyczalni DVD.

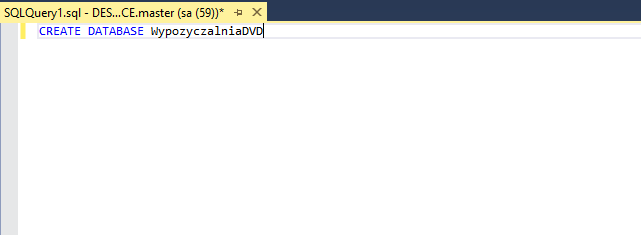
Źródło: opracowanie własne

## 2.5. Implementacja bazy danych

W tym podrozdziale zostaną omówione czynności, które należy wykonać w celu zaimplementowania projektu bazy danych. Punktem wyjścia dla implementacji jest model fizyczny PDM. Na podstawie modelu fizycznego jest generowany skrypt SQL, czyli zbiór poleceń dla SZBD. Wykonanie tych poleceń spowoduje utworzenie w bazie danych wszystkich niezbędnych obiektów, czyli tabel, widoków, procedur itd. Jako ostateczny rezultat otrzymujemy w pełni funkcjonalną bazę danych, która może zostać uzupełniona danymi i poddana procesowi testowania.

### 2.5.1. Tworzenie nowej bazy danych

Implementacja projektu bazy danych, opracowanego w formie modelu fizycznego, wymaga wcześniejszego utworzenia nowej (pustej) bazy danych. Zgodnie z przyjętym założeniem baza danych zostanie zainstalowana w programie Microsoft SQL Server. Aby utworzyć nową bazę danych, musimy na naszym lokalnym komputerze otworzyć program Microsoft SQL Server Management Studio. Po otwarciu tego programu łączymy się z serwerem, gdzie możemy się zalogować na dwa sposoby: lokalnie za pomocą autoryzacji Windows (w polu *Authentication* wybrać opcję *Windows Authentication*) albo za pomocą autoryzacji SQL Servera (opcja *SQL Server Authentication*). W naszym przypadku zalogujemy się za pomocą drugiej opcji, gdzie w polu *User name* podajemy nazwę użytkownika *sa* oraz hasło do tego konta (w moim przypadku to *admin1234*). Klikamy przycisk *Connect,* aby się połączyć z serwerem SQL przy pomocy tego konta. Po zalogowaniu się do serwera utworzymy nową pustą bazę danych, gdzie w eksploratorze obiektów wybieramy aktualnie połączoną instancję serwera i klikamy prawym przyciskiem myszy na opcję *New Query*. Pojawi się okno, gdzie wpisujemy następujące zapytanie, które zostało przedstawione na Rysunku 2.51.

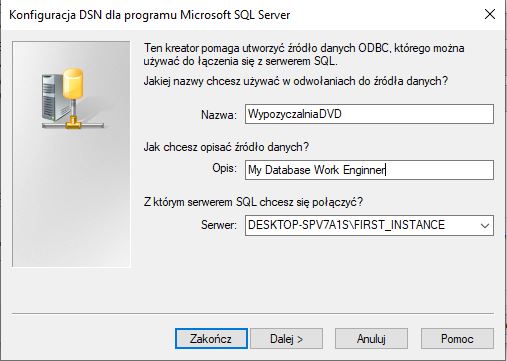


Rysunek 2.51. Zapytanie SQL pozwalające na utworzenie nowej bazy danych.

Źródło: opracowanie własne

Po napisaniu powyższego zapytania klikamy przycisk *Execute* lub przycisk *F5*, aby wykonać zapytanie. Powinien się wyświetlić komunikat o pomyślnym wykonaniu zapytania i zostanie utworzona nowa pusta baza danych o nazwie *WypozyczalniaDVD*. Nowa baza danych będzie miała lokalizację w eksploratorze obiektów w folderze *Databases* po wybraniu aktualnie połączonej instancji serwera.

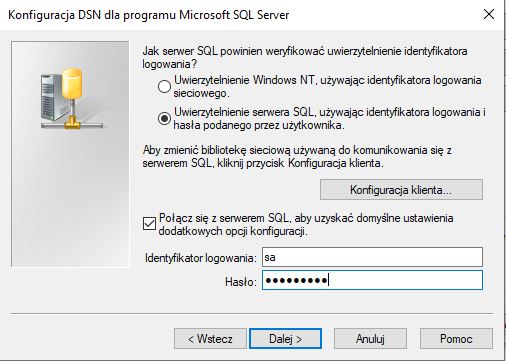
Aby udostępnić nowo utworzoną bazę danych dla innych aplikacji, także dla środowiska PowerDesigner, konieczne jest zdefiniowanie źródła danych ODBC. W tym celu na lokalnym komputerze należy wyszukać program *ODBC Data Sources* i uruchamiamy go jako administrator. Po otwarciu programu w górnym menu wybieramy zakładkę *SystemDSN*, a następnie klikamy przycisk *Add…* aby dodać nowe źródło ODBC. W oknie kreatora nowego źródła danych wybieramy sterownik, dla którego chcemy skonfigurować źródło danych. W tej sytuacji wybieramy z listy program *SQL Server* i klikamy przycisk *Zakończ*. Następnie należy wpisać nazwę i opis oraz wybrać z listy serwer, z którym chcemy się połączyć (Rysunek 2.52). Klikamy przycisk *Dalej*, aby przejść do dalszej części instalacji nowego źródła danych.



Rysunek 2.52. Określenie nazwy i wybór serwera w kreatorze nowego źródła danych ODBC.

Źródło: opracowanie własne

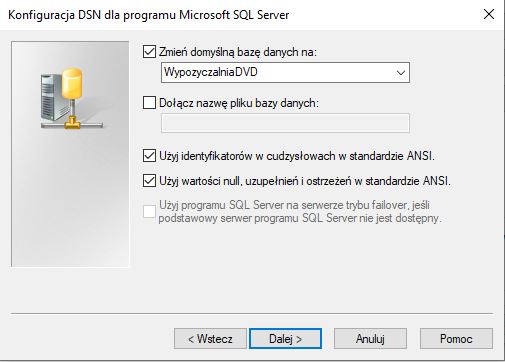
Następnie określamy w jaki sposób serwer SQL powinien weryfikować połączenie za pomocą identyfikatora logowania (Rysunek 2.53). W tym przypadku wybieramy opcję uwierzytelnianie SQL Server za pomocą identyfikatora logowania i hasła wprowadzonego przez użytkownika. Zaznaczamy także pole typu „checkBox” aby przy połączeniu się z serwerem uzyskać domyślne ustawienia dodatkowych opcji konfiguracji. W polu identyfikatora logowania podajemy nazwę *sa* (konto administratora systemu) i hasło do tego konta (*admin1234*).



Rysunek 2.53. Określenie weryfikacji połączenia z SQL Server i podanie loginu oraz hasła w kreatorze źródła danych ODBC.

Źródło: opracowanie własne

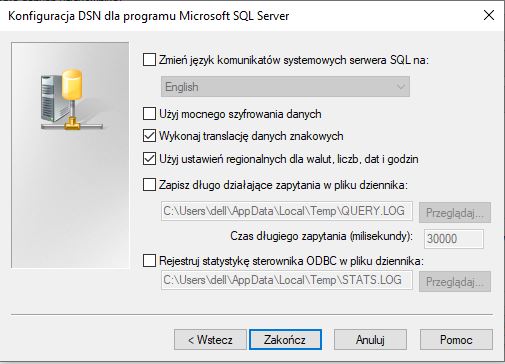
Dalej zaznaczamy pole pozwalające nam na zmianę domyślnej bazy danych przy połączeniu z SQL Server (Rysunek 2.54). Domyślną bazę danych zmieniamy na *WypozyczalniaDVD* i klikamy przycisk *Dalej*.



Rysunek 2.54. Okno pozwalające na ustawienie domyślnej bazy danych w kreatorze źródła danych ODBC.

Źródło: opracowanie własne

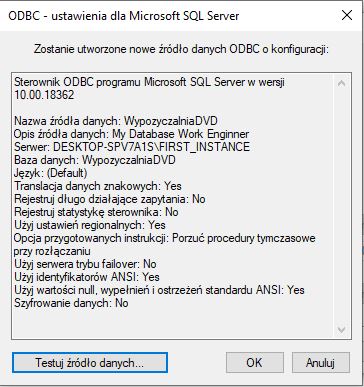
Potem pojawia się okno, gdzie możemy między innymi zmienić język komunikatów systemowych serwera SQL czy użyć ustawień regionalnych dla walut, liczb, dat i godzin (Rysunek 2.55). W tym kroku nie należy dostosowywać żadnych ustawień, gdzie wszystkie ustawienia pozostawiamy opcjami domyślnymi. Klikamy przycisk *Zakończ*, aby zakończyć proces dodawania nowego źródła danych.



Rysunek 2.55 Okno dokonywania ustawień dla nowego źródła danych ODBC.

Źródło: opracowanie własne

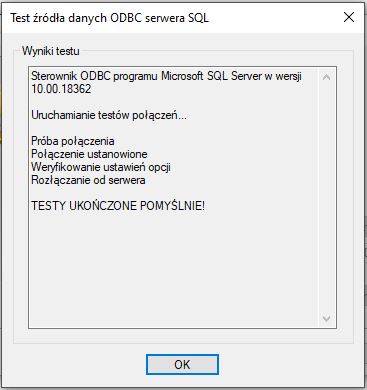
Pojawi się okno dialogowe ODBC, zawierające podsumowanie konfiguracji źródła danych dla Microsoft SQL Server (Rysunek 2.56).



Rysunek 2.56. Okno zawierające podsumowanie konfiguracji ODBC dla Microsoft SQL Server.

Źródło: opracowanie własne

Aby przetestować połączenie z serwerem SQL przy pomocy źródła danych, należy kliknąć przycisk *Testuj źródło danych*. Powinno się wyświetlić okno zawierające podsumowanie konfiguracji źródła danych oraz wyniki testowania połączenia z serwerem. Jak możemy zauważyć na Rysunku 2.57 testy połączenia się źródła danych z SQL Server zostały ukończone pomyślnie. Po pomyślnym zakończeniu testu klikamy przycisk OK, aby zamknąć kreator.



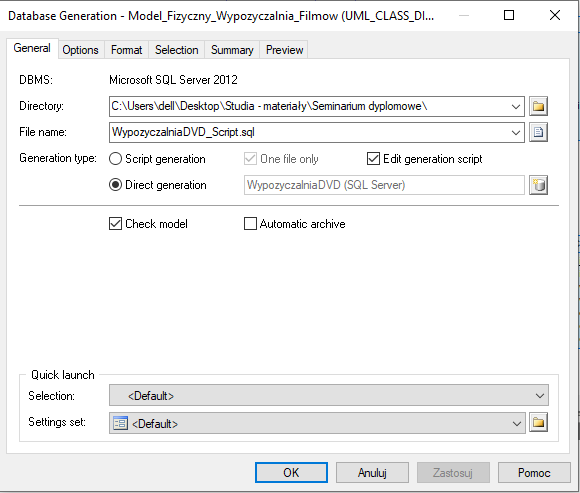
Rysunek 2.57. Okno zawierające podsumowanie konfiguracji źródła ODBC wraz z testem połączenia.

Źródło: opracowanie własne

### 2.5.2. Generowanie obiektów w bazie danych

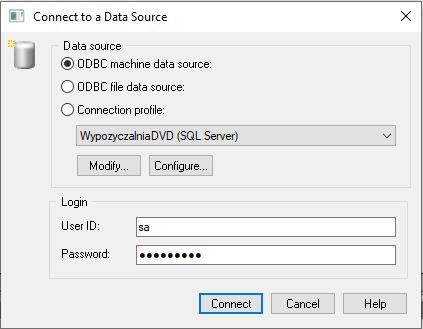
Generowanie obiektów oznacza utworzenie w bazie danych schematu, który musi być zgodny z modelem fizycznym opracowanym w PowerDesignerze. W pierwszym etapie tej operacji wygenerowany zostaje skrypt SQL, który zawiera polecenia tworzące poszczególne obiekty. Drugi etap obejmuje wykonanie poleceń zawartych w tym pliku na wskazanej bazie danych. W naszym przypadku będzie to nowo utworzona, pusta baza danych o nazwie *WypozyczalniaDVD*.

Aby wygenerować skrypt zawierającą strukturę bazy danych, należy w menu głównym programu PowerDesigner dla modelu fizycznego wybrać opcję *Database | Generate Database*. Pojawi się okno opcji generowania bazy danych, gdzie dla niej musimy określić ścieżkę dostępu i nadać nazwę pliku (Rysunek 2.58). W moim przypadku plik będzie nosił nazwę *WypozyczalniaDVD\_Script.sql*, a lokalizacja tego skryptu będzie się znajdować w następującym katalogu: *„C:\Users\dell\Desktop\Studia - materialy\Seminarium dyplomowe\”*. Następnie w polu *Generation type* wybieramy typ generowania bazy danych na opcję *Direct generation*. Opcja ta, oznacza wygenerowanie i zapis skryptu z rozszerzeniem SQL oraz wykonanie zawartych w nim poleceń. Jeżeli zostanie wybrana ta opcja, to musimy się połączyć z bazą danych, na której polecenia mają zostać wykonane. Do tego celu służy przycisk *Connect to a Data Source*. Przycisk otwiera okno, w którym należy wybrać źródło danych ODBC dla połączenia z docelową bazą danych (Rysunek 2.59). W oknie połączenia ze źródłem danych ODBC, w sekcji *Data Source* wybieramy opcję *ODBC machine data source* oraz wybieramy źródło danych, z którym chcemy się połączyć. Z pola typu „combo” wybieramy źródło danych *WypozyczalniaDVD (SQL Server)*. Następnie w sekcji *Login* podajemy nazwę użytkownika i hasło jakie podaliśmy dla tego źródła danych. Klikamy przycisk *Connect,* aby się połączyć z bazą danych SQL Server. [14]



Rysunek 2.58. Okno opcji generowania bazy danych.

Źródło: opracowanie własne

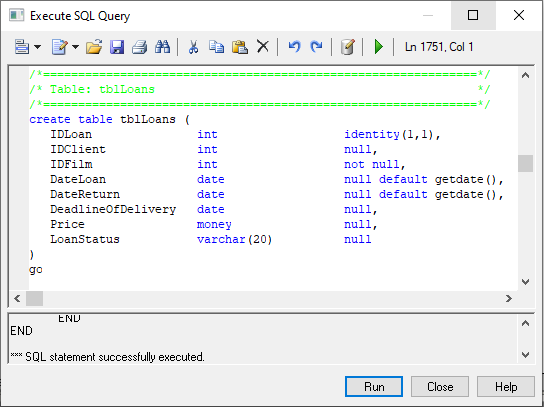


Rysunek 2.59. Okno połączenia się z źródłem danych ODBC.

Źródło: opracowanie własne

Szczegółowy opis wszystkich parametrów dostępnych w oknie generowania bazy danych można znaleźć w systemie pomocy programu PowerDesigner. Klikamy przycisk *Zastosuj*, aby zatwierdzić wszystkie ustawienia dokonane w oknie generowania bazy danych. Po zatwierdzeniu ustawień, klikamy przycisk *OK*, aby rozpocząć proces generowania bazy danych. W trakcie generowania bazy następuje sprawdzenie modelu fizycznego (jeżeli zaznaczona została opcja *Check Model*) oraz tworzy się plik z rozszerzeniem SQL zawierający całą strukturę bazy danych. Skrypt może zostać wyświetlony w oknie edytora. W przypadku zaznaczenia opcji *Direct generation* zostanie otwarte okno, w którym od razu możemy uruchomić wykonanie poleceń SQL tworzących obiekty w bazie danych (przycisk *RUN*). Na rysunku 2.60 został pokazany fragment skryptu, który zawiera polecenie utworzenia tabeli *Loans*.

Po uruchomieniu skryptu, możemy sprawdzić status realizacji poszczególnych poleceń. Jeżeli dane polecenie zostało wykonane poprawnie, pojawi się komunikat *Statement successfully executed*. W przypadku niepowodzenia wyświetlona zostanie informacja o błędzie. [14]

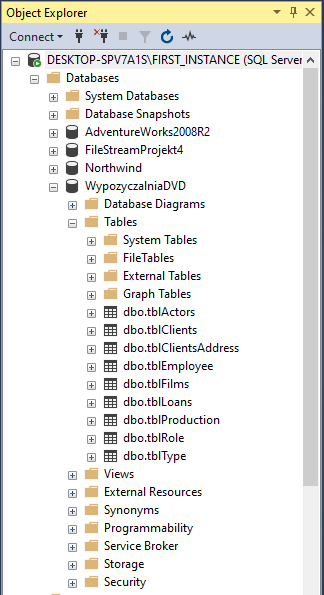


Rysunek 2.60. Okno skryptu SQL.

Źródło: opracowanie własne

Po wygenerowaniu bazy danych w PowerDesignerze, otwieramy program Microsoft SQL Server Management Studio, gdzie musimy się połączyć z serwerem. W oknie połączenia z bazą danych należy wybrać odpowiednią autoryzację (*SQL Server Authentication*) oraz podać nazwę użytkownika (*sa*) i hasło (*admin1234*). Jeżeli jesteśmy połączeni z serwerem, to otwieramy plik *WypozyczalniaDVD\_Script.sql*. W polu *Available Databases* znajdującego się u góry programu, zmieniamy bazę danych na *WypozyczalniaDVD*. Potem uruchamiamy skrypt za pomocą przycisku *Execute* lub klawisza *F5*. Jeżeli skrypt zostanie wykonany prawidłowo, to należy nacisnąć przycisk *Refresh* w eksploratorze obiektów, a następnie wybrać połączoną instancję serwera i rozwinąć katalog *Database | WypozyczalniaDVD | Tables*. Jak możemy zauważyć na rysunku 2.61 wszystkie tabele, zostały prawidłowo utworzone w bazie danych.

Na tym etapie implementacji dysponujemy już bazą danych, której schemat jest zgodny z tym, co zostało zaprojektowane w modelu fizycznym. Baza danych nie zawiera jeszcze żadnych danych, wszystkie jej tabele są puste.



Rysunek 2.61. Widok tabel w bazie danych WypozyczalniaDVD w Microsoft SQL Server Management Studio.

Źródło: opracowanie własne

# Rozdział III: Testowanie systemu bazy danych

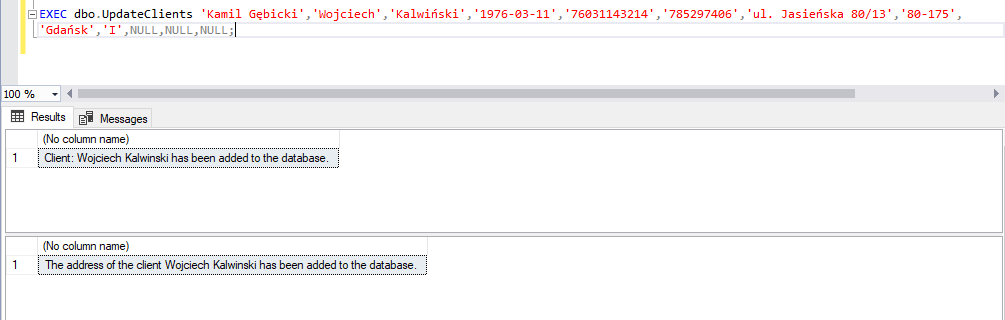
Baza danych jest jedną z ważnych części oprogramowania, gdzie jej spójność i integralność muszą być monitorowane i tworzone zgodnie z systemem. W każdym procesie tworzenia oprogramowania testowanie bazy danych jest krokiem, którego nie można pominąć. Jest ona uważana za serce każdej tworzonej aplikacji, więc niezbędne jest, aby sprawdzić poprawność wstawianych danych. Testowanie bazy danych obejmuje wykonywanie ważności danych, testowanie integralności danych, sprawdzenie wydajności związane z bazą oraz testowanie wyzwalaczy i funkcji w bazie danych. Bez w pełni działającej bazy danych system nie może zostać wykorzystany we właściwy sposób. Baza danych jest miejscem przechowującym wszystkie informacje i działa jako krytyczny punkt końcowy każdego oprogramowania. Głównym celem testowania bazy danych jest sprawdzenie czy system spełnia wszystkie funkcjonalności podane w wymaganiach oraz wykrycie możliwych błędów, jakie mogą wystąpić w bazie.

## 3.1. Testowanie systemu

W celu sprawdzenia poprawności działania bazy danych przetestowałem system pod względem wykonanych funkcjonalności, gdzie zweryfikowałem utworzone procedury, wyzwalacze, widoki itp. W tym procesie zostały wstawione przykładowe dane testowe dla poszczególnych tabel, gdzie musieliśmy zweryfikować, czy wstawione dane są zgodne.

Bazę danych została przetestowana w środowisku Microsoft SQL Server Management Studio (wersja 15.0.2080.9), gdzie do niego jest zainstalowany SQL Server 2019. Otwieramy program, gdzie łączymy się z serwerem logując się na konta *Employee* oraz *Client* utworzone w modelu fizycznym w programie PowerDesigner. W polu *Authentification* należy wybrać autoryzację SQL Servera, a następnie podajemy nazwę użytkownika (*Employee*) oraz hasło (*employee4G1!7*). Po połączeniu się z bazą danych dla konta *Employee* logujemy się do konta *Client*, gdzie w eksploratorze obiektów klikamy na przycisk *Connect*, a następnie z listy rozwijanej klikamy na opcję *Database Engine*. Pojawi się okno połączenia z serwerem, gdzie tym razem w polu *Login* podajemy użytkownika *Client* oraz wpisujemy hasło (*clientV2$7p*). Po połączeniu się z tym loginem w eksploratorze obiektów powinien się pojawić kolejny serwer, który będzie połączony z loginem *Client*. Od tej pory możemy w pełni korzystać z bazy danych, gdzie możemy wykonywać różne operacje na tabelach lub widokach.

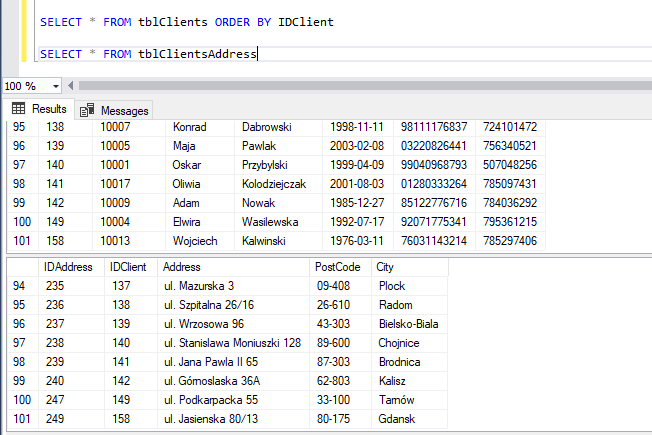
Pierwszą funkcjonalnością, jaka jest testowana to dodawanie, edycja i usuwanie klientów. W bazie danych będą się znajdowały dane klientów, którzy korzystają z oferowanych usług w wypożyczalni. Każdy klient, który będzie się znajdował w bazie danych musi podać swoje dane osobowe oraz adres zamieszkania w celu poprawnego działania bazy danych. Jeżeli chcemy dodać nowego klienta do bazy danych, to na koncie *Employee* wykonujemy zapytanie, które uruchomi procedurę *UpdateClients*. Procedura *UpdateClients* przyjmuje aż 13 parametrów, gdzie w pierwszym parametrze podajemy imię i nazwisko pracownika, który będzie obsługiwał danego klienta. Od drugiego do szóstego parametru podajemy dane, które znajdą się w tabeli *Clients*. W tabeli klientów, będą się znajdowały takie dane jak: imię i nazwisko, data urodzenia, PESEL oraz numer telefonu. Natomiast od siódmego do dziewiątego parametru podajemy dane, jakie znajdą się w tabeli *ClientsAddress*. W tej tabeli będą się znajdowały następujące dane: adres, kod pocztowy oraz miasto. Ostatnie 3 parametry wskazują odpowiednio na identyfikator, który musi zostać użyty w operacji edycji i usuwania rekordu oraz na stare imię i nazwisko klienta. Stare imię i nazwisko klienta zostaną wypełnione tylko wtedy, gdy będzie wykonywana w bazie danych operacja edycji klienta. Wykonujemy procedurę dodawania nowego klienta do bazy, podając odpowiednie parametry. Jeżeli nie został nadany identyfikator klienta oraz wartości starych danych klienta, to w odpowiednich parametrach tej procedury należy wstawić wartości NULL. Jeżeli zapytanie zostanie wykonane, to pojawią się dwa rodzaje komunikatów: o pomyślnym dodaniu klienta oraz adresie klienta do bazy danych. Zawartość procedury odpowiadającej za dodanie nowego klienta do bazy został przedstawiony na Rysunku 3.1.



Rysunek 3.1. Wykonanie procedury "UpdateClients" dodającą nowego klienta do tabel Clients i ClientsAddress.

Źródło: opracowanie własne

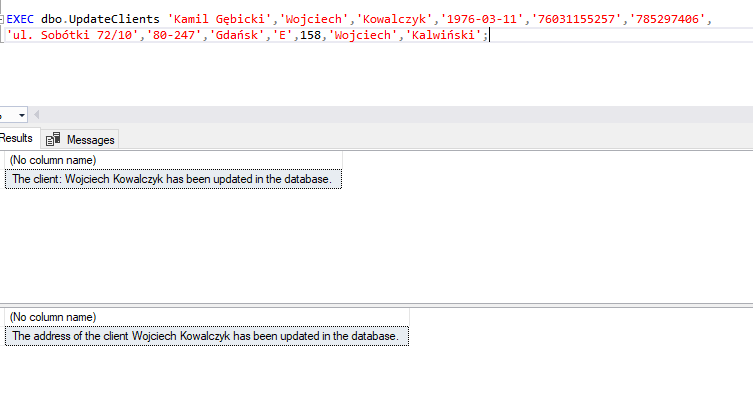
Teraz należy wykonać zapytania w bazie danych, które pobiorą zawartość tabel *Clients* oraz *ClientsAddress*. W treści zapytania należy wpisać polecenie SELECT, z której zostanie pobrana cała zawartość tabel *Clients* oraz *ClientsAddress*. Dodatkowo tabela klientów zostanie posortowana (instrukcja ORDER BY) rosnąco według kolumny *IDClient*. Jak możemy zauważyć na rysunku 3.2 nowe dane klienta zostały prawidłowo dodane do tabel *Clients* i *ClientsAddress*. W tabeli *Clients* dla wybranego klienta został przyporządkowany pracownik, do którego został pobrany identyfikator z tabeli *Employee* odpowiadający wybranemu pracownikowi. Dane adresowe nowego klienta zostały połączone przy pomocy kolumny *IDClient*, która za pomocą klucza obcego, łączy się za pomocą relacji z kolumną o takiej samej nazwie będącą kluczem głównym tabeli *Clients*.



Rysunek 3.2. Zawartość tabel "Clients" i "ClientsAddress" po dodaniu klienta do bazy danych.

Źródło: opracowanie własne

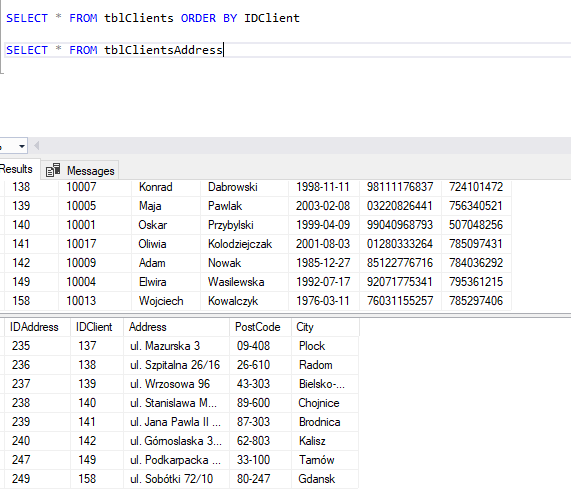
W przypadku, gdy pracownik źle wpisze dane klienta lub dane adresowe, to należy dane odpowiednio zaktualizować. Na koncie *Employee* wykonujemy nowe zapytanie do bazy, w którym uruchamiamy procedurę *UpdateClients* (Rysunek 3.3). Przykładowo możemy założyć, że pracownik niepoprawnie wpisał imię i nazwisko, numer PESEL w tabeli *Clients* oraz adres i kod pocztowy w tabeli *ClientsAddress*. W parametrach procedury zmieniamy odpowiednie wartości wybranych parametrów. Natomiast do ostatnich trzech parametrów wpisujemy identyfikator klienta, w którym zostaną zmodyfikowane dane adresowe oraz wpisujemy stare imię i nazwisko, pod którym był przypisany klient. Po wykonaniu tego zapytania zostaną wyświetlone komunikaty o zaktualizowanych danych klienta oraz danych adresowych w bazie danych.



Rysunek 3.3. Wykonanie procedury "UpdateClients" modyfikująca dane klienta w tabelach Clients i ClientsAddress.

Źródło: opracowanie własne

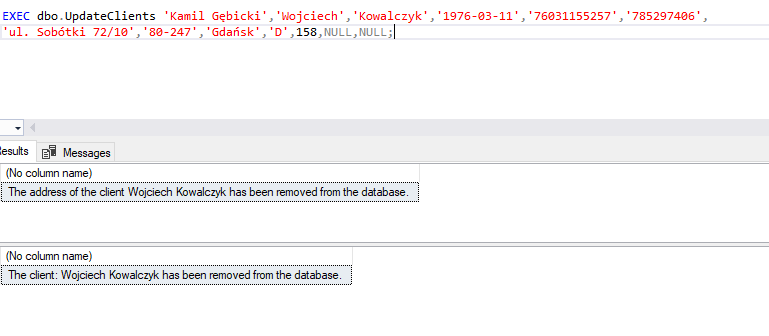
Jeżeli została wykonana procedura, to uruchamiamy kolejne zapytania do bazy, w których sprawdzimy czy dane klienta i adresowe zostały poprawnie zaktualizowane w tabelach (Rysunek 3.4). Po pobraniu wszystkich rekordów z tabel za pomocą instrukcji SELECT, możemy zauważyć, że dane które wprowadziliśmy do edycji w kodzie procedury, zostały poprawnie zmienione w tabelach.



Rysunek 3.4. Zawartość tabel "Clients" i "ClientsAddress" po zmodyfikowaniu klienta.

Źródło: opracowanie własne

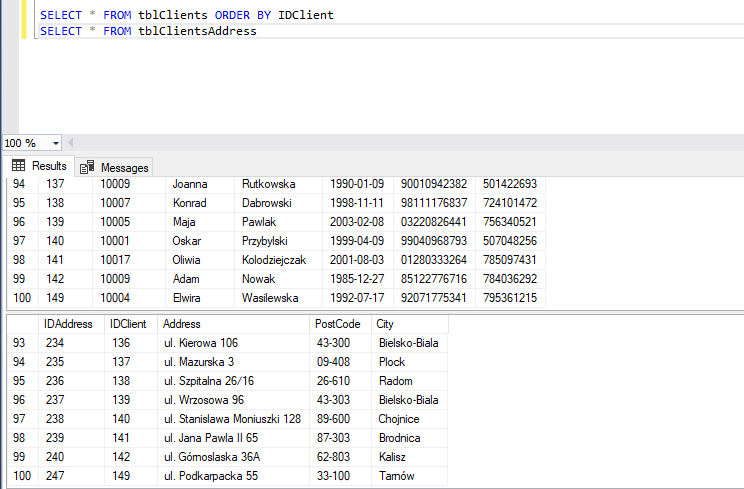
Kiedy się zdarzy sytuacja, że klient chcę zrezygnować z korzystania wszystkich usług wypożyczalni, to należy wykonać operację usuwania klienta i jego danych adresowych. Uruchamiamy procedurę *UpdateClients*, która pozwoli na usunięcie wybranego klienta z bazy danych. W parametrach procedury wpisujemy pracownika, który obsługiwał danego klienta, wszystkie dane, jego dane adresowe oraz określamy identyfikator klienta, pod którym był przypisany adres. W tej sytuacji musimy zastosować usuwanie kaskadowe. Usunięcie wiersza w tabeli nadrzędnej (tabela *Clients*) lub wartości klucza głównego (kolumna *IDClient*) wymaga usunięcia lub zaktualizowania wartości w powiązanym z nim kluczu obcym (kolumna *IDClient* w tabeli *ClientsAddress*). Jak można zauważyć na rysunku 3.5 po wykonaniu kodu procedury zostały wyświetlone komunikaty o pomyślnym usunięciu danych klienta oraz danych adresowych z bazy.



Rysunek 3.5. Wykonanie procedury "UpdateClients" usuwająca dane klienta z bazy danych.

Źródło: opracowanie własne

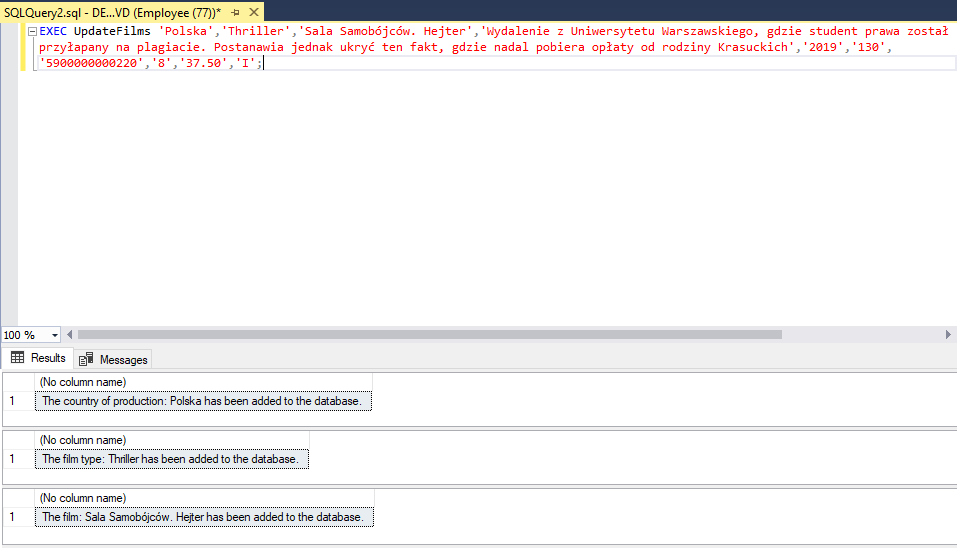
Dalej wykonujemy zapytania, w których sprawdzimy, czy dane klienta i adresowe zostały poprawnie usunięte z tabel *Clients* oraz *ClientsAddress*. Wykonujemy zapytania typu SELECT, gdzie pobieramy zawartość wszystkich rekordów z wymienionych wcześniej tabel (Rysunek 3.6). Jak można zauważyć na poniższym rysunku dane klienta oraz jego adres zostały prawidłowo usunięte z bazy danych.



Rysunek 3.6. Zawartość tabel "Clients" i "ClientsAddress" po usunięciu klienta z bazy danych.

Źródło: opracowanie własne

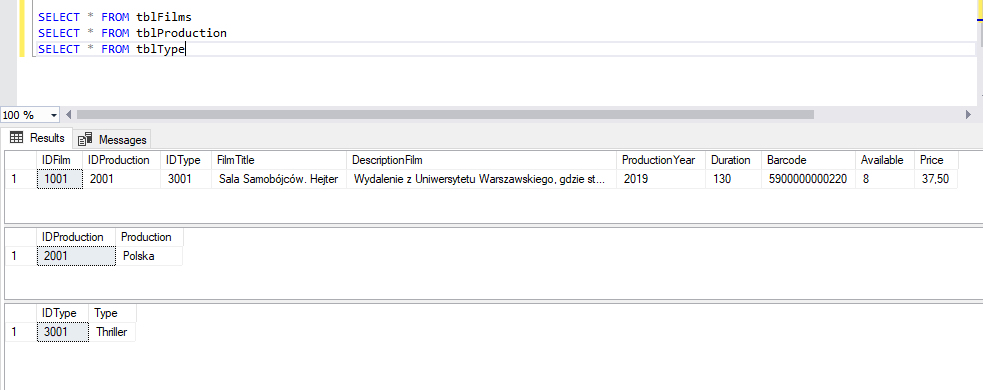
Następną funkcjonalnością, jaką należy przetestować jest dodawanie, edytowanie i usuwanie filmów. Sprawdzenie tej funkcjonalności jest jednym z kluczowych elementów płynnego i sprawnego działania systemu. Filmy, które będą przechowywane w bazie danych zostaną przyporządkowane do gatunku oraz kraju produkcji, w którym dany film został wyprodukowany. Jeżeli chcemy dodać nowy film do bazy, to należy wykonać kwerendę, która uruchomi procedurę o nazwie *UpdateFilm* (Rysunek 3.7). Procedura przyjmuje 10 parametrów. W pierwszym parametrze podajemy kraj produkcji, gdzie jego wartość zostanie dodana do tabeli *Production*. Przy dodawaniu filmu do bazy w tabeli *Films* zostanie wyświetlona wartość kolumny *IDProduction*, pod którą został przypisany rekord w tabeli *Production*. Jako drugi parametr procedury została podana nazwa gatunku filmu, która zostanie dodana do tabeli *Type*. Po dodaniu nowego filmu w tabeli zostanie wyświetlona wartość kolumny *IDType* odpowiadająca wartości rekordu przypisanego do kolumny *Type* w tabeli gatunków. Pozostałe parametry tej procedury odpowiadają poszczególnym kolumnom w tabeli *Films*, których wartości są w takiej samej kolejności jak kolumny w tabeli. Ostatni parametr procedury wskazuje na operację jaka ma zostać wykonana w bazie danych. W tej sytuacji operacją wykonywaną w tej procedurze będzie INSERT. Jeżeli zapytanie zostanie wykonane, to w systemie wyświetlą się komunikaty o pomyślnie dodanym kraju produkcji, gatunku oraz filmu do bazy danych.



Rysunek 3.7. Wykonanie procedury "UpdateFilms" dodająca nowy film do bazy danych.

Źródło: opracowanie własne

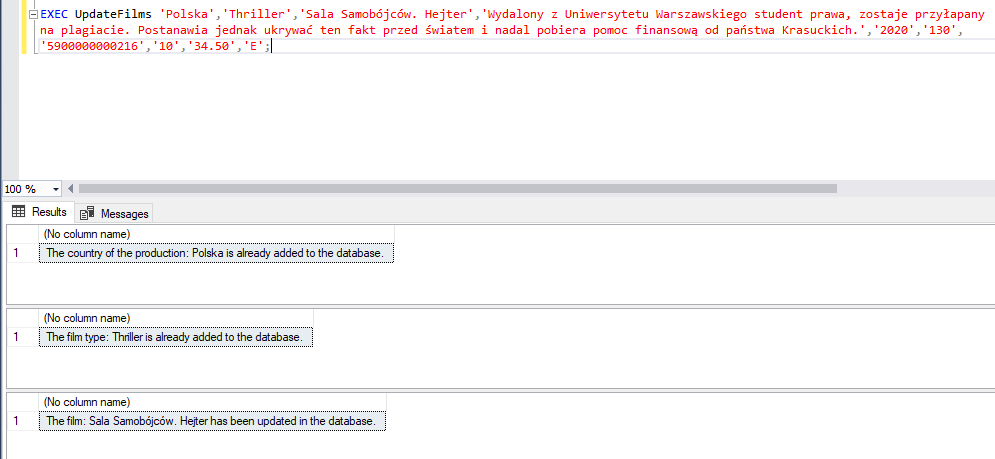
Teraz musimy się upewnić, czy nowy film wraz z gatunkiem i produkcją został zapisany w bazie danych. W bazie danych wykonujemy zapytania typu SELECT, w którym pobierzemy całą zawartość tabel *Films*, *Production* i *Type*. Jak możemy zauważyć na rysunku 3.8 dane zostały poprawnie dodane do odpowiednich kolumn w tabelach.



Rysunek 3.8. Zawartość tabel "Films", "Production" i "Type" po dodaniu nowego filmu.

Źródło: opracowanie własne

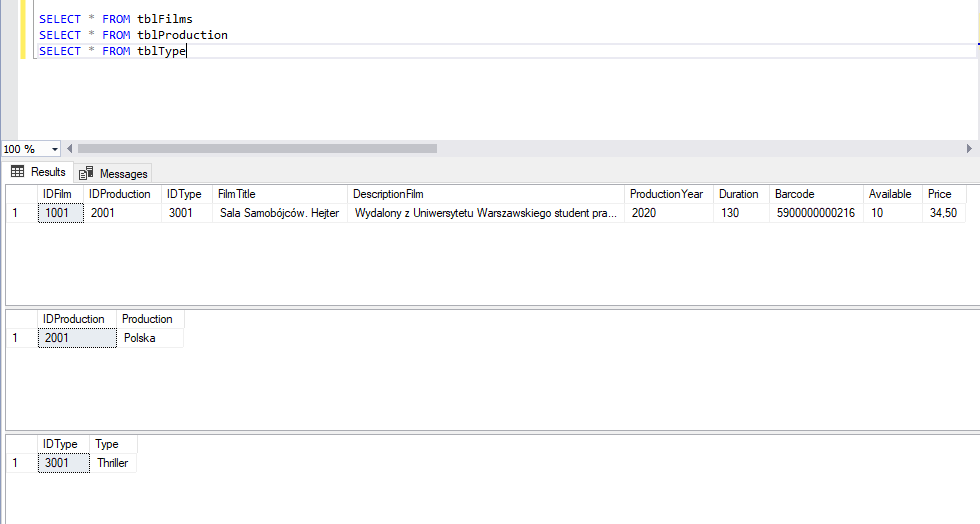
Gdy się zdarzy sytuacja, że pracownik wpisał dane filmu nieprawidłowo, to należy dokonać edycji jego danych w bazie. Po wybraniu opcji *New Query*, na koncie *Employee* uruchamiamy procedurę *UpdateFilms* (Rysunek 3.9). Przykładowo zakładamy, że pracownik źle wpisał dane dotyczące opisu filmu, roku produkcji, dostępności i ceny. W tej procedurze zmieniamy wartości wybranych parametrów na odpowiednie. Jeżeli będziemy chcieli edytować wartości kraju produkcji lub gatunku, to procedura wykona operację INSERT, która wstawi nową wartość do tabeli *Type* lub *Production*. Wtedy w tabeli *Films* zostanie zmieniona wartość identyfikatora, odpowiadająca nowej wartości rekordu przypisanej do gatunku lub produkcji. Ostatni parametr oznaczony literą „E” wskazuje na operację UPDATE, jaka ma zostać wykonana w bazie danych. Po wykonaniu zapytania za pomocą przycisku *Execute*, zostaną wyświetlone komunikaty o zaktualizowaniu kraju produkcji, gatunku oraz filmu w bazie danych.



Rysunek 3.9. Wykonanie procedury "UpdateFilms" modyfikująca dany film w tabeli.

Źródło: opracowanie własne

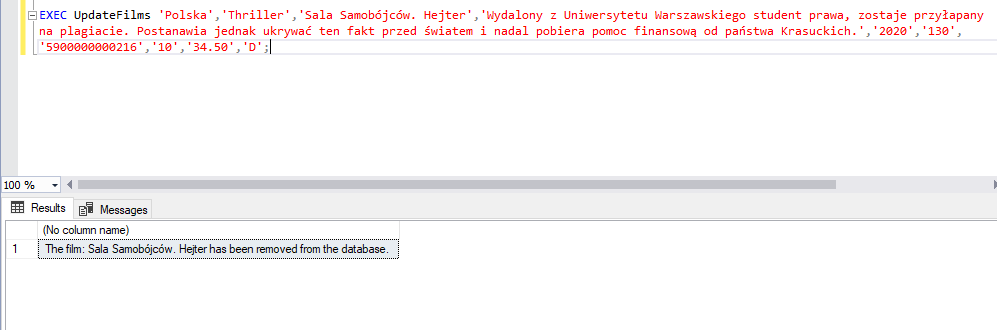
Następnie wykonujemy ponownie zapytania do bazy, w których sprawdzimy, czy dane filmu zostały prawidłowo zmienione w tabeli (Rysunek 3.10). Po pobraniu rekordów z tabel *Films*, *Production* oraz *Type* za pomocą instrukcji SELECT widzimy, że dane dotyczące filmu zostały poprawnie zmodyfikowane w tabeli.



Rysunek 3.10. Zawartość tabel "Films", "Production" i "Type" po zmodyfikowaniu filmu w bazie danych.

Źródło: opracowanie własne

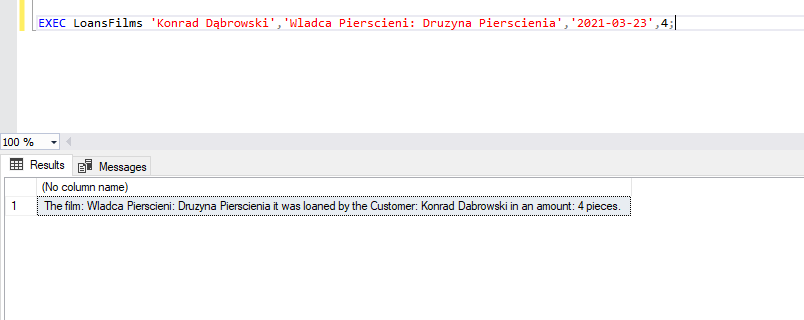
Gdy się wydarzy sytuacja, że dany film nie jest wypożyczany przez klientów, to pracownik może usunąć wybrany film z bazy. W tej sytuacji należy uruchomić procedurę *UpdateFilms*, gdzie w parametrach podajemy wszystkie dane filmu do usunięcia (Rysunek 3.11). Jako ostatni parametr podajemy literę „D”, która wskazuje na operację DELETE, pozwalającą na usunięcie wybranego rekordu. Jeżeli zapytanie zostanie poprawnie wykonane, to system wyświetli komunikat o pomyślnym usunięciu filmu z bazy. W sytuacji, gdy po usunięciu filmu wykonamy polecenie SELECT to zostanie zwrócona pusta zawartość tabeli *Films*.



Rysunek 3.11. Wykonanie procedury "UpdateFilms" usuwająca wybrany film z tabeli.

Źródło: opracowanie własne

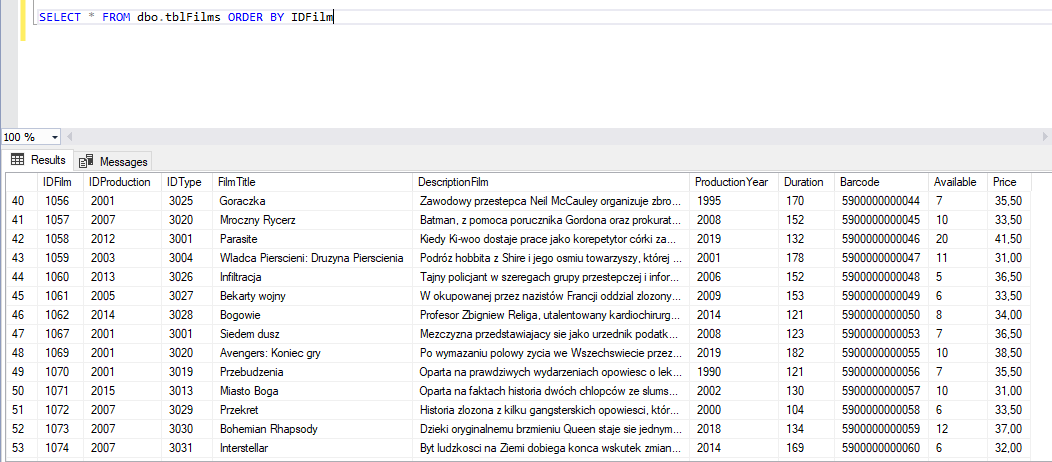
Kluczową funkcjonalnością w działaniu systemu bazy danych będzie wypożyczanie filmu przez klientów. Każdy klient, który korzysta z wszystkich oferowanych usług wypożyczalni, będzie miał możliwość wypożyczenia filmów dostępnych w magazynie. System musi odnotować film, który został wypożyczony oraz jednocześnie w kolumnie *Available* tabeli *Films* zaktualizować wartość kolumny o liczbę filmów, jaką wypożyczył klient. W tabeli *Loans* zostanie pobrany identyfikator klienta, do którego jest przyporządkowany wybrany klient w wypożyczalni. Analogicznie do kolumny *IDFilm* w tabeli *Loans* zostanie przypisany odpowiedni film, jaki się znajduje w tabeli *Films* pod tym numerem ID. Po odnotowaniu wypożyczenia przez klienta pracownik musi odnotować datę wypożyczenia i jednocześnie system ustawi termin oddania filmu na czas 7 dni. Wykonujemy zapytanie do bazy danych, która pozwoli pracownikowi uruchomić procedurę *LoanFilms* (Rysunek 3.12). W parametrach procedury określamy imię i nazwisko klienta, nazwę filmu, datę wypożyczenia oraz ilość sztuk. Po wykonaniu procedury system wyświetli odpowiedni komunikat o wypożyczonym filmie przez danego klienta w odpowiedniej ilości sztuk.



Rysunek 3.12. Wykonanie procedury "LoanFilms" pozwalająca na wypożyczenie filmu przez klienta.

Źródło: opracowanie własne

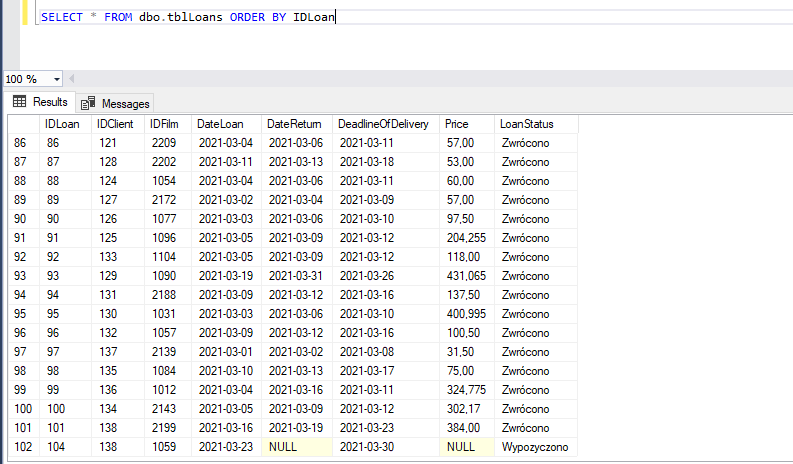
Teraz należy sprawdzić w bazie zawartość tabeli *Films*, w której sprawdzimy, czy wartość kolumny *Available* dla wybranego filmu została zmieniona. Wykonujemy zapytanie typu SELECT, gdzie pobieramy wszystkie kolumny z tabeli oraz posortujemy wyniki zapytania według identyfikatora filmu (Rysunek 3.13). Po wyświetleniu tabeli, możemy zauważyć, że dla filmu *„Władca Pierścieni: Drużyna Pierścienia”*, została zaktualizowana wartość kolumny *Available* z 15 na 11.



Rysunek 3.13. Zawartość tabeli "Films" po operacji wypożyczenia filmu przez klienta.

Źródło: opracowanie własne

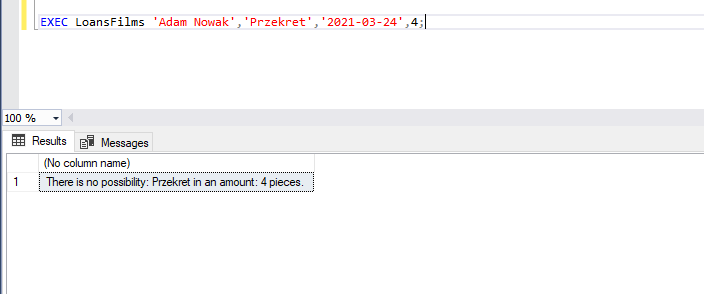
Później wykonujemy zapytanie do bazy danych, aby sprawdzić zawartość tabeli *Loans* po dodaniu wypożyczenia filmu. Wykonujemy kwerendę typu SELECT, gdzie pobieramy wszystkie kolumny z tabeli *Loans*. Rekordy zostaną posortowane po identyfikatorze wypożyczenia (Rysunek 3.14). Po wyświetleniu zawartości zapytania, możemy zauważyć, że w ostatnim wstawionym rekordzie zostały nadane odpowiednie numery identyfikatorów wskazujące na odpowiedniego klienta i film w bazie danych. Dodatkowo w systemie pojawiła się informacja dotycząca planowanego oddania filmu (kolumna *DeadlineOfDelivery*), gdzie data została ustawiona 7 dni do przodu. Status wypożyczenia został ustawiony na wartość *Wypożyczono*, która będzie informować pracownika, że dany film jest aktualnie wypożyczony przez klienta.



Rysunek 3.14. Zawartość tabeli "Loans" po wypożyczeniu filmu przez klienta.

Źródło: opracowanie własne

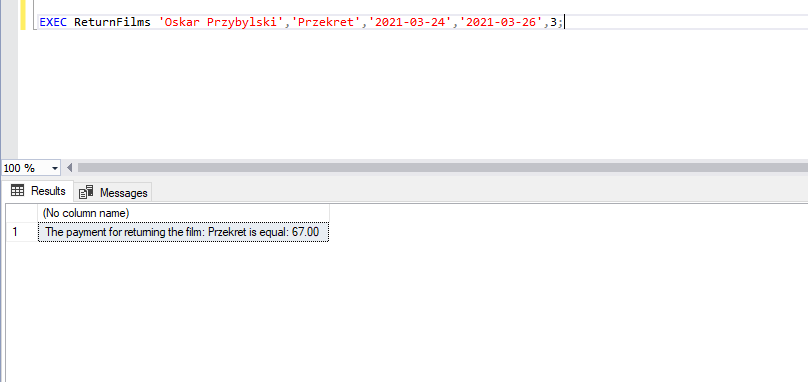
W sytuacji, kiedy klient chce wypożyczyć film w ilości większej od stanu posiadanego w magazynie, to system musi zablokować realizację wypożyczenia. Pracownik musi poinformować klienta o braku możliwości wypożyczenia wybranego filmu w takiej ilości. Wtedy pracownik może klientowi zaproponować wypożyczenie filmu w ilości mniejszej od tego co znajduje się w magazynie lub wybrać inny film. Uruchamiamy procedurę *LoansFilms*, gdzie wpisujemy odpowiednie parametry (Rysunek 3.15). Po wykonaniu zapytania w bazie, system wyświetla komunikat o braku możliwości wypożyczenia filmu w ilości podanej przez użytkownika.



Rysunek 3.15. Wykonanie procedury "LoansFilms" wraz z wyświetleniem komunikatu o zablokowaniu wypożyczenia filmu.

Źródło: opracowanie filmu

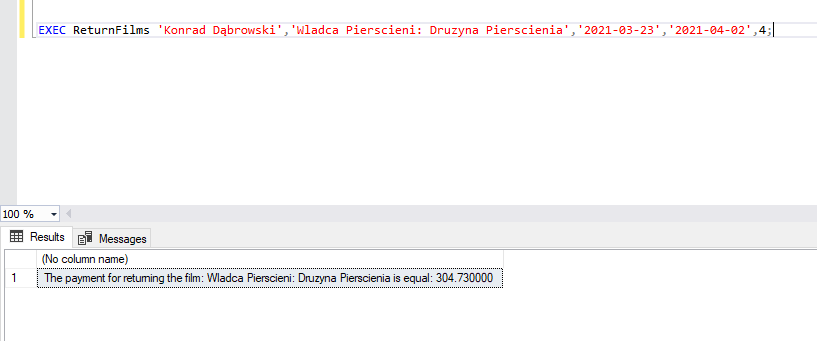
Kolejną funkcjonalnością, jaką należy przetestować to oddawanie filmów przez klientów. Każdy klient, który ma wypożyczone filmy powinien je oddać w określonym terminie. Po oddaniu wypożyczonego filmu przez klienta w kolumnie *Available* tabeli *Films* jest aktualizowana wartość tej kolumny o ilość sztuk filmu jaką oddaje klient. Natomiast w tabeli *Loans* system sprawdza, czy film wypożyczony przez klienta został oddany w terminie. Jeżeli klient odda film w ciągu 7 dni od wypożyczenia, to nie musi ponosić dodatkowych opłat. W przeciwnym wypadku, do ceny filmu zostanie naliczona opłata karna, która będzie wynosić 1 % za każdy dzień zwłoki. Aby pracownik mógł odnotować oddanie filmu w bazie danych, to musi wykonać zapytanie uruchamiające procedurę o nazwie *ReturnFilms* (Rysunek 3.16). W tej procedurze należy podać następujące parametry: imię i nazwisko klienta, tytuł filmu, datę wypożyczenia, datę oddania oraz ilość filmu, jaką chcę oddać klient. Dla tego przykładu zostały przedstawione dane klienta, który oddaje film w terminie 3 dni od wypożyczenia filmu. Gdy zapytanie zostanie wykonane za pomocą przycisku *Execute*, system wyświetli komunikat o cenie filmu, jaką musi ponieść klient za wypożyczony film.



Rysunek 3.16. Wykonanie procedury "ReturnFilms" pozwalająca na oddawanie filmu przez klienta.

Źródło: opracowanie własne

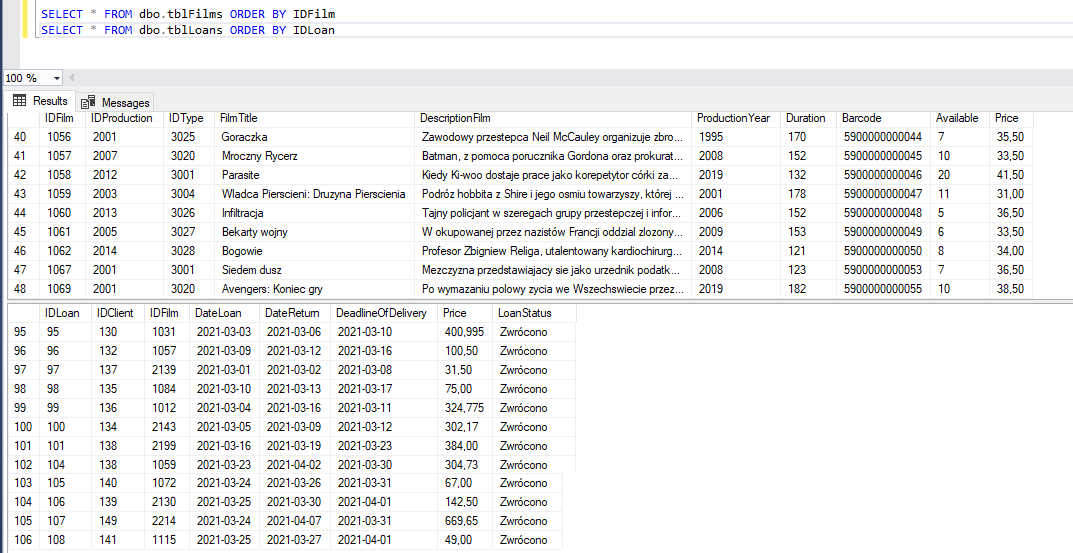
Gdyby się zdarzyła sytuacja, że klient oddaje wypożyczony film po 7 dniach, to pracownik musi obliczyć ostateczną sumę wraz z opłatą karną, jaką poniesie klient. Zostanie wykonane zapytanie w bazie, gdzie ponownie zostanie uruchomiona procedura *ReturnFilms* (Rysunek 3.17). W parametrach procedury podajemy tym razem dane innego klienta, który oddał film w ciągu 10 dni od wypożyczenia. Jeżeli zapytanie zostanie wykonane w bazie, to system wyświetli komunikat o cenie filmu, do którego została doliczona również opłata karna, jaką musi ponieść klient za wypożyczony film.



Rysunek 3.17. Wykonanie procedury "ReturnFilms" dla filmu oddawanego po terminie.

Źródło: opracowanie własne

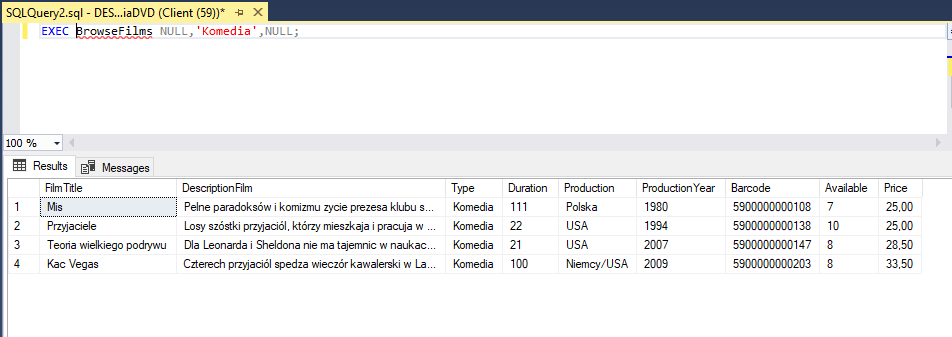
Teraz wykonujemy zapytania, które pozwolą na wyświetlenie zawartości tabeli *Loans* i *Films* po oddaniu filmów przez klientów. Wykonujemy zapytania typu SELECT, gdzie pobieramy wszystkie kolumny z wymienionych powyżej tabel. Jak możemy zauważyć na rysunku 3.18 w tabeli *Loans* zostały dodane wartości kolumn *DateReturn* oraz *Price* dla tych wypożyczeń, które wcześniej posiadały status *Wypożyczono*. W tabeli *Films* dla kolumny *Available* zostały zaktualizowane wartości dla filmów, które zostały oddane do wypożyczalni. System zaktualizował je do liczby początkowej wynoszącą tyle, ile było ich dostępnych w magazynie. W tej chwili filmy zostały oddane do wypożyczalni, a system prawidłowo obliczył cenę za każdy wypożyczony film oraz zaktualizował status na wartość *Zwrócono*.



Rysunek 3.18. Zawartość tabel "Films" i "Loans" po oddaniu filmów przez klientów.

Źródło: opracowanie własne

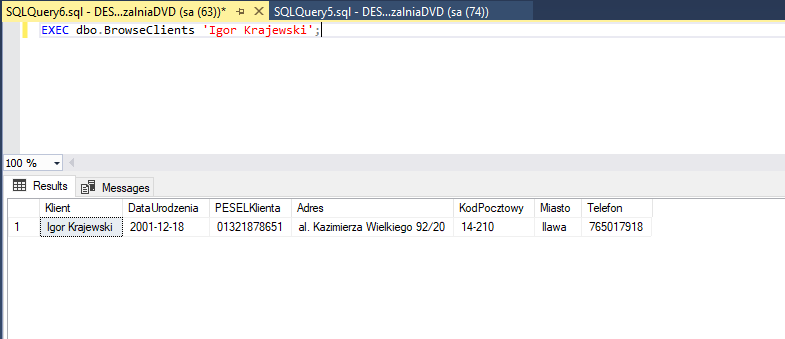
Przeglądanie filmów jest kolejną funkcjonalnością jaką należy sprawdzić w utworzonym systemie bazy danych. Filmy w bazie będzie można wyszukać według trzech kryteriów: tytule filmu, gatunku lub kraju produkcji. Dostęp do tej funkcjonalności, będą mieli zarówno klienci, jak i pracownicy wypożyczalni. W celu przetestowania tej funkcjonalności, wykonano instrukcje SQL dla konta *Client*. Aby klient mógł w pełni skorzystać z tej funkcji, w eksploratorze obiektów wybieramy odpowiednią instancję serwera, do którego jest zalogowany użytkownik *Client*. Na tym koncie należy wykonać zapytanie, które pozwoli na uruchomienie procedury *BrowseFilms* (Rysunek 3.19). Procedura ta przyjmuje trzy parametry, które odpowiadają kryteriom wymienionym powyżej w identycznej kolejności. Co jest ważne, wszystkie parametry tej procedury są opcjonalne, co oznacza, że można dodać wartości NULL. Na poniższym przykładzie zostało pokazane działanie tej procedury, która wyświetliła filmy należące do gatunku *Komedia*.



Rysunek 3.19. Wykonanie procedury "BrowseFilms" pozwalająca na przeglądanie filmów według określonych kryteriów.

Źródło: opracowanie własne

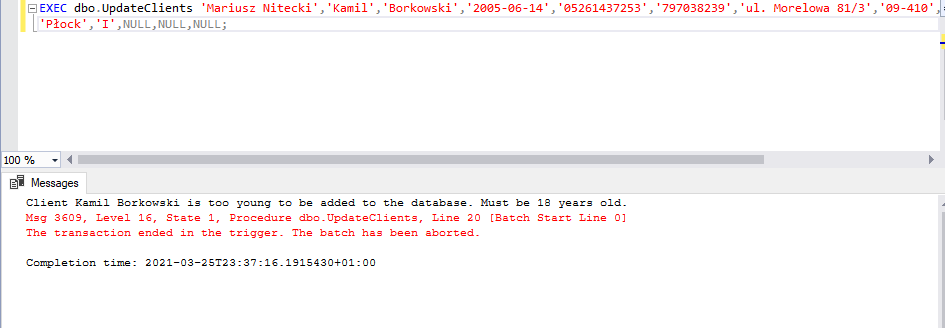
Ostatnią funkcjonalnością, jaka nam została do przetestowania to przeglądanie klientów. Klienta dostępnego w systemie będziemy mogli wyszukać po ciągach znaków zawierających imię i nazwisko lub po samym imieniu i nazwisku. Tym razem dostęp do tej funkcjonalności będą mieli tylko i wyłącznie pracownicy wypożyczalni. Po wybraniu opcji nowego zapytania w połączonej instancji serwera na koncie *Employee* wykonujemy zapytanie do bazy, które za pomocą instrukcji EXEC uruchamia procedurę *BrowseClients* (Rysunek 3.20). Procedura ta przyjmuje tylko jeden parametr, a dokładnie imię i nazwisko klienta, który jest dostępny w wypożyczalni. Przykładowo pracownik chce sprawdzić, czy w wypożyczalni jest zarejestrowany *‘Igor Krajewski’*. Po wpisaniu danych klienta, klikamy przycisk *Execute* w celu uruchomienia procedury. Jak to zostało pokazane na poniższym rysunku system wyszukał klienta o takim imieniu i nazwisku.



Rysunek 3.20. Wykonanie procedury "BrowseClients" pozwalające na przeglądanie klientów w bazie danych.

Źródło: opracowanie własne

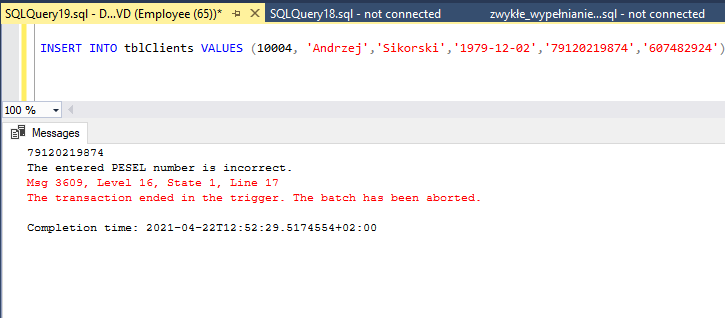
Jeżeli wszystkie funkcjonalności przedstawione w systemie działają bez większych problemów, to pozostaje nam przetestować błędy jakie mogą zostać wykryte przez system. Jednym z takich błędów, jest zapisanie osoby niepełnoletniej do tabeli klientów świadczących usługi w wypożyczalni. Zasada jaka została przyjęta przez wypożyczalnię jest taka, że klient. aby mógł się zarejestrować, musi mieć ukończone 18 lat. Jeżeli pracownik wykona zapytanie do bazy dodając nowego klienta przy pomocy polecenia INSERT INTO lub procedury składowanej i wykona takie zapytanie, to system powinien wyświetlić odpowiedni komunikat. Na wykonanie takich czynności służą między innymi wyzwalacze, które zostały przeze mnie omówione w punkcie 2.4.6.5. Aby przetestować działanie wyzwalacza, na koncie *Employee* wykonujemy nowe zapytanie, w którym uruchamiamy procedurę *UpdateClients*, gdzie spróbujemy dodać nowego klienta, który nie ma skończonych 18 lat (Rysunek 3.21). W parametrach procedury podajemy odpowiednie wartości, które będą odpowiadały za wykonanie operacji INSERT. Do czwartego parametru procedury podajemy datę urodzenia klienta, którego wiek jest mniejszy od 18 lat. Wykonujemy zapytanie za pomocą przycisku *Execute*, gdzie system wyświetli komunikat informujący klienta, że jest za młody, aby go zapisać do tabeli *Clients*.



Rysunek 3.21. Wykonanie procedury "UpdateClients", test działania wyzwalacza "tr\_checkPersonIsAnAdult".

Źródło: opracowanie własne

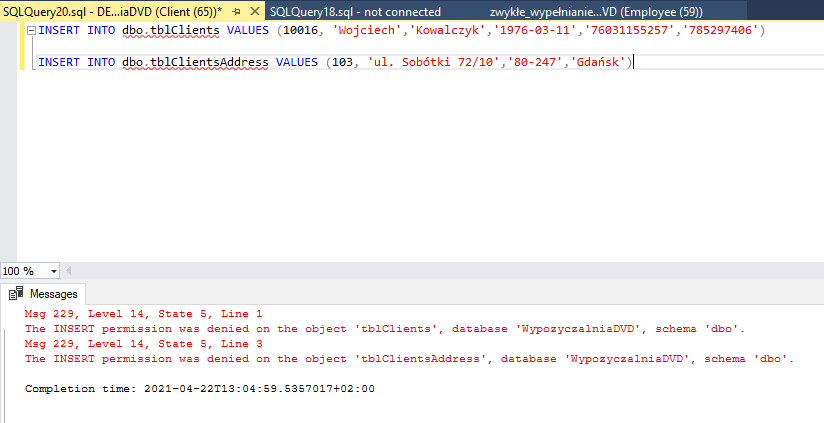
Kolejnym błędem jaki może wystąpić w bazie danych, to nieprawidłowy numer PESEL, podany przez klienta. W przypadku tego rodzaju usterki pracownik musi sprawdzić, czy prawidłowo wpisał numer PESEL. Weryfikacja tego numeru w bazie danych odbywa się przy pomocy wyzwalacza *tr\_ IsPESELOK*. Wyzwalacz zostanie wykonany w tabeli *Clients* po wykonaniu operacji dodawania lub aktualizowania rekordu. Wewnątrz kodu wyzwalacza, zostanie użyta funkcja CHECKPESEL sprawdzająca poprawność wpisanego numeru PESEL. Funkcja będzie zwracała typ danych INT. Wyzwalacz przy pomocy instrukcji warunkowej sprawdza czy wartość funkcji CHECKPESEL jest równa 0. Jeżeli funkcja zwróci taką wartość, to system wyświetli komunikat o poprawnym wpisaniu numeru PESEL. W tej sytuacji klient może zostać zarejestrowany w bazie danych. W przeciwnym wypadku system wyświetli komunikat o wpisaniu błędnego numeru PESEL. Wtedy pracownik musi poinformować klienta, aby się skontaktował z odpowiednim urzędem, w celu jego zmiany. Aby przetestować ten rodzaj błędu wykonujemy zapytanie, gdzie za pomocą instrukcji INSERT spróbujemy dodać do tabeli klienta z błędnym numerem PESEL. Jak możemy zauważyć na Rysunku 3.22, system przesłał odpowiednie komunikaty wraz z wywołaniem błędu o przerwanej transakcji za pomocą wyzwalacza. Podczas dodawania nowego klienta do bazy, jest wyświetlany numer PESEL. Drugi komunikat wyświetla informację czy PESEL podany przez pracownika jest prawidłowy, czy nieprawidłowy. Jak możemy zauważyć na poniższym przykładzie dane klienta są nieprawidłowe, ponieważ podano błędny numer PESEL.



Rysunek 3.22. Wykonanie zapytania dodające nowego klienta do tabeli, test działania wyzwalacza "tr\_IsPESELOK".

Źródło: opracowanie własne

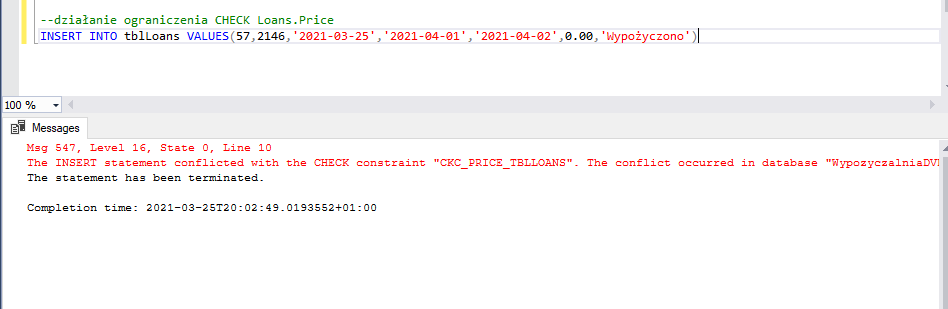
Gdyby się zdarzyła sytuacja, że klient chce wykonać w bazie zapytanie, do którego nie posiada uprawnień, to system powinien obsłużyć taki rodzaj błędu. Aby dany użytkownik mógł w pełni wykonać daną operację na obiektach bazy danych, musi posiadać odpowiednie uprawnienia. Podczas tworzenia obiektów baz danych, należy w sposób jawny udzielić uprawnień, aby były dostępne dla użytkowników. Każdy zabezpieczony obiekt ma uprawnienia, które mogą zostać przyznane podmiotowi zabezpieczeń przy użyciu instrukcji uprawnień. W celu przetestowania zabezpieczeń na koncie *Client* spróbujemy dodać nowego klienta do tabeli. W eksploratorze obiektów wybieramy instancję serwera, z którym jest połączone konto *Client* i klikamy prawym przyciskiem myszy na opcję tworzenia nowego zapytania. Wykonujemy zapytania za pomocą instrukcji INSERT, gdzie na tym koncie, spróbujemy dodać dane klienta do tabel *Clients* i *ClientsAddress* wstawiając odpowiednie wartości (Rysunek 3.23). Jeżeli zapytania zostaną wykonane, to system wyświetli komunikaty o błędzie. Błędy te są związane z odmową uprawnień dla operacji INSERT dla tabel *Clients* i *ClientsAddress*. Użytkownik *Client* nie posiada uprawnień do wykonywania operacji INSERT jak i do wymienionych wcześniej tabel. Posiada on jedynie uprawnienia do wykonywania operacji SELECT na obiektach dotyczących filmów. Jeżeli administrator chce nadać uprawnienia użytkownikowi *Client* do wykonywania operacji SELECT, to musiałby skorzystać z instrukcji GRANT.



Rysunek 3.23. Próba wykonania zapytania dodającego nowego klienta do tabeli na koncie "Client".

Źródło: opracowanie własne

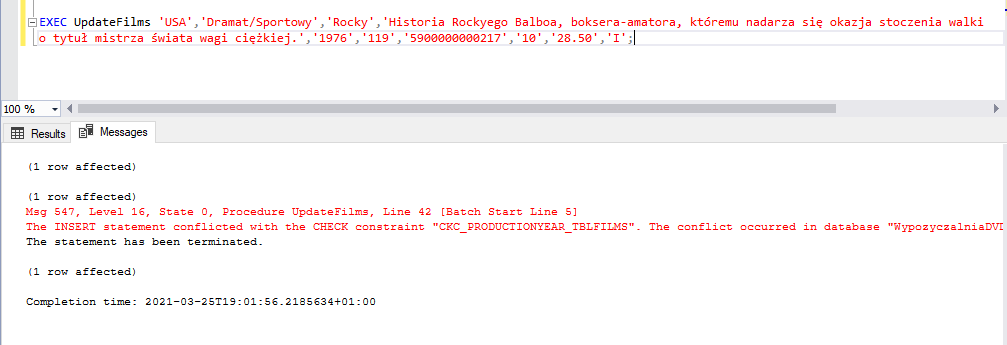
Kolejnym błędem, jaki może wystąpić w bazie danych, to podanie wartości w kolumnie wychodzącej poza ograniczenie CHECK. Ten rodzaj błędu może wystąpić, gdy użytkownik poda wartość przekraczającą zakres, jaki został nałożony na kolumnę w tabeli. Zabezpieczenie się przed wprowadzeniem nieprawidłowych danych, jest bardzo ważnym elementem systemu, ponieważ pomaga nam zachować integralność danych. Aby sprawdzić poprawność działania tego błędu wykonujemy zapytanie w bazie, gdzie do tabeli *Loans* spróbujemy dodać w kolumnie *Price* wartość 0. W oknie zapytań wykonujemy instrukcję INSERT, gdzie do tabeli *Loans* wstawiamy odpowiednie wartości dla poszczególnych kolumn. Jak możemy zauważyć na Rysunku 3.24 próba wstawienia nowego rekordu zakończyła się niepowodzeniem. System wyświetlił błąd o numerze 547, który mówi, że dla instrukcji INSERT wystąpił konflikt z ograniczeniem CHECK. Stało się tak, ponieważ do kolumny *Price* tabeli *Loans* próbowaliśmy wstawić wartość, która nie spełnia warunku, jakie przyjmuje to ograniczenie. Aby zapisać wypożyczony film w bazie, kolumna *Price* tabeli *Loans* musi przyjmować wartości większe od „0.00”.



Rysunek 3.24. Próba wykonania operacji INSERT w tabeli "Loans", wyświetlenie komunikatu dla ograniczenia CHECK.

Źródło: opracowanie własne

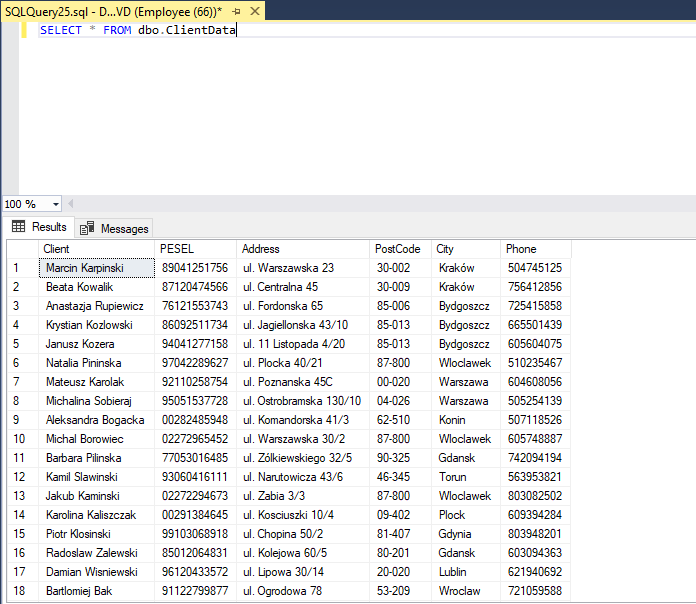
Aby się upewnić, że w systemie jest zachowana integralność danych, wykonujemy zapytanie, pozwalające na uruchomienie procedury *UpdateFilms*. Wpisujemy odpowiednie wartości dla poszczególnych parametrów, gdzie dla piątego parametru ustawiamy wartość na 1976 rok. Jak możemy zauważyć na rysunku 3.25 film nie został dodany do bazy danych, ponieważ w kolumnie *ProductionYear* tabeli *Films* wstawiono wartość wychodzącą poza zakres. Jeżeli chcemy dodać nowy film do bazy, kolumna *ProductionYear* musi przyjmować wartości dla roku produkcji filmu pomiędzy 1980, a 2100 rokiem.



Rysunek 3.25. Próba wykonania procedury "UpdateFilms", wyświetlenie komunikatu o błędzie dla ograniczenia CHECK.

Źródło: opracowanie własne

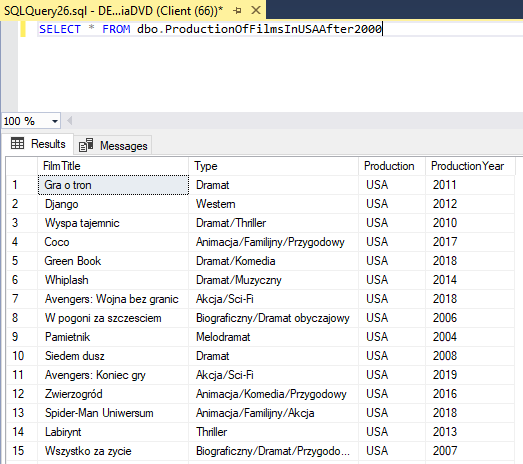
Ostatnią rzeczą, jaką trzeba sprawdzić w systemie, to wyświetlenie zawartości widoków. Widoki pozwalają na zapisanie w bazie danych zapytań, które pobierają dane z jednej lub wielu encji (tabel). Do widoków odwołujemy się w taki sam sposób, jak do tabel. W celach testowych wyświetlimy zawartość jednego widoku, gdzie uprawnienia do jego wykonania posiada użytkownik *Employee*. Jednym z takich widoków jest widok, pozwalający pracownikowi wyświetlić zawartość wszystkich klientów, którzy świadczą usługi w wypożyczalni. Widok będzie wyświetlał takie dane jak imię i nazwisko klienta, PESEL, adres zamieszkania oraz numer telefonu. Na koncie *Employee* wykonujemy kod SQL, który za pomocą instrukcji SELECT, wyświetli wszystkie rekordy i kolumny z widoku *ClientData* (Rysunek 3.26). Po wykonaniu kodu kwerendy, wyświetlana jest zawartość widoku z danymi wszystkich klientów.



Rysunek 3.26. Zawartość widoku "ClientData".

Źródło: opracowanie własne

Widok, który zostanie wykonany na koncie *Clients* będzie wyświetlał wszystkie filmy, które zostały wyprodukowane w USA po 2000 roku. Dane dotyczące tych filmów, mogą przypaść do gustu klientom, którzy są fanatykami amerykańskiego kina oraz lubią oglądać nowsze filmy. Widok będzie wyświetlał takie dane jak tytuł filmu, gatunek, kraj oraz rok produkcji. W eksploratorze obiektów programu Microsoft SQL Server Management Studio należy wybrać instancję serwera, z którym jest połączone konto *Client*. Po wybraniu instancji serwera klikamy prawym przyciskiem myszy, aby otworzyć okno, pozwalające na wykonanie nowego zapytania. W oknie zapytań należy wykonać kwerendę, gdzie przy użyciu instrukcji SELECT, wyświetlamy wszystkie kolumny z widoku *ProductionOfFilmsInUSAAfter2000*. Jak możemy zauważyć na rysunku 3.27 po wykonaniu zapytania została wyświetlona zawartość widoku, zawierająca filmy, które zostały przefiltrowane według określonych kryteriów.



Rysunek 3.27. Zawartość widoku "ProductionOfFilmsInUSAAfter2000".

Źródło: opracowanie własne

# Podsumowanie

Celem pracy było zaprojektowanie bazy danych opartej na modelu UML. System został utworzony na podstawie diagramów klas, konceptualnego i fizycznego za pomocą programu PowerDesigner. Zastosowanie programu PowerDesigner może w znaczny sposób ułatwić projektowanie oraz implementację baz danych. Program zawiera wszelkie funkcje wbudowane obsługujące większość nowoczesnych systemów zarządzania relacyjnymi bazami danych (RDBMS). Może także współpracować z innymi językami programowania takimi jak Java czy C#.

Proces projektowania poprzedza zbieranie wymagań, gdzie należy określić wymagania funkcjonalne i niefunkcjonalne jakie ma spełnić tworzony system. Wymagania będą powiązane ze scenariuszami, które opisują docelowy sposób użytkowania projektowanej bazy danych. Scenariusze mogą zostać przedstawione w sposób graficzny za pomocą diagramów UML takich jak diagramy przypadków użycia, aktywności i sekwencji. Do utworzenia tych diagramów został użyty program Microsoft Visio, który służy m.in. do tworzenia tego typu diagramów. Microsoft Visio pozwala uwzględnić w diagramach UML dane, które będą niezbędne do realizowania planowanych czynności.

Modelowanie struktur danych jest wspomagane przez funkcje modelu klas. Na diagramie klas są tworzone klasy, które będą identyfikowały poszczególne obiekty w systemie. Następnie tworzymy atrybuty określające zestaw wartości, które może przyjmować klasa. Jeżeli poszczególne klasy będą wykonywały określone funkcje, to należy utworzyć operacje, które są funkcjami dostarczanymi przez obiekt, deklarowane przez odpowiednie jego zachowanie. Na samym końcu są uwzględniane związki, które występują pomiędzy obiektami.

Gotowy model klas za pomocą mechanizmów generowania modeli jest przekształcony do modelu konceptualnego. W modelu konceptualnym następuje sklasyfikowanie elementów danych, które zostały wygenerowane za pomocą diagramu klas. Czynności te polegają na przypisaniu danych do odpowiednich encji. Pod uwagę zostaną wzięte również związki, które występują pomiędzy obiektami, reprezentowanymi przez encje.

Jeżeli model konceptualny jest poprawnie wykonany, to jest przekształcany na model fizyczny dla docelowego systemu zarządzania bazą danych. Model fizyczny zezwala na uwzględnienie w projekcie szczegółowych wymagań, dotyczących zasad realizacji poszczególnych operacji w bazie danych. W większości przypadków projektowanie elementów takich jak widoki, procedury składowane czy wyzwalacze wymagają znajomości języka SQL oraz pisania części kodu. Jednak większość obiektów tworzących bazę danych może zostać zaprojektowana w sposób graficzny przez wstawienie odpowiednich symboli na diagramie i wybranie kilku opcji określających właściwości tych obiektów.

Funkcje programu PowerDesigner znacznie ułatwiają i przyspieszają wykonywanie czynności, które są związane z procesem implementacji bazy danych. Mowa tutaj o funkcji generowania skryptu SQL na podstawie opracowanego modelu fizycznego. Po wygenerowaniu modelu fizycznego, możemy gotowy skrypt uruchomić w programie Microsoft SQL Server, gdzie zostanie utworzona struktura bazy danych zawierającą pustą zawartość tabel.

W pracy został wykonany projekt bazy danych dla systemu wypożyczalni płyt DVD. Założenia, które podałem w podrozdziale związanym z wymaganiami systemu baz danych zostały spełnione. Utworzona baza danych będzie umożliwiała wypożyczanie i oddawanie filmów przez klientów oraz zarządzanie filmami i klientami przez pracowników. Gotowa baza danych może zostać wykorzystana w przyszłości przez dowolną wypożyczalnię, która będzie chciała przechowywać filmy, klientów, pracowników, aktorów i role aktorów w systemach zarządzania bazami danych takich jak MySQL czy Microsoft SQL Server. Może ona zostać również wykorzystana przez programistów aplikacji internetowych czy desktopowych, do zaprojektowania systemu łączącego się z bazą danych, gdzie system zostanie wzbogacony o interfejs użytkownika.

# Bibliografia

[1] Benyon-Davies, P. (2001). *Systemy Baz Danych.* Warszawa: Wydawnictwa Naukowo – Techniczne.

[2] Bertino, E. i Martino, L. (1993). *Object – Oriented Database Systems Concepts and Architectures.* Addison – Wesley Publishing Company.

[3] Booch, G., Rumbaugh, J. i Jacobson, I. (2005). *The Unified Modelling Language User Guide Second Edition.* Addison Wesley .

[4] Connolly, T. i Begg, C. (2005). *Database Systems A Practical Approach to Design, Implementation, and Management.* Pearson.

[5] Date, C. J. (2000). *Wprowadzenie do systemów baz danych.* Warszawa: Wydawnictwa Naukowo – Techniczne.

[6] Fowler, M. i Scott, K. (2005). *UML w kropelce.* Wydawnictwo LTP.

[7] Garcia-Mollina, H., Ullman, J. i Widom, J. (2002). *Database Systems The Complete Book.* Pearson.

[8] *ii.uwb.edu.pl*. (2020, Luty 26). Pobrano z lokalizacji Witryna sieci Web UWB: http://ii.uwb.edu.pl/~aboltuc/images/stories/wykad\_1.pdf

[9] Ling, T. W., Lee, M. L. i Dobbie, G. (2005). *Semistructured Database Design.* Springer.

[10] *Microsoft SQL Server*. (2020, Maj 23). Pobrano z lokalizacji Witryna sieci Web SearchSQLServer: https://searchsqlserver.techtarget.com/definition/SQL-Server

[11] Microsoft Visio. (2020, Kwiecień 15). Pobrano Kwiecień 15, 2020 z lokalizacji https://products.office.com/pl-pl/visio/flowchart-software

[12] *Object Management Group - specyfikacja UML*. (2000, Marzec). Pobrano z lokalizacji Witryna sieci Web Object Management Group: https://www.omg.org/spec/UML/1.3/PDF

[13] Pękała, B. (2015). *Bazy danych Teoria i praktyka.* Rzeszów: Wydawnictwo Uniwersytetu Rzeszowskiego.

[14] PowerDesigner. (2004). *Physical Data Model User's Guide.* Sybase Inc.

[15] PowerDesigner. (2005). *Conceptual Data Model User's Guide.* Sybase Inc.

[16] PowerDesigner. (2011). *Object-Oriented Modeling.* Sybase Inc.

[17] PowerDesigner. (2020, Kwiecień 15). Pobrano Kwiecień 15, 2020 z lokalizacji https://www.powerdesigner.biz/EN/powerdesigner/powerdesigner-features.html#history

[18] Rumbaugh, J., Jacobson, I. i Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Addison – Wesley.

[19] Sherman, R. (2015). Business Intelligence Guidebook: From Data Integration to Analytics. Morgan Kaufmann.

[20] Taylor, A. G. (2003). *SQL for Dummies.* Willey Publishing, Inc.

[21] West, M. (2011). Developing High Quality Data Models. Morgan Kaufmann.

# Spis tabel i rysunków

***Spis tabel:***

[Tabela 1.1. Lista typów danych użytych w systemie wypożyczalni płyt DVD. 41](#_Toc72767264)

[Tabela 2.2. Standardowe typy danych zastosowane w modelu konceptualnym dla systemu wypożyczalni płyt. 84](#_Toc72767281)

[Tabela 2.3. Związki w modelu związków encji dla systemu wypożyczalni płyt DVD. 95](#_Toc72767290)

[Tabela 2.4. Transformacja obiektów CDM na PDM. 126](#_Toc72767312)

***Spis rysunków:***

[Rysunek 2.1. Diagram związków encji przedstawiający relacyjną bazę danych dla potrzeb wypożyczalni płyt DVD. 40](#_Toc72767263)

[Rysunek 2.2. Przykładowa tabela zawierająca dane filmów wraz z atrybutami znajdująca się w bazie danych. 46](#_Toc72767265)

[Rysunek 2.3. Diagram przypadków użycia dla systemu bazy danych zarządzającą wypożyczalnią DVD. 51](#_Toc72767266)

[Rysunek 2.4. Diagram aktywności dla funkcji wypożyczenia filmów w systemie wypożyczalni płyt DVD. 53](#_Toc72767267)

[Rysunek 2.5. Diagram sekwencji dla funkcji dodawania filmów w systemie wypożyczalni płyt DVD. 58](#_Toc72767268)

[Rysunek 2.6. Lista atrybutów zdefiniowanych w klasie Films. 63](#_Toc72767269)

[Rysunek 2.7. Właściwości atrybutu FilmTitle w klasie Films. 64](#_Toc72767270)

[Rysunek 2.8. Lista operacji zdefiniowanych w klasie Films. 66](#_Toc72767271)

[Rysunek 2.9. Właściwości ogólne asocjacji "Film\_Production". 67](#_Toc72767272)

[Rysunek 2.10. Szczegóły asocjacji dla właściwości "Film\_Production". 69](#_Toc72767273)

[Rysunek 2.11. Właściwości ogólne asocjacji "Role\_In\_Film". 70](#_Toc72767274)

[Rysunek 2.12. Diagram klas UML dla systemu wypożyczalni płyt DVD. 71](#_Toc72767275)

[Rysunek 2.11. Okno z parametrami sprawdzania modelu na przykładzie modelu klas UML. 75](#_Toc72767276)

[Rysunek 2.14. Lista wyników sprawdzania modelu na przykładzie diagramu klas. 76](#_Toc72767277)

[Rysunek 2.15. Okno opcji generowania modelu konceptualnego. 78](#_Toc72767278)

[Rysunek 2.16. Symbol encji "Clients" występujący w modelu konceptualnym systemu wypożyczalni płyt DVD. 81](#_Toc72767279)

[Rysunek 2.17. Lista atrybutów encji Loans. 82](#_Toc72767280)

[Rysunek 2.18. Okno definicji parametrów sprawdzających poprawność danych na przykładzie atrybutu Duration. 85](#_Toc72767282)

[Rysunek 2.19. Lista identyfikatorów encji Loans w modelu konceptualnym. 87](#_Toc72767283)

[Rysunek 2.20. Okno właściwości identyfikatora IDLoan. 88](#_Toc72767284)

[Rysunek 2.21. Symbol związku encji Production -> Films w systemie wypożyczalni płyt DVD. 89](#_Toc72767285)

[Rysunek 2.22. Okno właściwości związku Films->Production. 90](#_Toc72767286)

[Rysunek 2.23. Związek encji z rolą dominującą Clients-> ClientsAddress. 91](#_Toc72767287)

[Rysunek 2.24. Związek zależny Films -> Role w systemie wypożyczalni płyt DVD. 93](#_Toc72767288)

[Rysunek 2.25. Diagram konceptualny dla systemu wypożyczalni płyt DVD. 94](#_Toc72767289)

[Rysunek 2.26. Okno opcji generowania modelu fizycznego. 97](#_Toc72767291)

[Rysunek 2.27. Symbol tabeli w modelu fizycznym PDM, na przykładzie tabeli Films. 99](#_Toc72767292)

[Rysunek 2.28. Okno właściwości tabeli "Films" w modelu fizycznym PDM. 100](#_Toc72767293)

[Rysunek 2.29. Okno właściwości kolumny "FilmTitle" w tabeli "Films". 102](#_Toc72767294)

[Rysunek 2.30. Lista kluczy tabeli "Films". 104](#_Toc72767295)

[Rysunek 2.31. Okno właściwości klucza "IDFilm" w tabeli Films". 105](#_Toc72767296)

[Rysunek 2.32. Lista indeksów zdefiniowanych w tabeli "Films". 106](#_Toc72767297)

[Rysunek 2.33. Okno właściwości indeksu PK\_FILMS dla tabeli Films, lista kolumn. 107](#_Toc72767298)

[Rysunek 2.34. Referencja zachodząca pomiędzy tabelami "Loans" i "Films". 108](#_Toc72767299)

[Rysunek 2.35. Okno właściwości referencji Loans\_Films. 109](#_Toc72767300)

[Rysunek 2.36. Okno właściwości referencji "Loans\_Films, lista powiązań kolumn. 110](#_Toc72767301)

[Rysunek 2.37. Okno właściwości referencji "Loans\_Films", opcje więzów integralności. 112](#_Toc72767302)

[Rysunek 2.38. Okno właściwości widoku "LoanFilms" zawierający zapytanie SQL. 114](#_Toc72767303)

[Rysunek 2.39. Symbol widoku w modelu fizycznym na przykładzie "LoanFilms". 115](#_Toc72767304)

[Rysunek 2.40. Symbol procedur w diagramie PDM. 117](#_Toc72767305)

[Rysunek 2.41. Definicja procedury "BrowseFilms". 117](#_Toc72767306)

[Rysunek 2.42. Okno właściwości tabeli "Clients", lista wyzwalaczy. 119](#_Toc72767307)

[Rysunek 2.43. Definicja wyzwalacza sprawdzającego pełnoletniość klienta. 120](#_Toc72767308)

[Rysunek 2.44. Okno właściwości użytkownika "Employee", uprawnienia dotyczące tabel. 123](#_Toc72767309)

[Rysunek 2.45. Uprawnienia użytkowników na przykładzie tabeli Actors w oknie właściwości tabeli. 124](#_Toc72767310)

[Rysunek 2.46. Okno właściwości roli "ClientLoanFilms", lista użytkowników przypisanych do roli. 125](#_Toc72767311)

[Rysunek 2.47. Transformacja związku jeden do wielu: a) CDM, b) PDM. 127](#_Toc72767313)

[Rysunek 2.48. Transformacja związku jeden do jednego dla roli dominującej: a) CDM, b) PDM. 128](#_Toc72767314)

[Rysunek 2.49. Transformacja związku zależnego: a) CDM, b) PDM. 129](#_Toc72767315)

[Rysunek 2.50. Model fizyczny dla systemu wypożyczalni DVD. 130](#_Toc72767316)

[Rysunek 2.51. Zapytanie SQL pozwalające na utworzenie nowej bazy danych. 131](#_Toc72767317)

[Rysunek 2.52. Określenie nazwy i wybór serwera w kreatorze nowego źródła danych ODBC. 132](#_Toc72767318)

[Rysunek 2.53. Określenie weryfikacji połączenia z SQL Server i podanie loginu oraz hasła w kreatorze źródła danych ODBC. 133](#_Toc72767319)

[Rysunek 2.54. Okno pozwalające na ustawienie domyślnej bazy danych w kreatorze źródła danych ODBC. 134](#_Toc72767320)

[Rysunek 2.55 Okno dokonywania ustawień dla nowego źródła danych ODBC. 134](#_Toc72767321)

[Rysunek 2.56. Okno zawierające podsumowanie konfiguracji ODBC dla Microsoft SQL Server. 135](#_Toc72767322)

[Rysunek 2.57. Okno zawierające podsumowanie konfiguracji źródła ODBC wraz z testem połączenia. 136](#_Toc72767323)

[Rysunek 2.58. Okno opcji generowania bazy danych. 137](#_Toc72767324)

[Rysunek 2.59. Okno połączenia się z źródłem danych ODBC. 138](#_Toc72767325)

[Rysunek 2.60. Okno skryptu SQL. 139](#_Toc72767326)

[Rysunek 2.61. Widok tabel w bazie danych WypozyczalniaDVD w Microsoft SQL Server Management Studio. 140](#_Toc72767327)

[Rysunek 3.1. Wykonanie procedury "UpdateClients" dodającą nowego klienta do tabel Clients i ClientsAddress. 143](#_Toc72767328)

[Rysunek 3.2. Zawartość tabel "Clients" i "ClientsAddress" po dodaniu klienta do bazy danych. 144](#_Toc72767329)

[Rysunek 3.3. Wykonanie procedury "UpdateClients" modyfikująca dane klienta w tabelach Clients i ClientsAddress. 145](#_Toc72767330)

[Rysunek 3.4. Zawartość tabel "Clients" i "ClientsAddress" po zmodyfikowaniu klienta. 145](#_Toc72767331)

[Rysunek 3.5. Wykonanie procedury "UpdateClients" usuwająca dane klienta z bazy danych. 146](#_Toc72767332)

[Rysunek 3.6. Zawartość tabel "Clients" i "ClientsAddress" po usunięciu klienta z bazy danych. 147](#_Toc72767333)

[Rysunek 3.7. Wykonanie procedury "UpdateFilms" dodająca nowy film do bazy danych. 148](#_Toc72767334)

[Rysunek 3.8. Zawartość tabel "Films", "Production" i "Type" po dodaniu nowego filmu. 149](#_Toc72767335)

[Rysunek 3.9. Wykonanie procedury "UpdateFilms" modyfikująca dany film w tabeli. 150](#_Toc72767336)

[Rysunek 3.10. Zawartość tabel "Films", "Production" i "Type" po zmodyfikowaniu filmu w bazie danych. 150](#_Toc72767337)

[Rysunek 3.11. Wykonanie procedury "UpdateFilms" usuwająca wybrany film z tabeli. 151](#_Toc72767338)

[Rysunek 3.12. Wykonanie procedury "LoanFilms" pozwalająca na wypożyczenie filmu przez klienta. 152](#_Toc72767339)

[Rysunek 3.13. Zawartość tabeli "Films" po operacji wypożyczenia filmu przez klienta. 153](#_Toc72767340)

[Rysunek 3.14. Zawartość tabeli "Loans" po wypożyczeniu filmu przez klienta. 154](#_Toc72767341)

[Rysunek 3.15. Wykonanie procedury "LoansFilms" wraz z wyświetleniem komunikatu o zablokowaniu wypożyczenia filmu. 155](#_Toc72767342)

[Rysunek 3.16. Wykonanie procedury "ReturnFilms" pozwalająca na oddawanie filmu przez klienta. 156](#_Toc72767343)

[Rysunek 3.17. Wykonanie procedury "ReturnFilms" dla filmu oddawanego po terminie. 156](#_Toc72767344)

[Rysunek 3.18. Zawartość tabel "Films" i "Loans" po oddaniu filmów przez klientów. 157](#_Toc72767345)

[Rysunek 3.19. Wykonanie procedury "BrowseFilms" pozwalająca na przeglądanie filmów według określonych kryteriów. 158](#_Toc72767346)

[Rysunek 3.20. Wykonanie procedury "BrowseClients" pozwalające na przeglądanie klientów w bazie danych. 159](#_Toc72767347)

[Rysunek 3.21. Wykonanie procedury "UpdateClients", test działania wyzwalacza "tr\_checkPersonIsAnAdult". 160](#_Toc72767348)

[Rysunek 3.22. Wykonanie zapytania dodające nowego klienta do tabeli, test działania wyzwalacza "tr\_IsPESELOK". 161](#_Toc72767349)

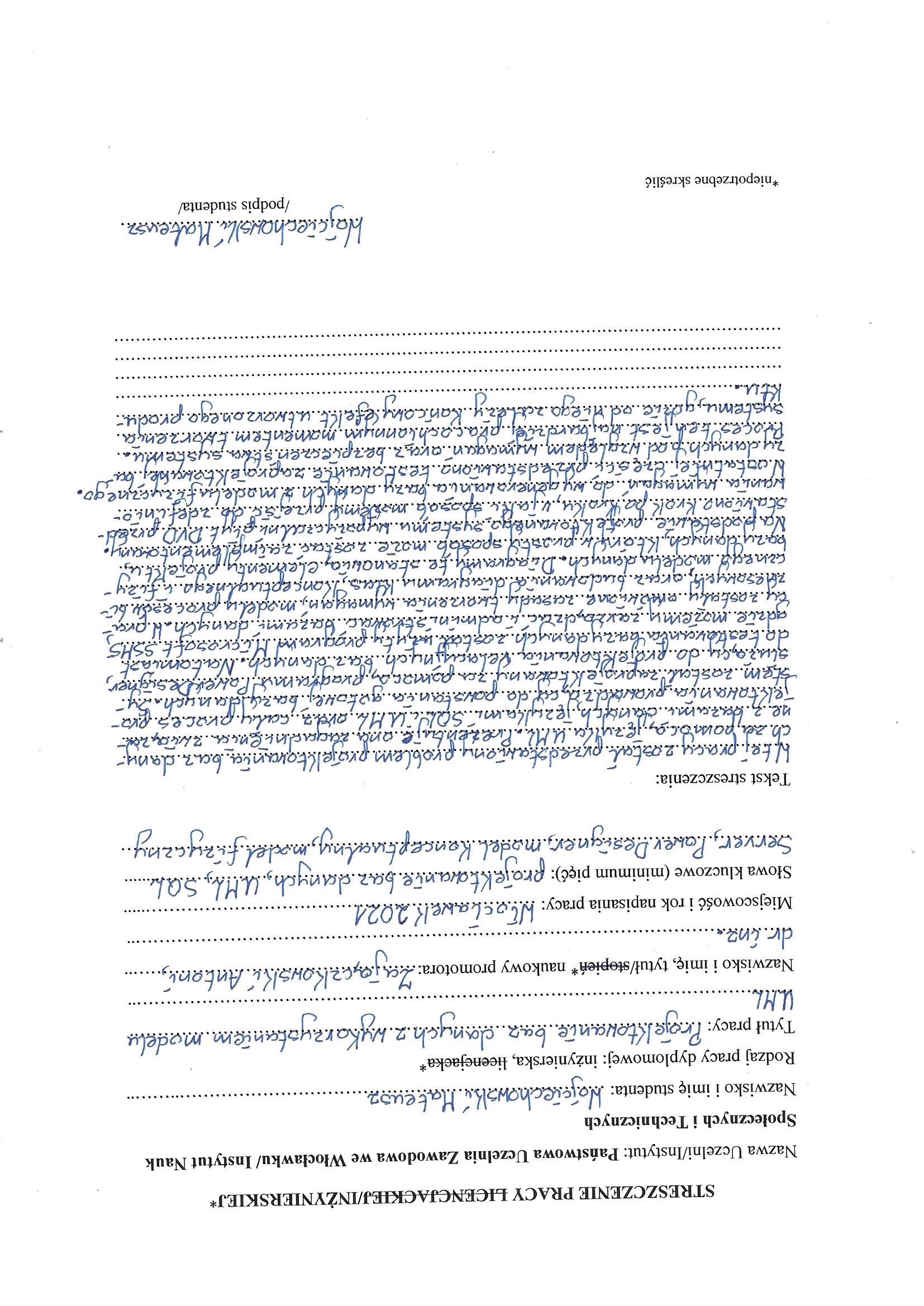
[Rysunek 3.23. Próba wykonania zapytania dodającego nowego klienta do tabeli na koncie "Client". 162](#_Toc72767350)

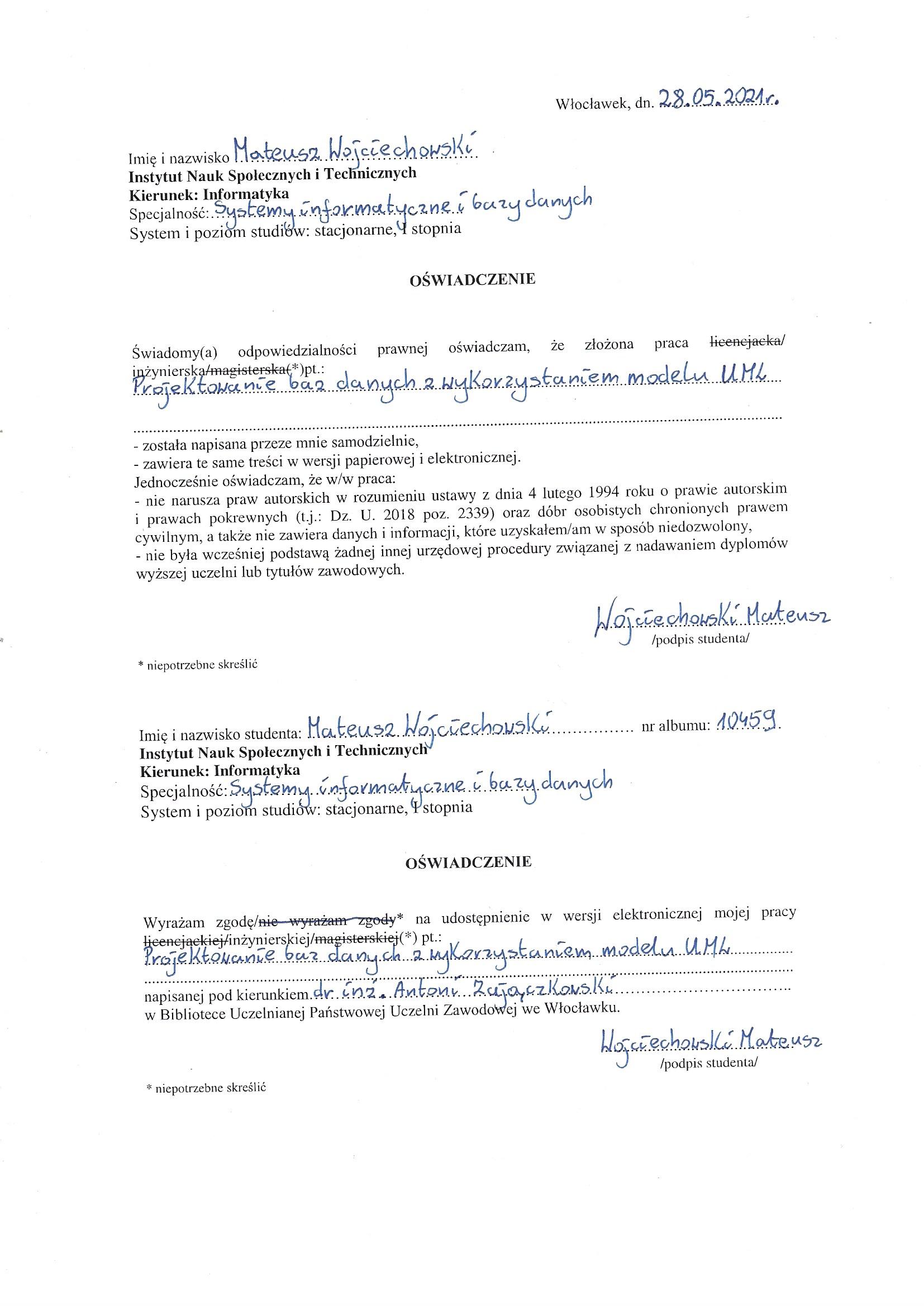
[Rysunek 3.24. Próba wykonania operacji INSERT w tabeli "Loans", wyświetlenie komunikatu dla ograniczenia CHECK. 163](#_Toc72767351)

[Rysunek 3.25. Próba wykonania procedury "UpdateFilms", wyświetlenie komunikatu o błędzie dla ograniczenia CHECK. 164](#_Toc72767352)

[Rysunek 3.26. Zawartość widoku "ClientData". 165](#_Toc72767353)

[Rysunek 3.27. Zawartość widoku "ProductionOfFilmsInUSAAfter2000". 166](#_Toc72767354)





1. ER – (ang. Entity – Relationship Model) - patrz podrozdział 1.11. [↑](#footnote-ref-1)