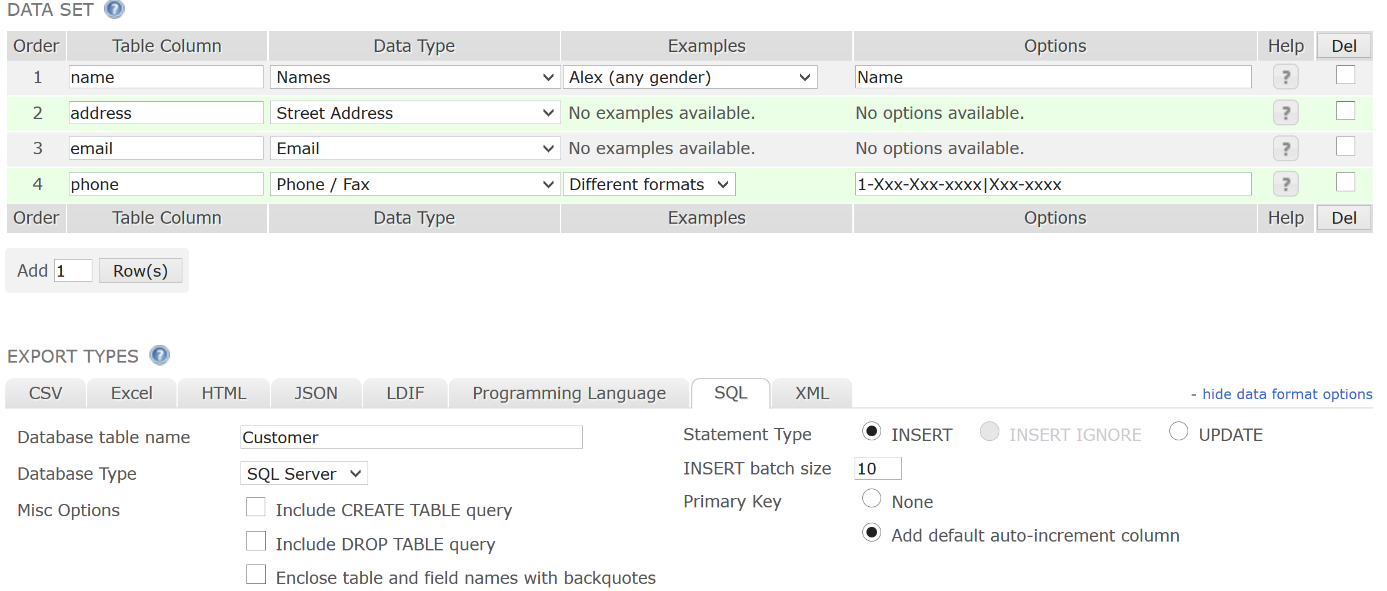
W projekcie została stworzona i zaimplementowana relacyjna baza danych, wraz z wypełnieniem jej przykładowymi danymi oraz ustawieniem ograniczeń w postaci dostępnych słów, jak i triggerów odpowiadających za automatyczne ustawianie wartości jak i zezwolenia na dane operacje.

Baza danych została zaimplementowana w programie *„SQL Server Management Studio”.* Zawiera ona osiem tabeli takich jak: *Category, Customer, Employee, Item, Model, Reservation, Transactions* oraz *TransactionItem*. Tabela *Model* została wypełniona przy użyciu danych pobranych ze strony internetowej[[1]](#footnote-1) przy wykorzystaniu tekstowej przeglądarki *lynx* dostępnej na systemie Linux CentOS. Przy jej użyciu wraz z prostymi skryptami shell-owymi, zostały stworzone pliki tekstowe, które później zaimportowane zostały do tabeli. Natomiast tabela *Customer* została wypełniona przy użyciu generatora danych[[2]](#footnote-2) dostępnych w sieci. Wszystkie inne zmiany dotyczące tabeli *Customer* były wprowadzane za pomocą generacji odpowiednich komend przy użyciu prostych programów napisanych w C++. Poniżej znajduje się przykładowy program, który generuje komendy SQL, dzięki którym losowo aktualizowane są świeżo dodane kolumny jakimi są *itemstatus* oraz *condition*.

|  |
| --- |
| int i**;**  srand**(**time**(**0**));**  int warunek **=** item **+** ilosc**;**  **for** **(**warunek**;** item **<=** warunek**;** item**++)** **{**  i **=** rand**()** **%** 10000**;**  cout **<<** i**;**  **if** **(**i **%** 3 **==** 0**)** **{**  plik2 **<<** "INSERT INTO [dbo].[Item] ([model\_id] ,[itemstatus] ,[condition]) VALUES  (" **<<** model **<<** ",'Available' ,'Good')" **<<** endl**;**  **}**  **else** **if** **(**i **%** 3 **==** 1**)** **{**  plik2 **<<** "INSERT INTO [dbo].[Item] ([model\_id] ,[itemstatus] ,[condition]) VALUES  (" **<<** model **<<** ",'Rent ' ,'Normal')" **<<** endl**;**  **}else**  plik2 **<<** "INSERT INTO [dbo].[Item] ([model\_id] ,[itemstatus] ,[condition]) VALUES  (" **<<** model **<<** ",'Available ' ,'Bad')" **<<** endl**;**  **}**  system**(**"pause"**);** |

Listing 1 Zautomatyzowanie generacji komend w SQL

Poniżej został także umieszczony zrzut ekranowy strony z której zostały wygenerowane dane dla tabeli *Customer*.

Rysunek 1 Zrzut ekranowy generatora danych

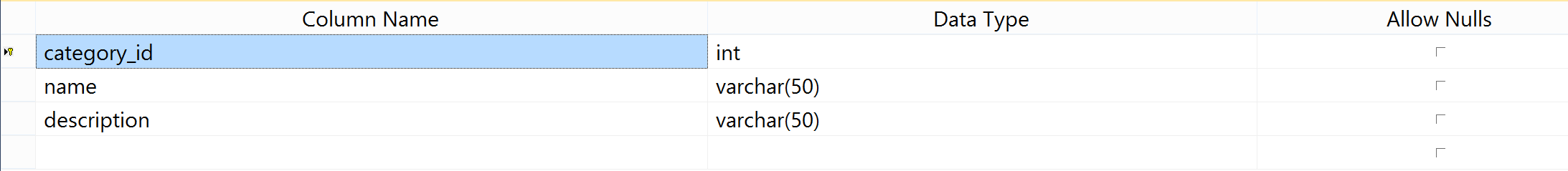
Dane w kolumnach takich jak *itemstatus* oraz *condition* w tabeli *Item* są ograniczone pod względem wpisywanych do nich wartości. Aby wykonać takie ograniczenia należy użyć komendy SQL -owej podanej poniżej. Niestety w SQL Server Management Studio nie występuje komenda *enum* dlatego w projekcie zostało to zastąpione komendą CHECK.

|  |
| --- |
| **ALTER** **TABLE** dbo**.**Item **ADD** **CONSTRAINT** check\_itemstatus  **CHECK** **(**itemstatus **IN(**'Rent'**,** 'Reserved'**,** 'Available'**))** |

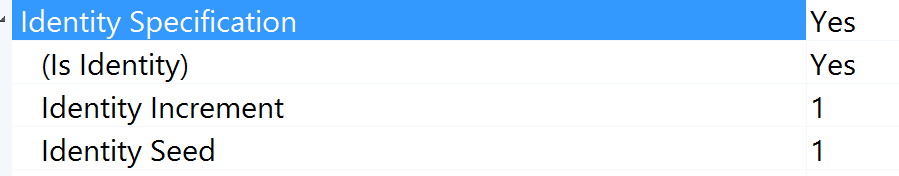
Listing 2 Przykładowa komenda, sprawdzająca poprawność wprowadzonych danych

Używając programu SQL Server Management Studio, zostały stworzone już wcześniej wymienione tabele. A cała baza danych jest znormalizowana do trzeciej postaci normalnej. Poniżej znajdują się wszystkie stworzone tabele wraz z opisem, oraz przykład ustawienia auto inkrementującego się klucza głównego.

* Tabela *Category*

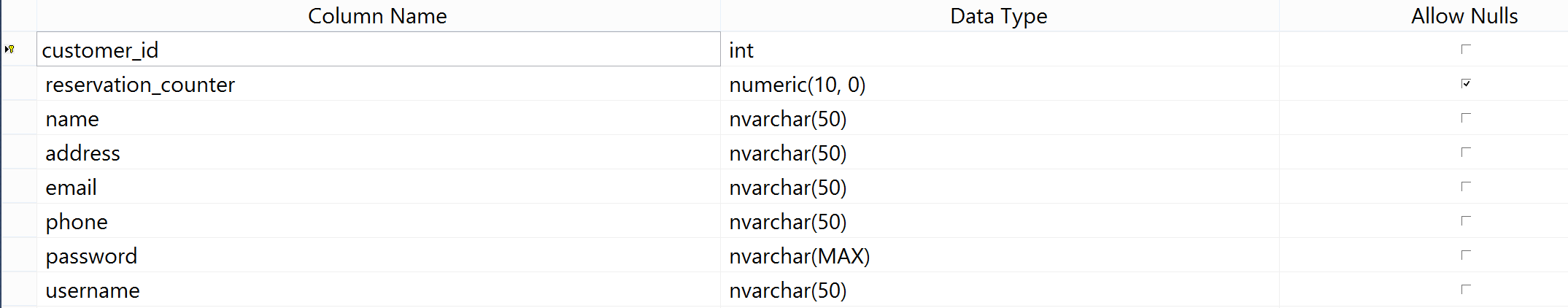
Posiada ona klucz główny *category\_id* oraz kolumny *name* i *description*. Przechowuje ona dane dotyczące kategorii sprzętu, który będziemy chcieli wypożyczyć. Kolumny odpowiednio *name* i *description* odpowiadają nazwie kategorii oraz jej opisie.

Rysunek 2 Tabela Category

Aby ustawić klucz główny, który sam się inkrementuje należy w ustawieniach w tym wypadku *category\_id* zmienić wartość opcji *Identity Specification* na *yes*. Poniżej znajduje się zrzut ekranu ustawionego klucza głównego który automatycznie się inkrementuje.

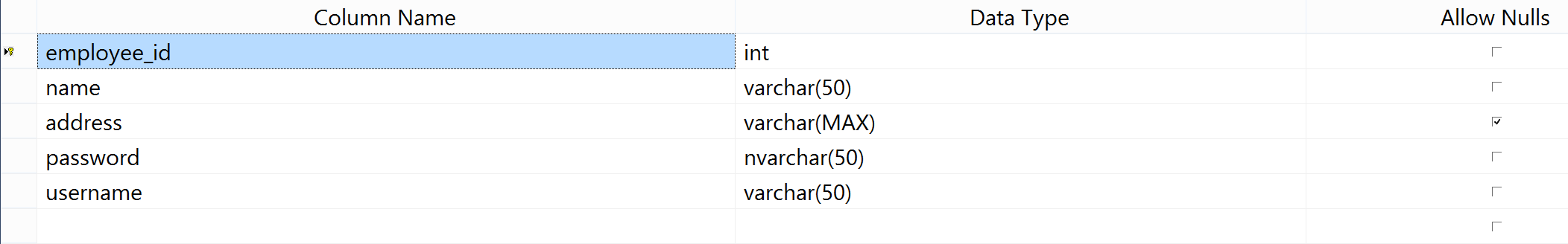
Rysunek 3 Ustawienie auto inkrementacji dla klucza głównego

* Tabela *Customer*

Posiada ona klucz główny *customer­\_id*, który jest analogicznie ustawiony jak ten z poprzedniej tabeli. Kolumny *name, address, email, phone, password* oraz *username* zawierają dane użytkownika. Na tym poziomie aplikacji również hasło nie jest szyfrowane ponieważ tym zajmuje się REST API. Kolumna *reservation\_counter* przechowuje informacje o tym ile dany użytkownik zarezerwował przedmiotów. Maksymalna liczba przedmiotów, które można zarezerwować wynosi 10.

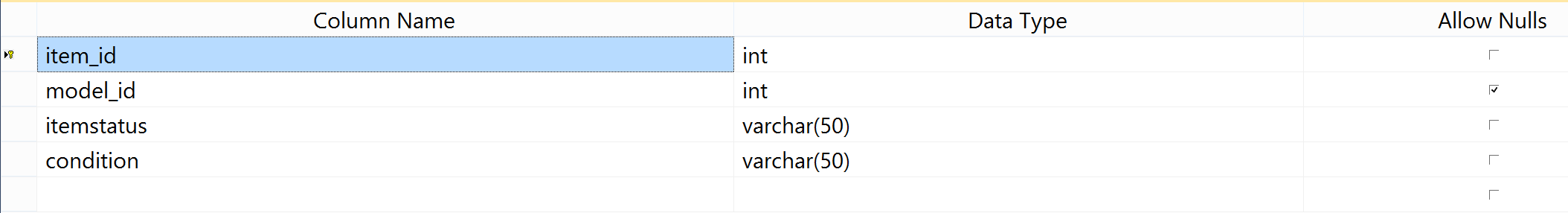
Rysunek Tabela Customer

* Tabela Employee

Bardzo podobna do tabeli Customer lecz nie przechowuje ani numeru telefonu ani adresu email, a co najważniejsze nie posiada licznika rezerwacji przedmiotów. Klucz główny ustawiony jest na kolumnę employee\_id i tak jak w poprzednich tabelach jest on ustawiony na automatyczną inkrementacje.

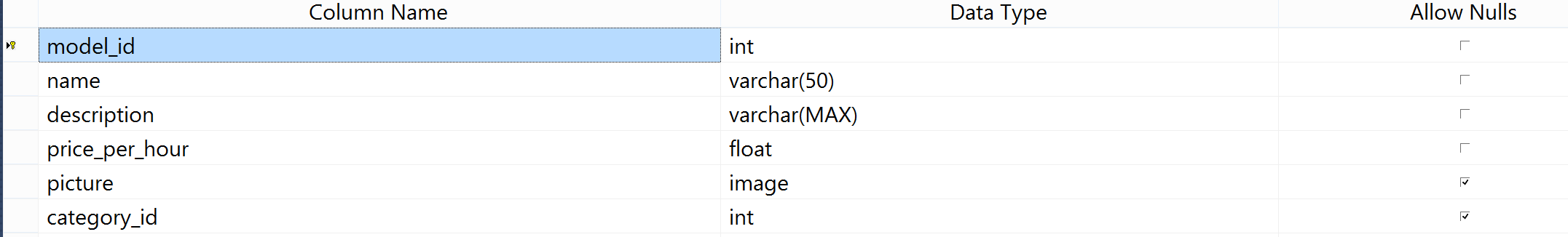
Rysunek 5 Tabela Employee

* Tabela Item

Tabela ta przechowuje klucz główny *item\_id* oraz klucz obcy *model\_id*, dzięki któremu możemy odwoływać się do tabeli Model. Tabela Item posiada również kolumny *itemstatus* oraz *condition*, które są ograniczone o dozwolone słowa, które mogą być wpisane w te kolumny. Dla *itemstatus* są to: R*ent, Reserved, Available* natomiast dla *condition: Good, Normal, Bad*. Poniżej znajduje się zrzut ekranowy tabeli.

Rysunek 6 Tabela Item

* Tabela Model

Tabela ta przechowuje klucz główny *model\_id* jak i klucz obcy *category\_id, dzięki któremu możemy odwoływać się do tabeli Category.* Tabela ta posiada także kolumny *name, description, price\_per\_hour,* które zawierają nazwę modelu, jego opis oraz cenę za godzinę. Kolumna *picture* natomiast przechowuje obraz w bazie danych.

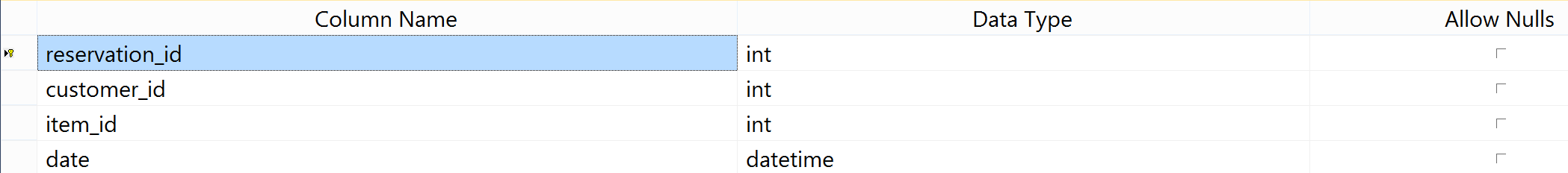
Rysunek 7 Tabela Model

Aby wstawić do tabeli Model zdjęcie, została wykorzystana komenda w SQL, która znajduje się poniżej.

|  |
| --- |
| **INSERT** **INTO** Model **(**name**,** description**,** price\_per\_hour**,** picture**)**  **SELECT** 'Cable Installer'**,**  'Install low voltage lighting or hidden pet fence quickly and easily  with the EZ-Cable Installer. This machine actually installs the wire or  cable and backfills while digging the trench.'**,**  90.0**,** **\***  **FROM** OPENROWSET **(BULK** N'C:\path\Cable-Installer.jpg'**,** SINGLE\_BLOB**)** IMAGE |

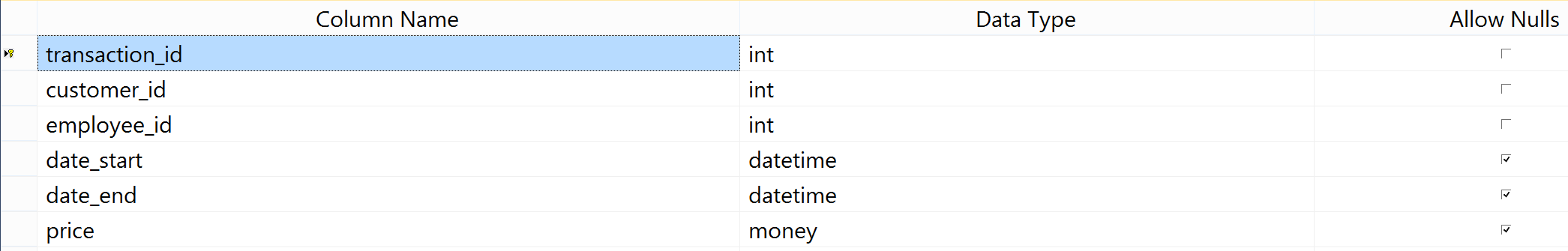
Listing 3 Komendy SQL pozwalające na dodanie zdjęcia oraz innych danych dla Modelu

* Tabela Reservation

Tabela ta odpowiada za przechowywanie informacji o rezerwacji sprzętu, posiada klucz główny *reservaion\_id* oraz dwa klucze obce *customer\_id* i *item\_id* które pozwalają się połączyć odpowiednio z tabelami Customer oraz Item. Tabela Reservation posiada także kolumnę *date*, która przechowuje datę wykonania rejestracji. Data ta pobierana jest za pomocą funkcji *getdate()* ustawioną w programie SQL Server Management Studio.

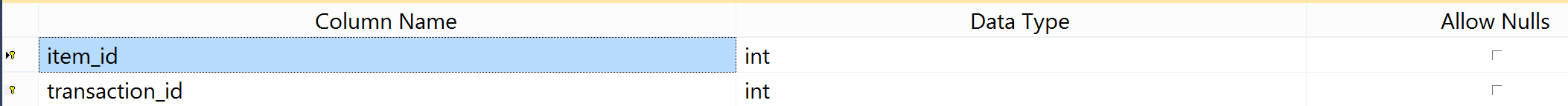
Rysunek 8 Tabela Reservation

* Tabela Transactions

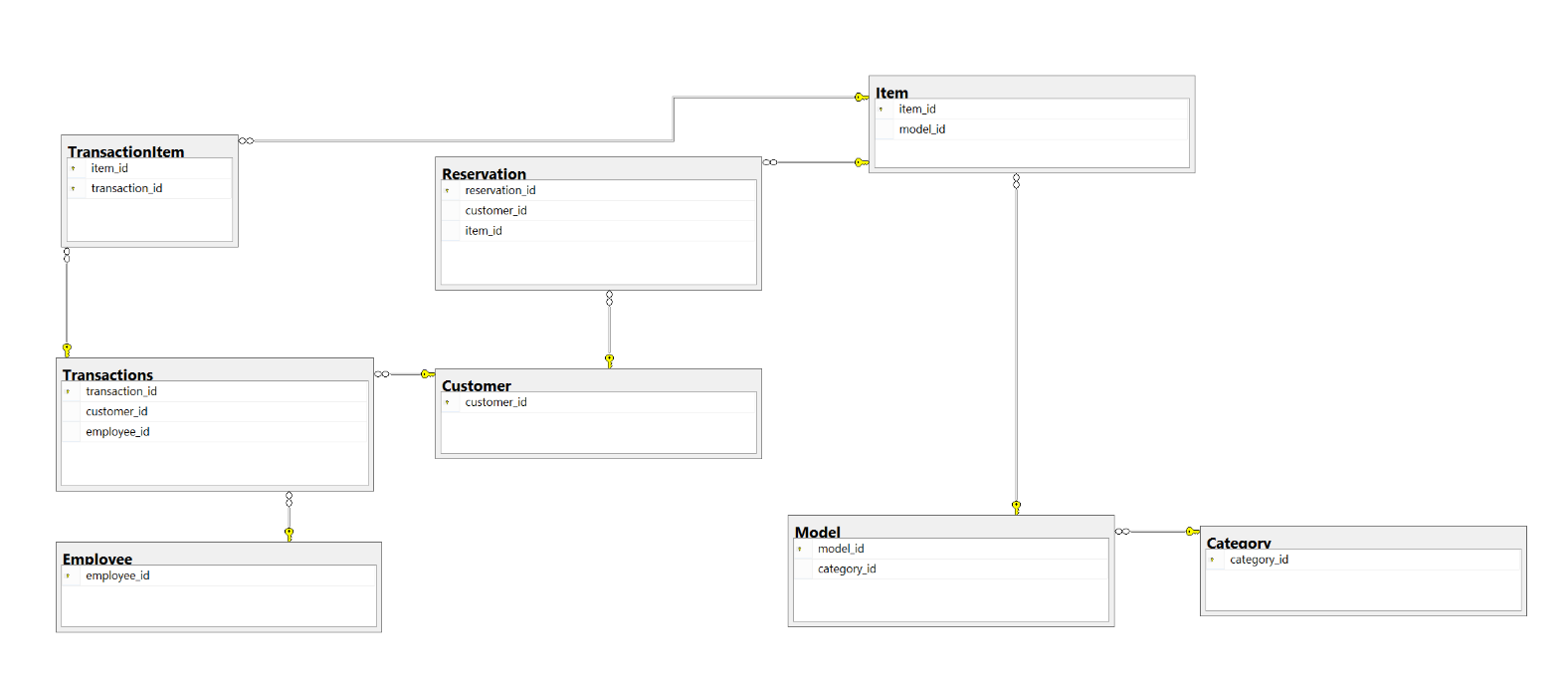
Tabela odpowiadająca transakcje, posiada ona klucz główny *transaction\_id* oraz dwa klucze obce *customer\_id, employee\_id*, które umożliwiają połączenie z tabelami Customer i Employee. Tabela Transactions przechowuje informacje o tym który klient u jakiego wypożyczającego wybrał sprzęt, posiada także kolumnę *date\_start* i *date\_end* informujące nas o rozpoczęciu i zakończeniu wypożyczenia, a także posiada kolumnę *price*, która przechowuje informację o należytej zapłacie za sprzęt.

Rysunek 9 Tabela Transactions

* Tabela TransactionItem

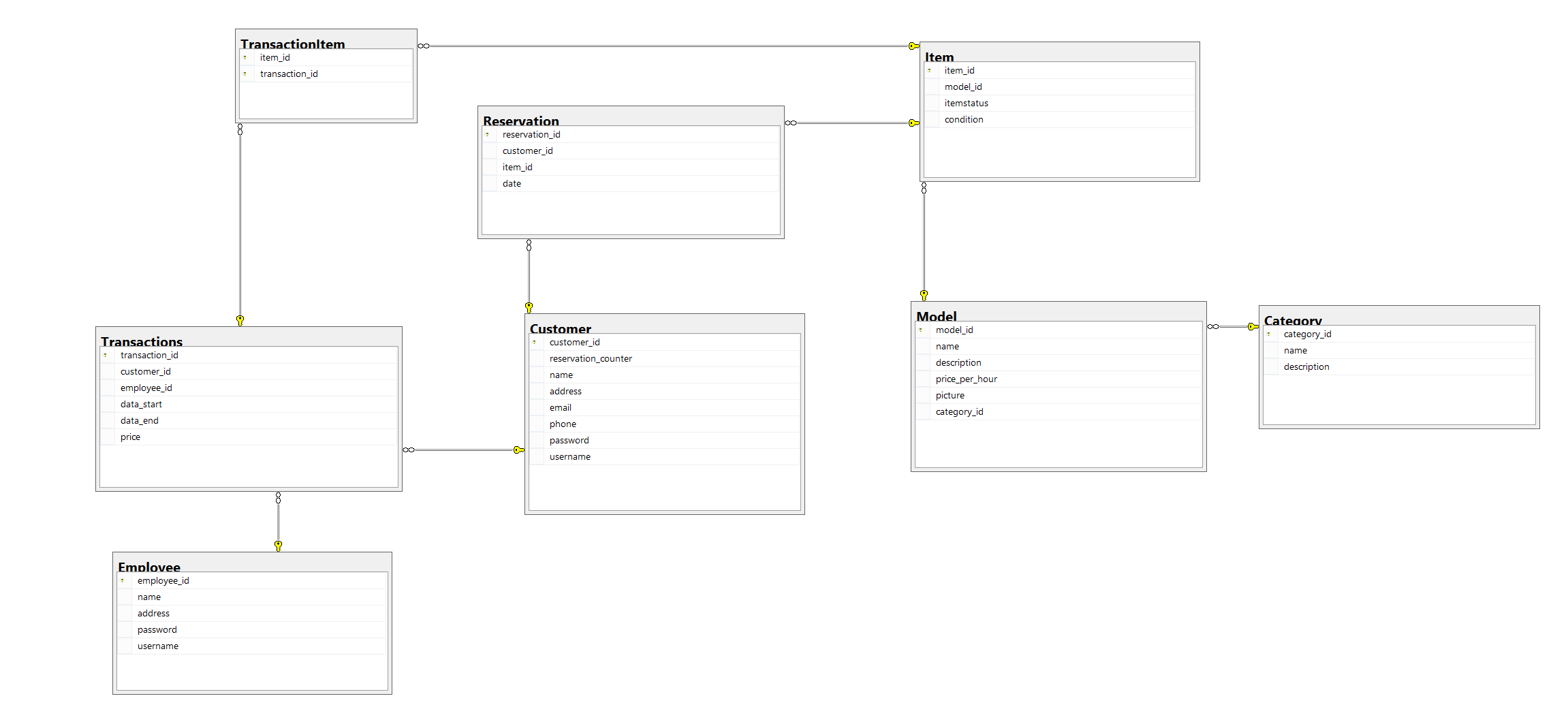
Tabela ta składa się tylko z klucza głównego, który jest złożeniem kluczy obcych, *item\_id* oraz *transaction\_id*. Tabela ta przechowuje informacje o wypożyczonych przedmiotach i numerach ich transakcji jest to najlepszy sposób na przechowywanie takich danych ponieważ pozbywamy się redundancji danych.

Rysunek 10 Tabela TransactionItem

Poniżej znajduje się diagram, pokazujący w graficzny sposób połączenie tabel przy użyciu kluczy obcych.

Rysunek 11 Diagram z kluczami głównymi i obcymi

Na kolejnej stronie znajduje się diagram bazy danych oraz graficzny sposób połączenia jej wykorzystując klucze obce.



Rysunek Diagram bazy danych

W projekcie zostały także zaimplementowane triggery aby usprawnić powtarzalne zmiany w bazie danych.

* Trigger ReservationCounter

Dzięki niemu pole w tabeli Customer zwiększa się o jeden po każdym zarezerwowanym modelu, ma to na celu zwiększenie bezpieczeństwa przed zarezerwowaniem wszystkich dostępnych modeli. Trigger wyzwala się w momencie wywołania komendy Insert w tabeli Reservation. Sprawdza on czy klient nie przekroczył dozwolonej ilości zarezerwowanych przedmiotów, jeśli tak zwraca informację o zaistniałej sytuacji. Poniżej znajduje się kod triggera.

|  |
| --- |
| **GO**  **CREATE** **TRIGGER** ReservationCounter  **ON** Reservation  **after** **Insert**  **AS**  **BEGIN**  **SET** NOCOUNT **ON**  **DECLARE** @vname **varchar(**50**);**  **SET** @vname **=** **(SELECT** name **FROM** Customer **WHERE** Customer**.**customer\_id **in** **(select** customer\_id **from** inserted**))**  **IF** **(SELECT** reservation\_counter **FROM** Customer **WHERE** Customer**.**customer\_id **in** **(select** customer\_id **from** inserted**))** **<** 10  **Update** **[**dbo**].[**Customer**]** **set** reservation\_counter **=** reservation\_counter +1 **where** Customer**.**customer\_id **in** **(select** customer\_id **from** inserted**)**  **ELSE**  **begin**  PRINT @vname **+** ', You cannot order more Item.'  **ROLLBACK** **TRANSACTION;**  **END**  **end** |

Listing 4 Trigger ReservationCounter

* Trigger ReservedStatus

Ten wyzwalacz ma na celu ustawienie wartości w *itemstatus* w tabeli Item na *Reserved* jeśli ta jest ustawiona na wartość *Available*. Trigger uruchamiany jest gdy w tabeli *Reservation* wprowadzana jest nowa dana.

|  |
| --- |
| **GO**  **CREATE** **TRIGGER** ReservedStatus  **ON** Reservation  **after** **Insert**  **AS**  **BEGIN**  **SET** NOCOUNT **ON**  **IF** **(SELECT** itemstatus **FROM** Item **WHERE** Item**.**item\_id **in** **(select** item\_id **from** inserted**))** **=** 'Available'  **Update** **[**dbo**].[**Item**]** **set** itemstatus **=** 'Reserved' **where** Item**.**item\_id **in** **(select** item\_id **from** inserted**)**  **ELSE**  **begin**  **ROLLBACK** **TRANSACTION;**  **END**  **end** |

Listing 5 Trigger ReservedStatus

* Trigger DeleteReservation

Wyzwalacz ten usuwa rezerwacje z tabeli *Reservation* oraz ustala wartość *Rent* w polu *itemstatus* w tabeli Item. Trigger jest wyzwalany w tabeli TransactionItem po wywołaniu komendy Insert.

|  |
| --- |
| **GO**  **CREATE** **TRIGGER** DeleteReservation  **ON** TransactionItem  **after** **Insert**  **AS**  **BEGIN**  **SET** NOCOUNT **ON**  **IF** **(SELECT** itemstatus **FROM** Item **WHERE** Item**.**item\_id **in** **(select** item\_id **from** inserted**))** **=** 'Reserved'  **begin**  **Delete** Reservation **where** Reservation**.**item\_id **in** **(select** item\_id **from** inserted**)**  **Update** Item **SET** itemstatus **=** 'Rent' **where** Item**.**item\_id **in** **(select** item\_id **from** inserted**)**  **end**  **ELSE**  **IF** **(SELECT** itemstatus **FROM** Item **WHERE** Item**.**item\_id **in** **(select** item\_id **from** inserted**))** **=** 'Available'  **begin**  **Delete** Reservation **where** Reservation**.**item\_id **in** **(select** item\_id **from** inserted**)**  **Update** Item **SET** itemstatus **=** 'Rent' **where** Item**.**item\_id **in** **(select** item\_id **from** inserted**)**  **end**  **ELSE**  **IF** **(SELECT** itemstatus **FROM** Item **WHERE** Item**.**item\_id **in** **(select** item\_id **from** inserted**))** **=** 'Rent'  **begin**  **ROLLBACK** **TRANSACTION;**  **END**  **end** |

Listing 6 Trigger DeleteReservation

* Trigger AvaiableStatus

Wyzwalacz ten ustawia wartość *Available* dla *itemstatus* w tabeli Item, po tym gdy pole *data\_end* zostanie ustalone w tabeli Transactions. Trigger ten wywołuje się w momencie wykonania komendy update w tabeli Transactions.

|  |
| --- |
| **GO**  **CREATE** **TRIGGER** AvaiableStatus  **ON** Transactions  **after** **Update**  **AS**  **BEGIN**  **SET** NOCOUNT **ON**  **IF** **((SELECT** data\_end **FROM** Transactions **WHERE** Transactions**.**transaction\_id **in** **(SELECT** transaction\_id **FROM** TransactionItem **WHERE** TransactionItem**.**transaction\_id **in** **(select** transaction\_id **from** Transactions**)))is** **not** **null)**  **begin**  **Update** **[**dbo**].[**Item**]** **set** itemstatus **=** 'Available'  **where** Item**.**item\_id **in** **(SELECT** item\_id **FROM** TransactionItem **WHERE** TransactionItem**.**transaction\_id **in** **(select** transaction\_id **from** inserted**))**  **end**  **ELSE**  **begin**  **ROLLBACK** **TRANSACTION;**  **END**  **end** |

Listing 7 Trigger AvaiableStatus

W każdym z powyższych triggerów wykorzystana została komenda ROLLBACK TRANSACTION, jest ona używana aby po negatywnym wyniku ze sprawdzenia warunku SQL nie wprowadził zmiany do tabel. Gdyby nie było tej komendy, to na przykład, w polu sprawdzającym stan rezerwacji w tabeli Customer, klient mógłby mieć wartość reservation\_count powyżej liczy 10.

1. <http://www.homedepot.com/> [↑](#footnote-ref-1)
2. <http://www.generatedata.com/> [↑](#footnote-ref-2)