|  |  |
| --- | --- |
| Projekt **Technologie obiektowe**  Wydział Elektrotechniki Automatyki i Informatyki  Politechnika Świętokrzyska | |
| St Studia: **Stacjonarne II stopnia** | KiKierunek: **Informatyka** |
| Grupa: **1ID22A** | |
| **Autor:** Karol Skorek | |
| **Dokumentacja techniczna:**  Budowanie narzędzia do budowania diagramów erd i konwersja na język sql | |

Spis treści

[1. Cel projektu 3](#_Toc72684366)

[2. Wstęp teoretyczny 3](#_Toc72684367)

[2.1. Relacja 1 do 1 3](#_Toc72684368)

[2.2. Relacja 1 do \* 3](#_Toc72684369)

[2.3. Relacja \* do \* 4](#_Toc72684370)

[2.4. Dziedziczenie 5](#_Toc72684371)

[2.4.1. SINGLE TABLE 5](#_Toc72684372)

[2.4.2 JOINED 6](#_Toc72684373)

[2.4.3 TABLE\_PER\_CLASS 7](#_Toc72684374)

[2.5. Pozostałe pojęcia 7](#_Toc72684375)

[3. Prezentacja opracowanej przeze mnie aplikacji 8](#_Toc72684376)

[3.1. Opis technologii 8](#_Toc72684377)

[3.2. Prezentacja opracowanej aplikacji 8](#_Toc72684378)

[3.3. Przedstawienie wybranych elementów kodu 18](#_Toc72684379)

[4. Wnioski 24](#_Toc72684380)

# Cel projektu

Celem projektu było opracowanie aplikacji, która pozwala na tworzenie diagramu ERD(Entity Relationship Diagram) oraz konwersję tego diagramu na język SQL. Jako rozszerzenie projektu należy dodać mechanizm dziedziczenia.

# Wstęp teoretyczny

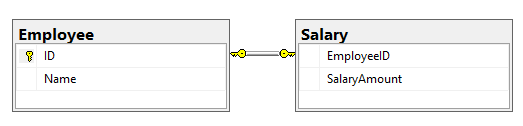
W świecie SQL istnieją trzy podstawowe relacje tj. jeden do jednego (ang. one to one relationship – 1 do 1), jeden do wielu (ang. one to many relationship – 1 do \*) oraz wiele do wielu (ang. many to many - \* do \*). Do każdej z tych relacji opracowana jest inna strategia, która będzie opisana poniżej. Ponadto zastosowano mechanizm dziedziczenia.

## Relacja 1 do 1

Relacja 1 do 1 jest jedną z najprostszych relacji w języku SQL. Założono, że istnieją dwie tabele, które są połączone relacją 1 do 1 o nazwach A i B. W takim wypadku każdy wiersz z tabeli A może mieć tylko jednego odpowiednika tabeli B oraz każdy wiersz tabeli B może mieć tylko jednego odpowiednika tabeli A. Przykładem relacji 1 do 1 może być podzbiór większego zbioru np. Tabela doradcy kredytowi banku są podzbiorem pracowników tego banku. Ostatnią kwestią, jaką należy ustalić to miejsce klucza obcego. W takiej relacji nie ma ściśle określonych reguł, po której stronie klucz obcy musi występować, ale zaleca się, by występował po ważniejszej ze stron. Jest również możliwość utworzenia klucza obcego jako klucza głównego po jednej ze stron tabel.

Przykład graficzny relacji 1 do 1

Przykład użycia relacji 1 do 1 w praktyce zaprezentowano poniżej.

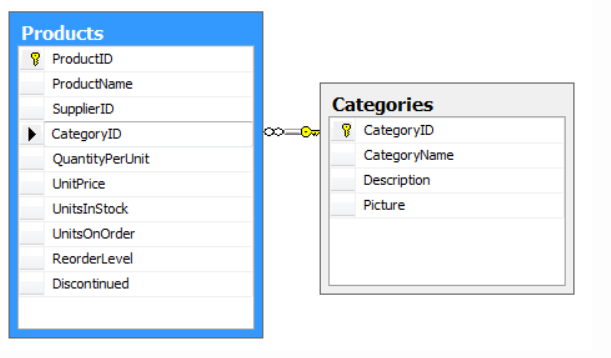


## Relacja 1 do \*

Relacja 1 do \* jest jedną z najlepszych typów relacji. Najprościej można ją zdefiniować jako relację, w której jednemu rekordowi pierwszej tabeli przypisuje się wiele rekordów drugiej tabeli. Przykładem może być relacja Student Grupa, ponieważ 1 student może należeć tylko do jednej grupy, ale w jednej grupie może być wiele studentów. Graficzną reprezentacją relacji 1 do wielu jest.



Przykład użycia relacji 1 do \* w praktyce zaprezentowano poniżej.



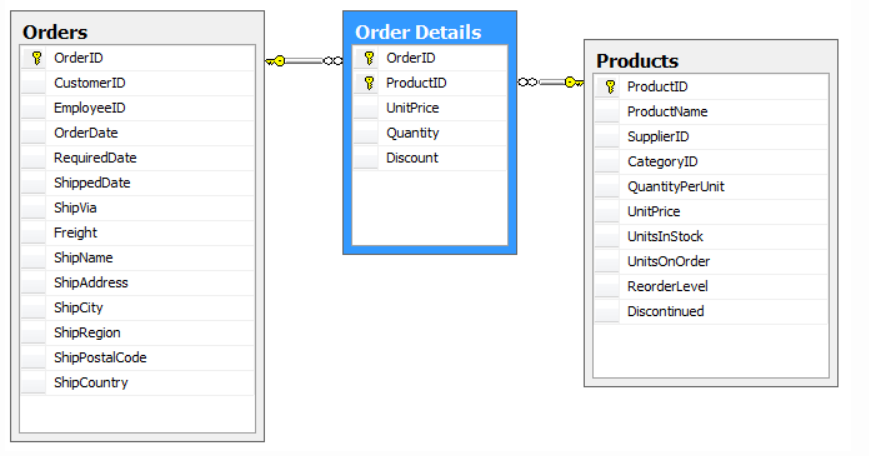
## Relacja \* do \*

Czasem zdarzają się sytuacje, że wiele rekordów pierwszej tabeli odwołuje się do wielu rekordów tabeli drugiej. Przykładem takiej relacji może być związek między produktami a zamówieniami. Jeden produkt może być na wielu zamówieniach oraz jedno zamówienie może dotyczyć wielu produktów. W takim wypadku należy wprowadzić tabele pośrednią.

Są różne reprezentacje graficzne tej relacji. Jednym z takich przykładów zaprezentowano poniżej.

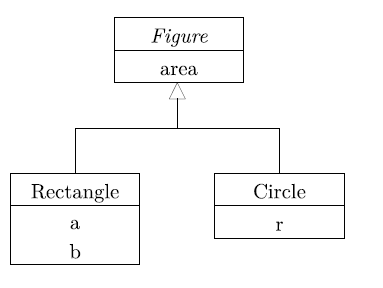


Przykład użycia relacji \* do \* w praktyce zaprezentowano poniżej.



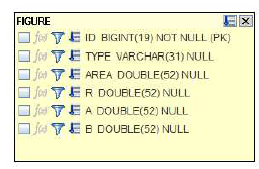
## Dziedziczenie

W tradycyjnym świecie relacji nie ma mechanizmu, który gwarantuje język obiektowy. Mechanizm, o którym mowa jest dziedziczeniem, co oznacza, że klasa dziedzicząca ma dostęp do wybranych pól i metod klasy bazowej. Dziedziczenie pozwala na stworzenie hierarchii klas i jest często wykorzystywany w celu refaktoryzacji kodu. W związku, że język SQL jest uboższy, co oznacza, że nie posiada pewnych elementów takich jak języki obiektowe, należy taki mechanizm zasymulować tak, by chociaż w części przypominał język obiektowy. Opracowano trzy strategie dziedziczenia w przypadku diagramów ERD tj. SINGLE TABLE, JOINED, TABLE\_PER\_CLASS. Każdy przykład będzie rozpatrywany na podstawie poniższego przykładu dziedziczenia.



### 2.4.1. SINGLE TABLE

Single table to strategia, która odpowiada strategii Single Table Inheritance. Metoda to polega na reprezentowaniu hierarchii dziedziczenia klas jako pojedynczą tabelę, zawierającą kolumny, które reprezentują wszystkie pola klasy hierarchii. Zaprezentowany przykład dziedziczenia po przejściu przez metodę single table będzie odpowiadał następującej tabeli.

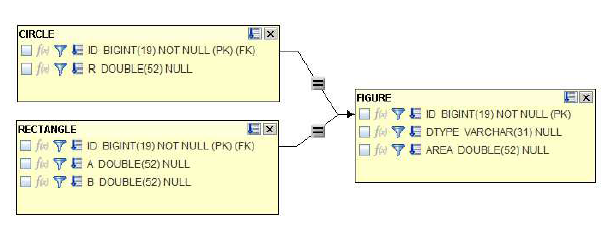


Zaprezentowany przykład ma swoją następującą reprezentację w języku SQL.

|  |
| --- |
| CREATE TABLE FIGURE (  ID BIGINT NOT NULL,  type VARCHAR(31),  AREA FLOAT,  R FLOAT,  A FLOAT,  B FLOAT,  PRIMARY KEY (ID)  ) |

### 2.4.2 JOINED

Joined jest strategią, która ma swój odpowiednik w strategii Class Table Inheritance. Strategia reprezentuje hierarchie dziedziczenia wykorzystując po jednej tabeli dla każdej klasy. Zaprezentowany przykład dziedziczenia po przejściu przez metodę joined będzie odpowiadał następującej reprezentacji tabel.

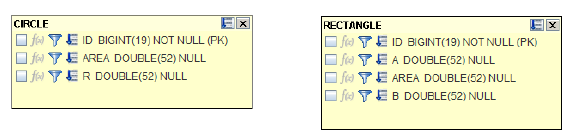


Zaprezentowany przykład ma swoją następującą reprezentację w języku SQL.

|  |
| --- |
| CREATE TABLE FIGURE (ID BIGINT NOT NULL, DTYPE VARCHAR(31), AREA FLOAT, PRIMARY KEY (ID))  CREATE TABLE CIRCLE (ID BIGINT NOT NULL, R FLOAT, PRIMARY KEY (ID))  CREATE TABLE RECTANGLE (ID BIGINT NOT NULL, A FLOAT, B FLOAT, PRIMARY KEY (ID))  ALTER TABLE CIRCLE ADD CONSTRAINT FK\_CIRCLE\_ID FOREIGN KEY (ID) REFERENCES FIGURE (ID)  ALTER TABLE RECTANGLE ADD CONSTRAINT FK\_RECTANGLE\_ID FOREIGN KEY (ID)  REFERENCES FIGURE (ID) |

### 2.4.3 TABLE\_PER\_CLASS

Table per class jest strategią Concrete Table Inheritance. Strategia ta polega na przypisaniu jednej tabeli każdej konkretnej klasie hierarchii dziedziczenia. Każda tabela zawiera kolumny tej klasy i jej przodków. Pola superklasy znajdują się więc w każdej z tabel jej konkretnych podklas. Zaprezentowany przykład dziedziczenia po przejściu przez metodę joined będzie odpowiadał następującej reprezentacji tabel.



Zaprezentowany przykład ma swoją następującą reprezentację w języku SQL.

|  |
| --- |
| CREATE TABLE CIRCLE (ID BIGINT NOT NULL, AREA FLOAT, R FLOAT, PRIMARY KEY (ID))  CREATE TABLE RECTANGLE (ID BIGINT NOT NULL, A FLOAT, AREA FLOAT, B FLOAT, PRIMARY KEY (ID)) |

## Pozostałe pojęcia

Relacja – związek pomiędzy tabelami występujący różnych wariantach. Najpopularniejsze to 1 do 1, 1 do wielu i wiele do wielu. Relacja realizowana jest za pomocą kluczy obcych i głównych.

Klucz główny – zwany również PRIMARY KEY to kolumna bądź zestaw kolumn, które w sposób unikalny definiują wiersz w danej tabeli. Kluczem głównym nie może być wartość NULL.

Klucz obcy – zwany również FOREIGN KEY to kolumna lub grupa kolumn w jednej tabeli, która stanowi klucz główny innej tabeli. Wartość taka musi istnieć w innej tabeli. Możliwe jest, że klucz obcy nie musi odwoływać się do klucza głównego jednakże w takim przypadku musi odwoływać się do kolumny oznaczonej jako UNIQUE.

Encja – jednoznaczny składnik badanej rzeczywistości i w którym przechowywane są dane. Jest odróżnialna od innych elementów, a gdy posiadają te same właściwości tworzone są zbiory encji. Jej reprezentacją graficzną jest prostokąt. Encje są opisywane przez atrybuty.

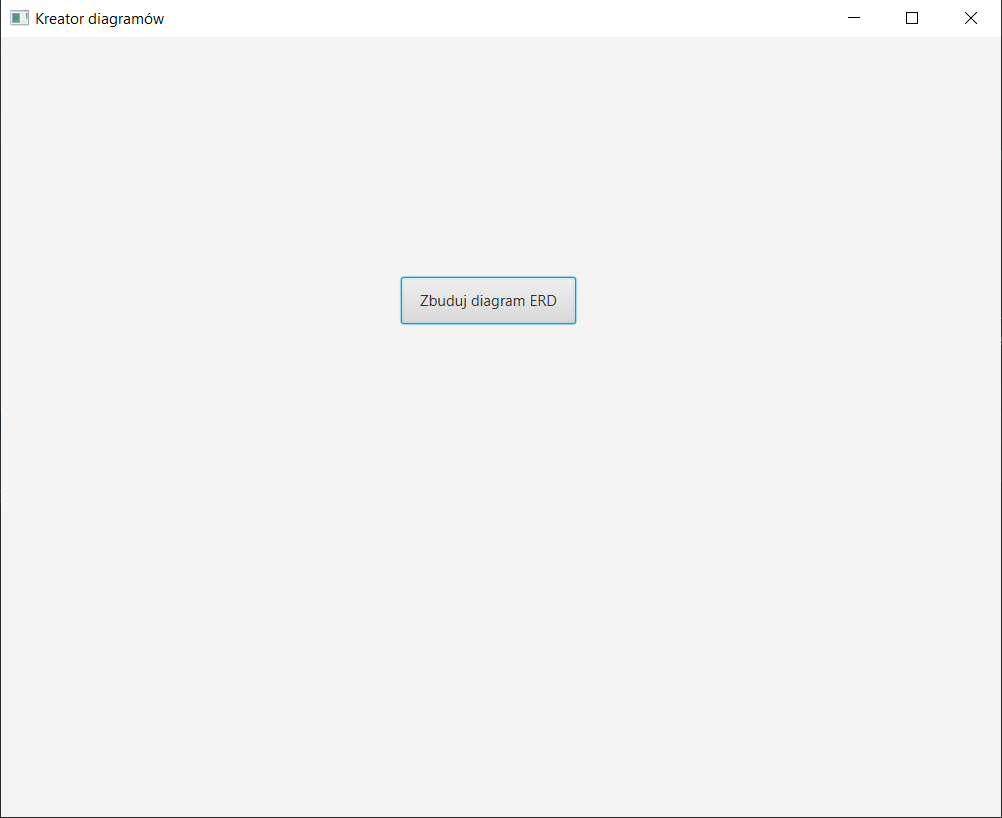
Atrybut – opisują właściwości encji

# Prezentacja opracowanej przeze mnie aplikacji

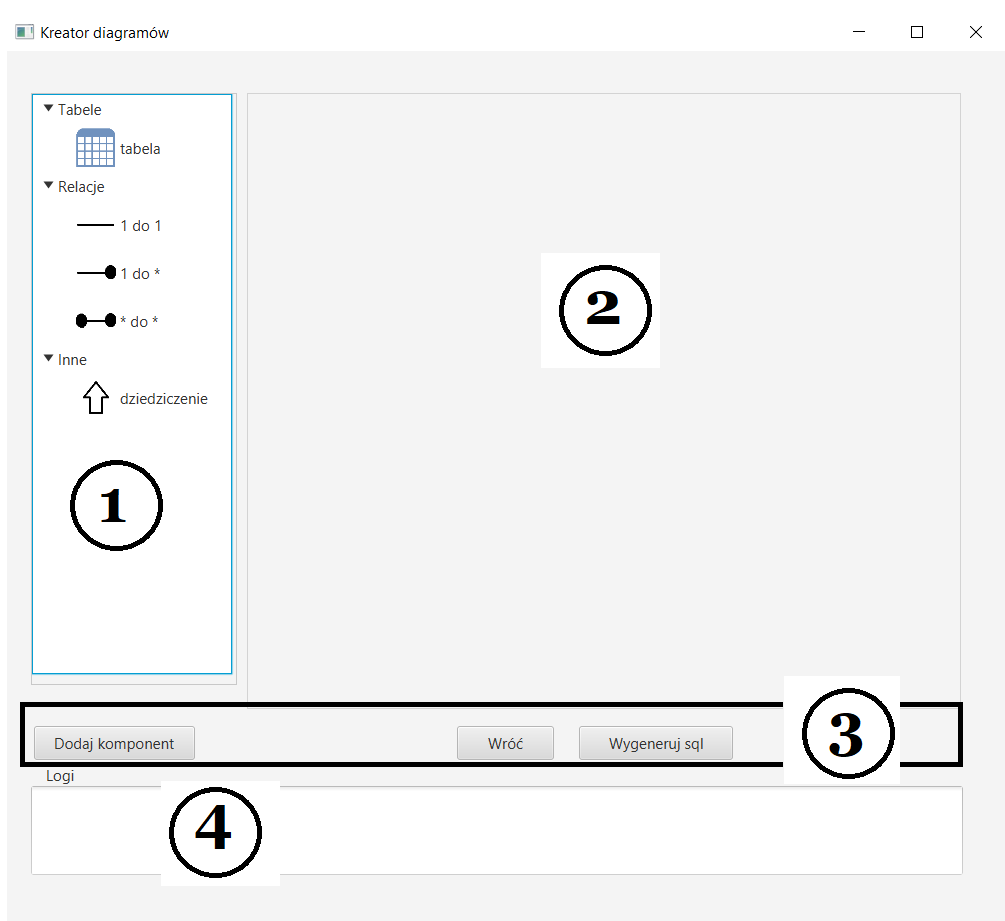
## Opis technologii

Javafx jest technologią do tworzenia aplikacji desktopowych opracowaną przez firmę Sun Microsystems. Wraz z pojawieniem się Javafx w Javie 8 Oracle oświadczyło, że od tej pory staje się rekomendowaną biblioteką do tworzenia interfejsu graficznego. JavaFX była dostępna wcześniej niż w wersji Java 8, jednakże dopiero od tej wersji weszła do zestawu bibliotek podstawowych. Za pomocą JavyFX można tworzyć aplikacje, które wyglądają nie tylko nowocześnie, ale i są wydajne i pozwala na pisanie czytelnego kodu. JavaFX umożliwia definiowanie widoku za pomocą języka XML, co niewątpliwie ułatwia odseparowanie widoku od logiki biznesowej aplikacji. Nie oznacza to jednak, że nie jest możliwe wpływanie na widok z poziomu kodu Javy, jednakże tam gdzie można tego uniknąć należy oddzielać te operacje. Graficzną reprezentację gwarantuje narzędzie Scene Builder, który dostarcza mnóstwo podstawowych kontrolek. Można modyfikować jego widok w kodzie fxml albo dołączając plik CSS. Z wyżej opisanych powodów zdecydowałem się na tą technologię

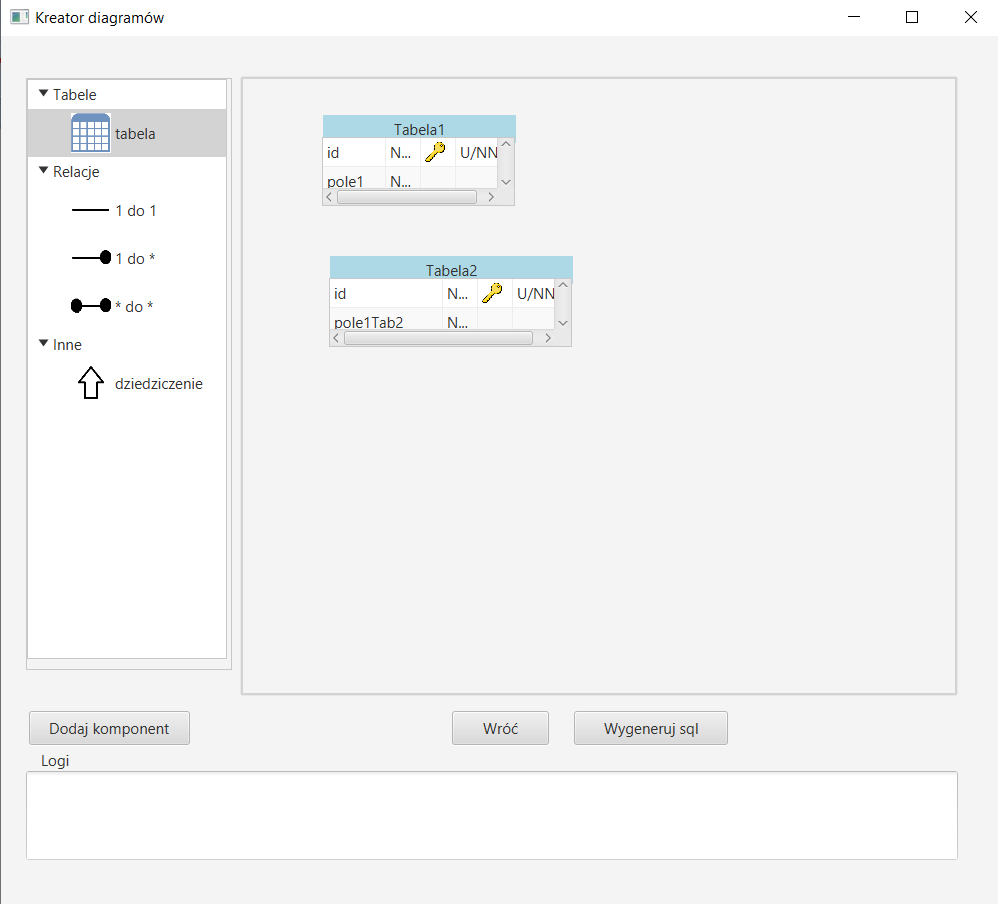
## Prezentacja opracowanej aplikacji



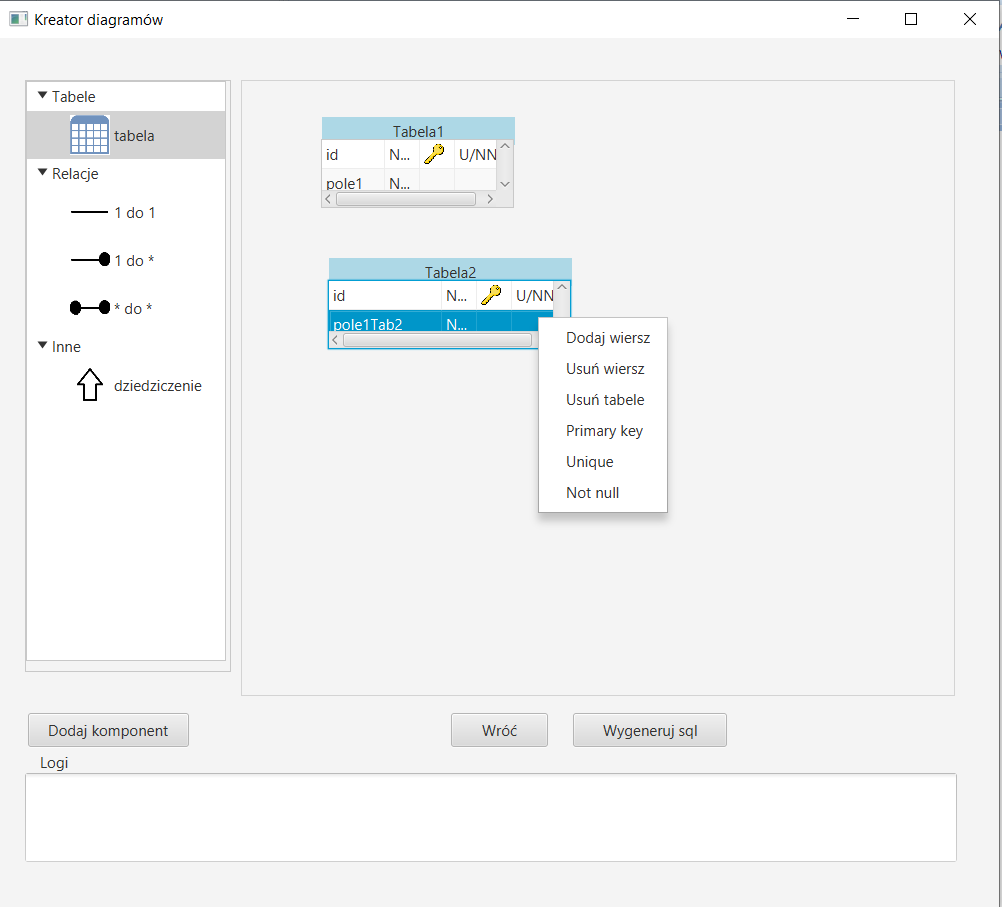
Zaprezentowany wyżej obrazek pokazuje okno główne programu. Jest do wyboru tylko jedna opcja *Zbuduj diagram ERD*.



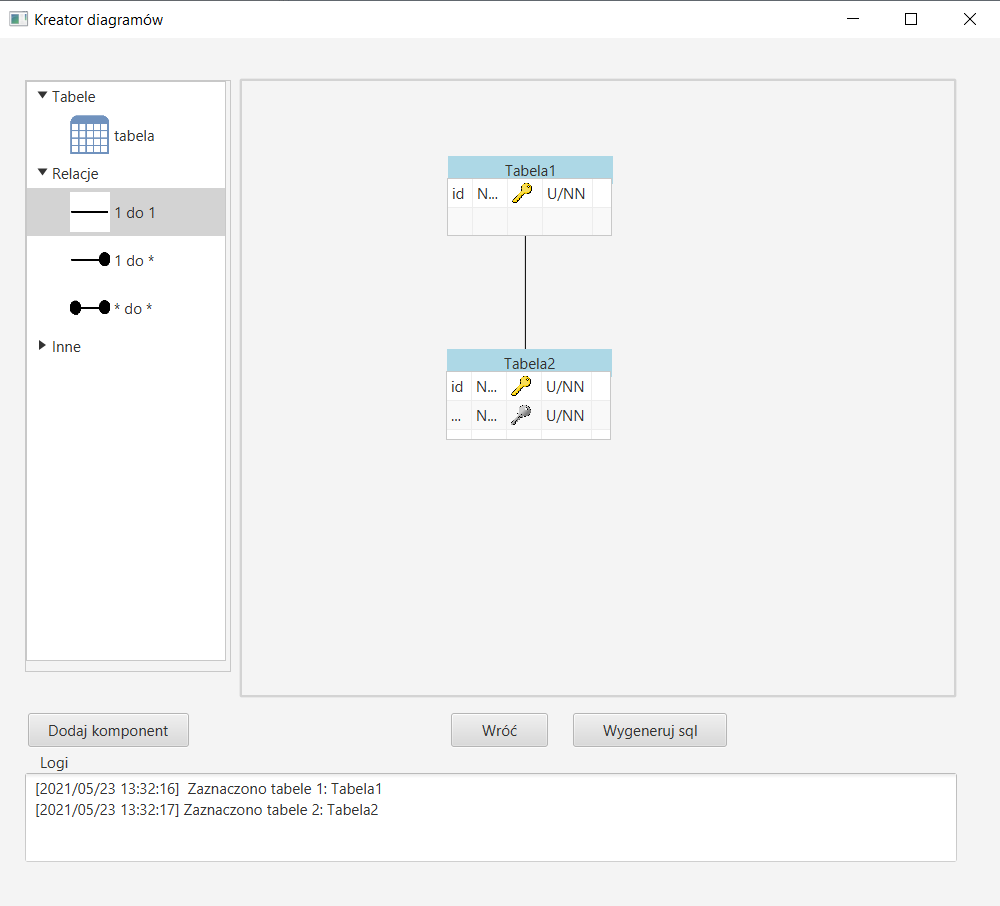
Po kliknięciu przycisku *Zbuduj diagram ERD* można wyróżnić cztery części. Część oznaczona cyfrą 1 to pole wyboru opcji tj. dodawanie tabeli, czy też relacji między tabelami. Pole robocze oznaczone cyfrą 2 jest elementem, w którym będą widoczne tabele. To właśnie tam można modyfikować nazwy tabeli, dodawać rekordy czy też łączyć tabele poprzez wybór odpowiedniej relacji. Pole przycisków oznaczone obszarem numer 3. Można wybrać opcję *Dodaj komponent*, który jest zaznaczony z obszaru numer 1. Przycisk *Wróć* pozwala na powrót do poprzedniego okna. Natomiast przycisk Wygeneruj sql spowoduje wygenerowanie kodu na podstawie tabel i relacji między nimi. Ostatnim omawianym obszarem to obszar logów oznaczony numerem 4. Tu pojawiają się wszystkie komunikaty związane z akcjami użytkownika.



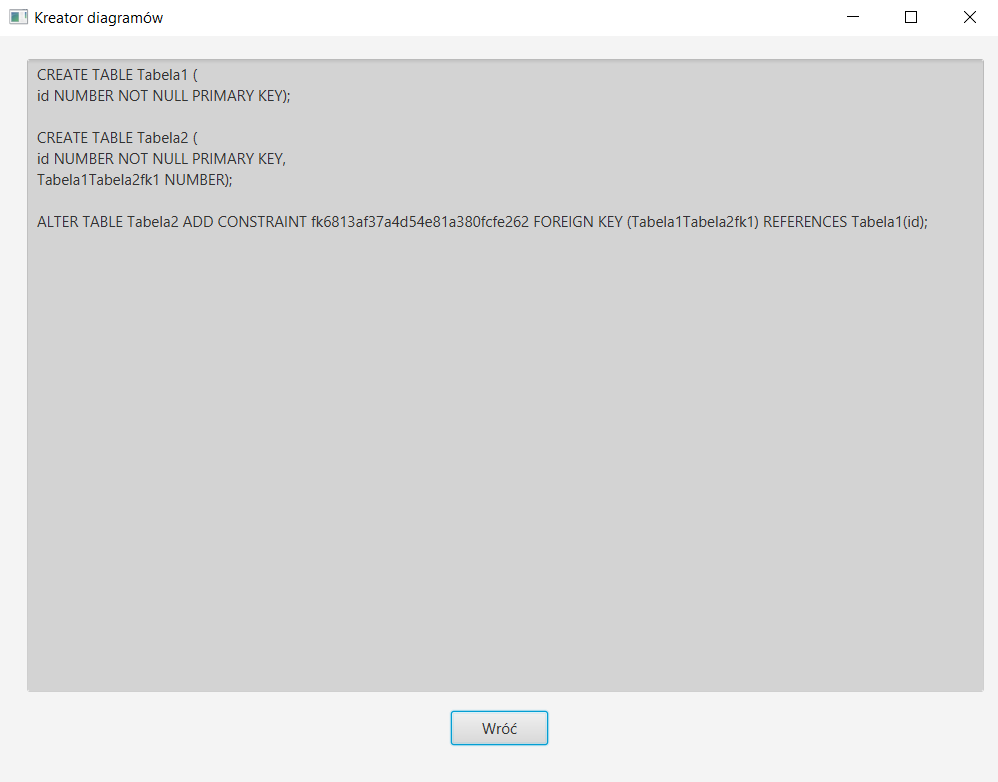
Przy zmianie długości tekstu tabela automatycznie się rozciąga. Warto zwrócić uwagę, że id ma złoty klucz, co oznacza PRIMARY KEY. Stąd obok jest informacja U/NN co oznacza UNIQUE/NOT NULL.



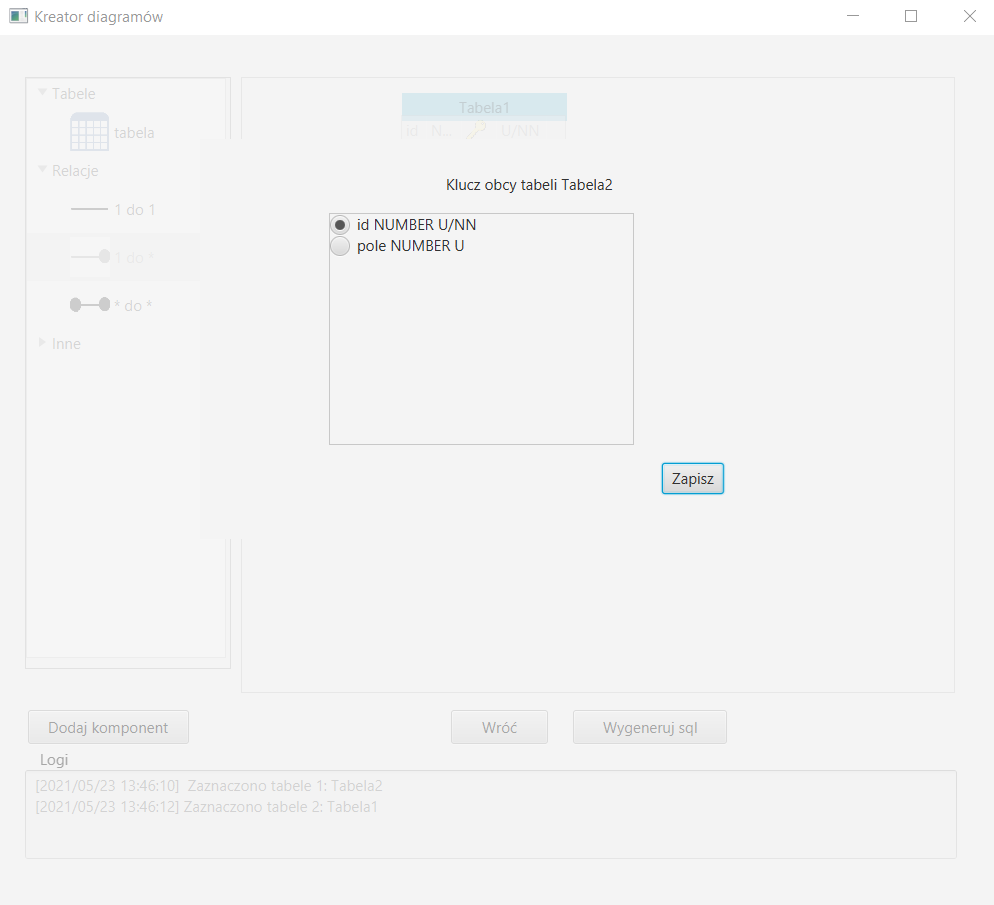
Po wybraniu danego pola i przyciśnięciu prawego przycisku myszy rozwija się pole z opcjami. Pierwsza opcja dodaje wiersz. Po kliknięciu w nazwę danego pola można modyfikować jego wartości. Jest możliwość zmiany klucza głównego. Wtedy program znajduje klucz obcy i zmienia go na zaznaczony wiersz. Można usunąć tabele lub pojedynczy wiersz. Można zdefiniować czy pole ma być unikalne oraz czy ma być inne niż null.



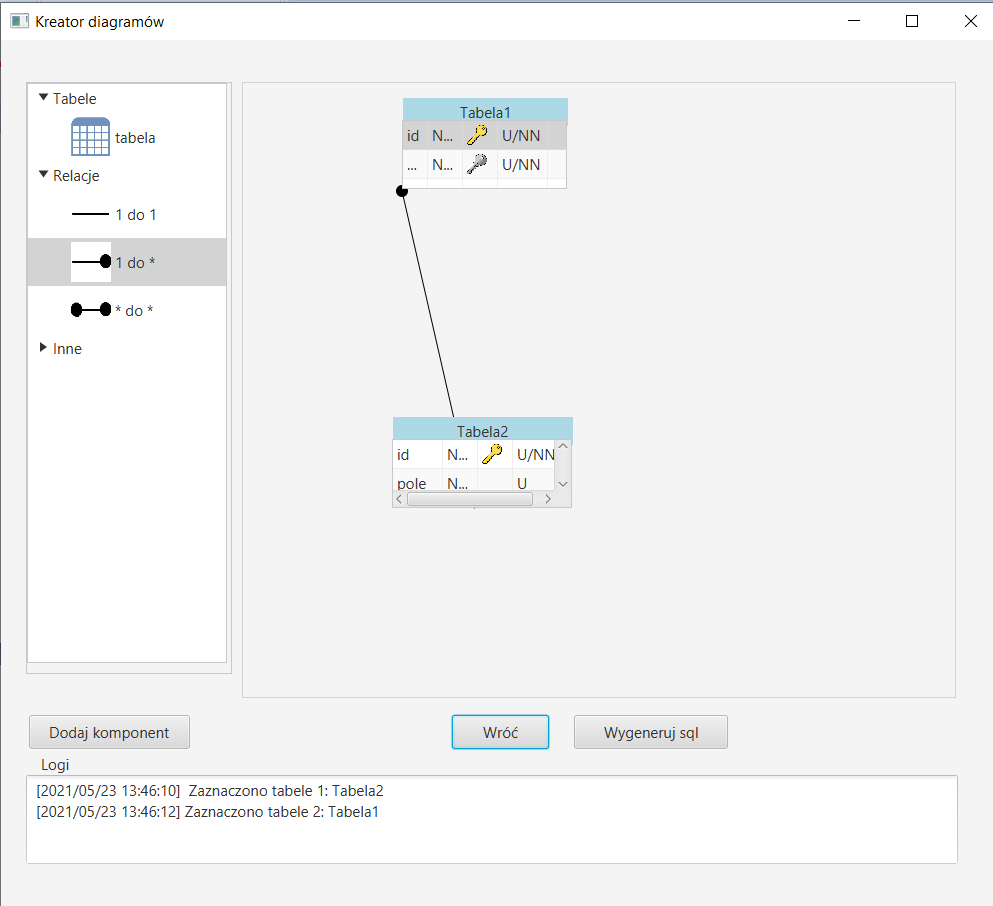
Po połączeniu między tabelami pojawia się relacja. W tym wypadku wybrano relację 1 do 1. Zawsze druga kliknięta tabela będzie zawierała klucz obcy do tej pierwszej. Relację można usunąć klikając na linię prawym przyciskiem myszy. Pojawi się okno do usunięcia relacji. Po kliknięciu przycisku *Wygeneruj sql* wygeneruje się następujący kod.



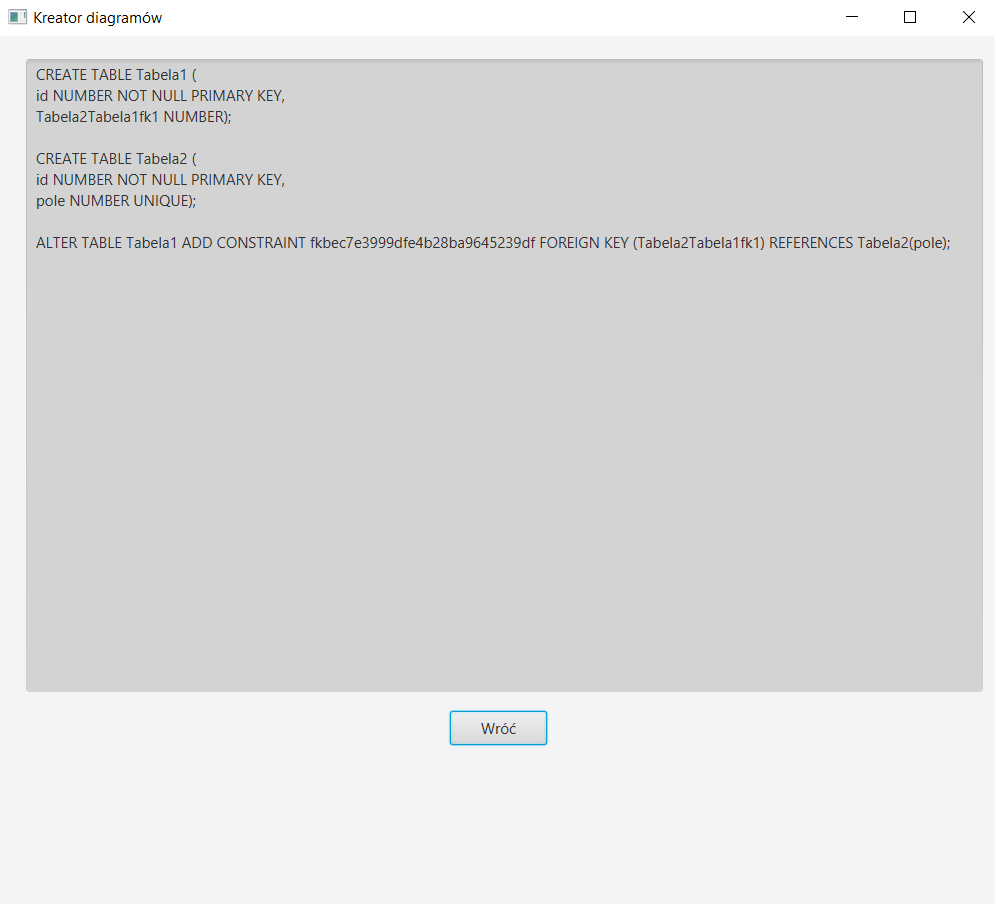
W moim rozwiązaniu zawsze druga wybrana tabela odwołuje się do klucza obcego pierwszej tabeli. Teraz zostanie omówiona kolejna relacja tj. 1 do \*



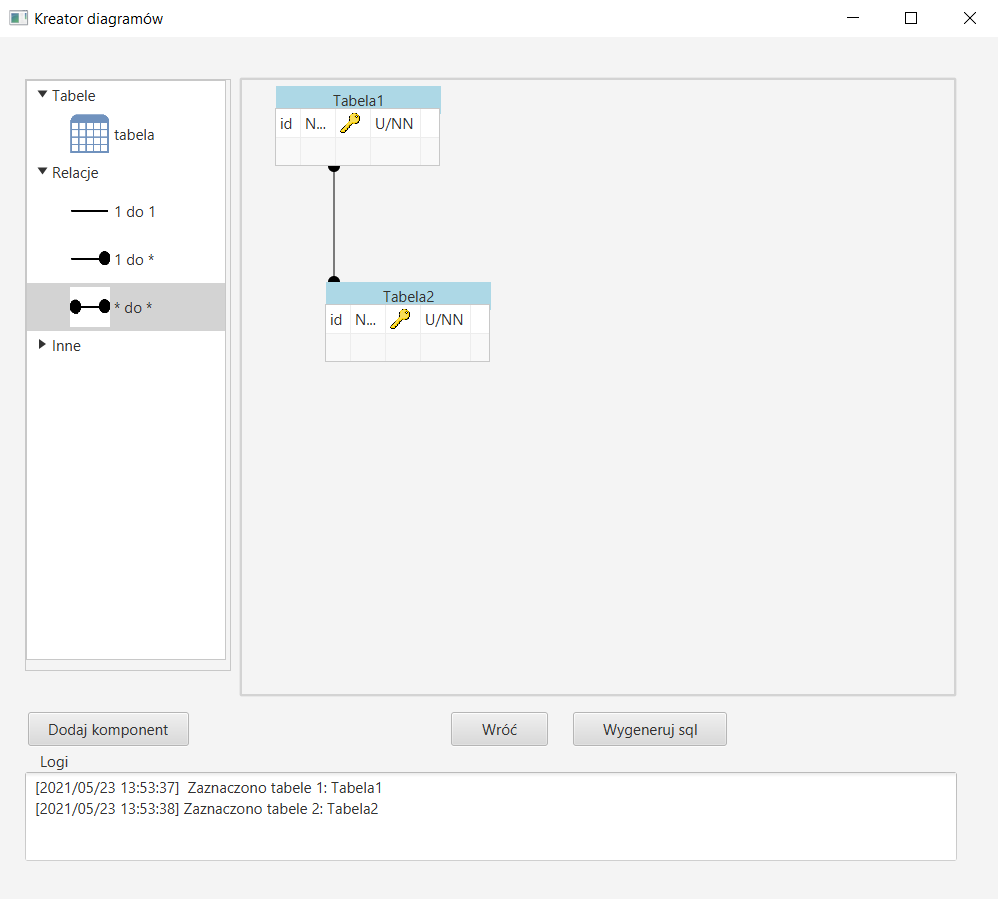
W przypadku relacji 1 do wielu możliwy jest wybór innego pola niż klucz główny jako odwołanie. Warunek jest taki, że takie pole musi mieć modyfikator UNIQUE. W zaprezentowanym przykładzie wybrałem opcje pole i kliknąłem zapisz. Po dokonaniu takiej operacji wyświetlił się następujący wynik.

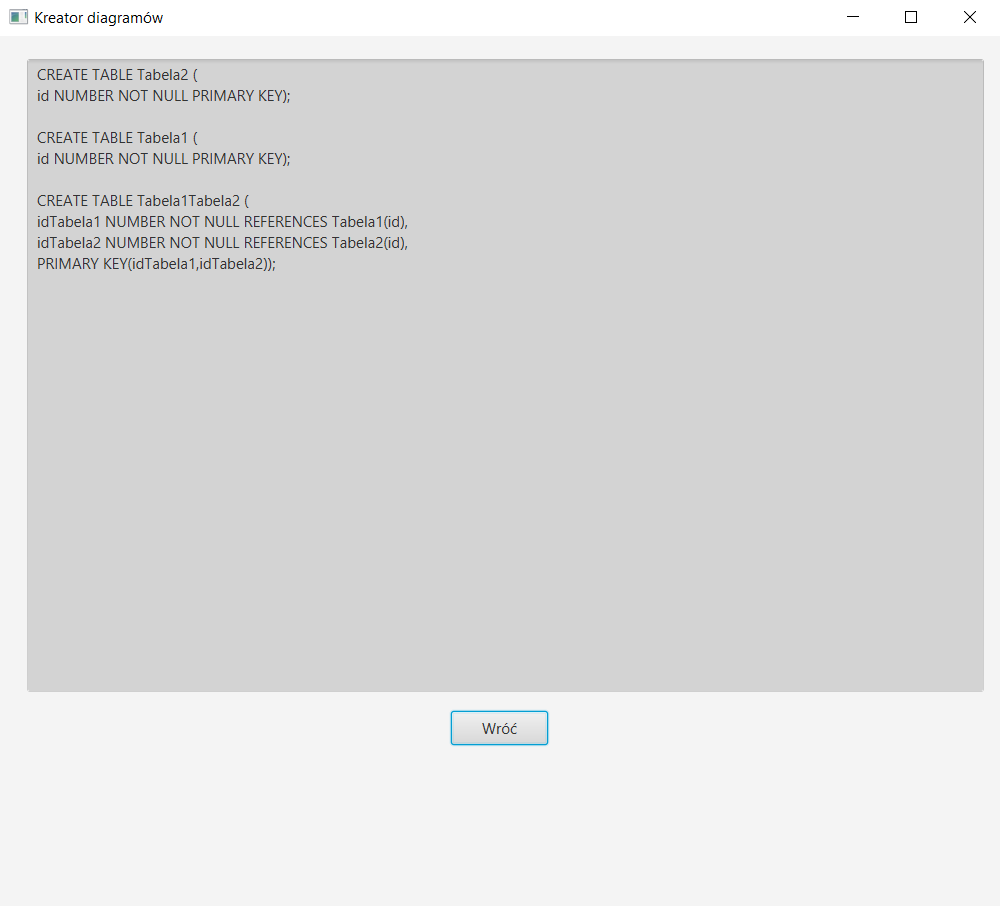


Proszę zwrócić uwagę na końcówkę połączenie. Tam gdzie strona jest wiele linia zakończona jest kołem które reprezentuje ten rodzaj relacji. Wynikiem takiej operacji jest następujący kod sql.

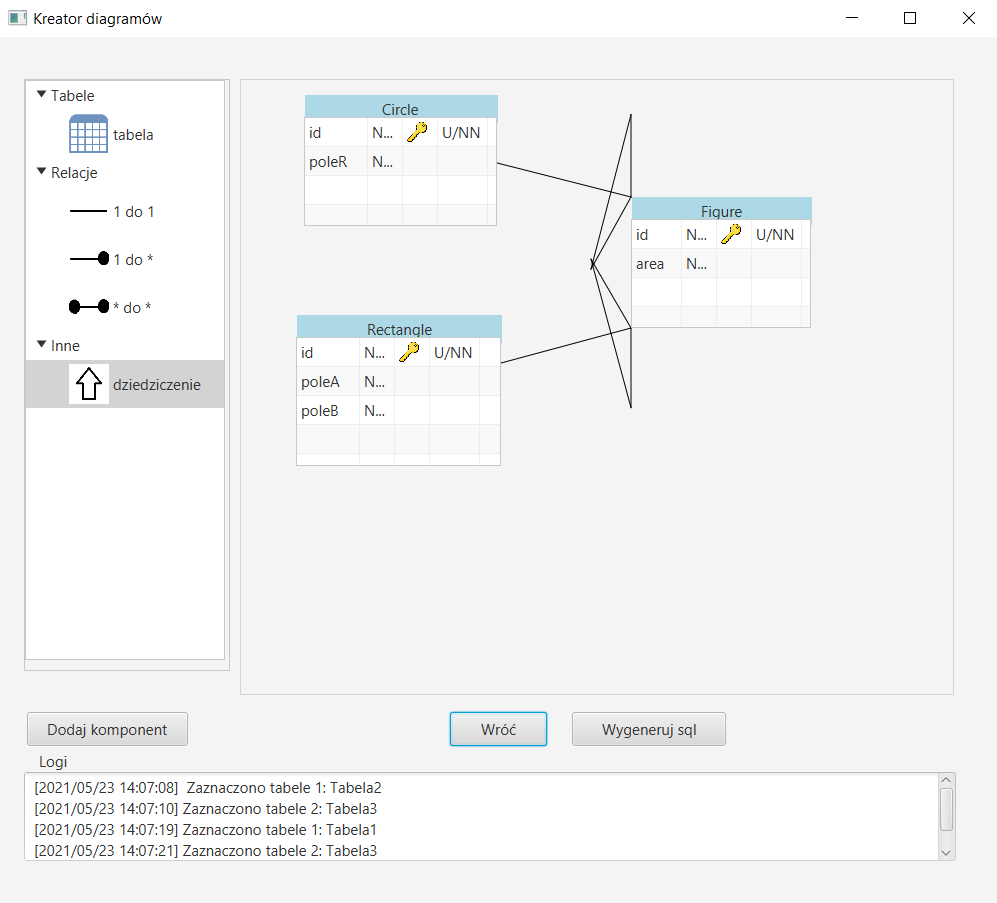


Trzecim rodzajem relacji jest wiele do wielu.

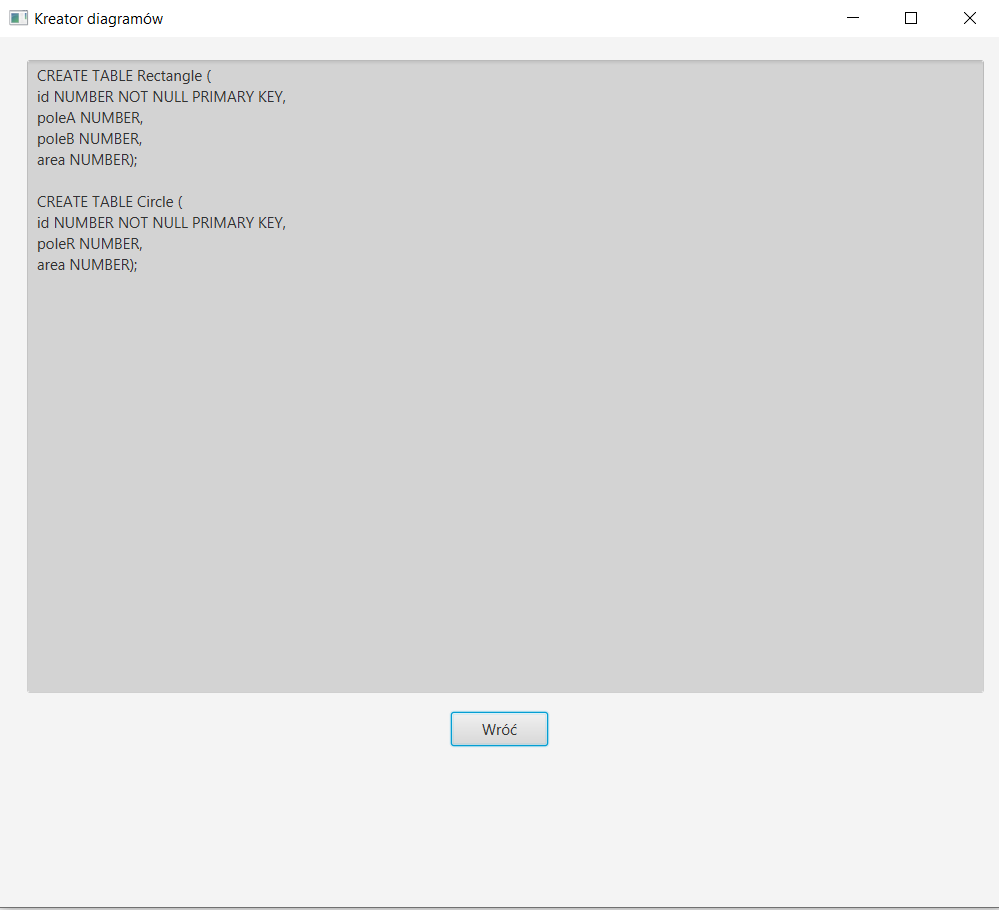


Ten problem rozwiązałem w ten sposób, że graficznie nie powstaje nowa tabela pośrednia oraz nie jest widoczny, żaden nowy klucz obcy. Natomiast po kliknięciu przycisku *Wygeneruj sql* pojawia się następujący kod.

Ostatnim typem operacji, jaki chciałbym pokazać jest operacja dziedziczenia. Zdecydowałem się na strategię TABLE\_PER\_CLASS.



Zaprezentowany przykład jest odpowiednikiem przykładu omawianego w części teoretycznej. Wygenerowany kod SQL jest następujący.



## Przedstawienie wybranych elementów kodu

|  |
| --- |
| public class TableModel {   private String id;  private String type;  private ImageView primaryForeignNoneKey;  private String additional;  private boolean primaryKey, foreignKey, isUnique, isNotNull;   public TableModel(String id, String type,ImageView primaryForeignNoneKey,String additional) {  this.id = id;  this.type = type;  this.primaryForeignNoneKey = primaryForeignNoneKey;  this.additional = additional;  }   public ImageView getPrimaryForeignNoneKey() {   return primaryForeignNoneKey;  }   public boolean isPrimaryKey() {  return primaryKey;  }   public void setPrimaryKey(boolean primaryKey) {  this.primaryKey = primaryKey;  }   public boolean isForeignKey() {  return foreignKey;  }   public void setForeignKey(boolean foreignKey) {  this.foreignKey = foreignKey;  }   public boolean isUnique() {  return isUnique;  }   public void setUnique(boolean unique) {  isUnique = unique;  }   public boolean isNotNull() {  return isNotNull;  }   public void setNotNull(boolean notNull) {  isNotNull = notNull;  }   public void setPrimaryForeignNoneKey(ImageView primaryForeignNoneKey) {  this.primaryForeignNoneKey = primaryForeignNoneKey;  }   public String getId() {  return id;  }   public void setId(String id) {  this.id = id;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public String getAdditional() {  return additional;  }   public void setAdditional(String additional) {  this.additional = additional;  }   public ObservableList<?> getColumns(){   TableColumn keyColumn = new TableColumn("id");  keyColumn.setCellValueFactory(new PropertyValueFactory("id"));  keyColumn.setCellFactory(TextFieldTableCell.*forTableColumn*());  keyColumn.setCellFactory(TextFieldTableCell.*forTableColumn*());  keyColumn.setMinWidth(25);  keyColumn.setMaxWidth(25);   TableColumn typeColumn = new TableColumn("type");  typeColumn.setCellValueFactory(new PropertyValueFactory("type"));  typeColumn.setMinWidth(35);  typeColumn.setMaxWidth(35);  typeColumn.setCellFactory(TextFieldTableCell.*forTableColumn*());   TableColumn primaryForeignNoneKeyColumn = new TableColumn("primaryForeignNoneKey");  primaryForeignNoneKeyColumn.setCellValueFactory(new PropertyValueFactory("primaryForeignNoneKey"));  primaryForeignNoneKeyColumn.setMinWidth(35);  primaryForeignNoneKeyColumn.setMaxWidth(35);   TableColumn additionalColumn = new TableColumn("additional");  additionalColumn.setCellValueFactory(new PropertyValueFactory("additional"));  additionalColumn.setMinWidth(50);  additionalColumn.setMaxWidth(50);   List list = new ArrayList();  list.add(keyColumn);  list.add(typeColumn);  list.add(primaryForeignNoneKeyColumn);  list.add(additionalColumn);  return FXCollections.*observableList*(list);  }     public void assignKey(XTableView xTableView){  List list = new ArrayList(xTableView.getItems());  list.add(this);  ObservableList data = FXCollections.*observableList*(list);  xTableView.setItems(data);  }   public void updateDataKey(XTableView xTableView, boolean isUpdatedPK){  List list = new ArrayList(xTableView.getItems());  if(isUpdatedPK) {  xTableView.getItems().forEach(pk -> {  if (!pk.equals(this) && ((TableModel) pk).isPrimaryKey()) {  ((TableModel) pk).setPrimaryForeignNoneKey(null);  ((TableModel) pk).setPrimaryKey(false);  int index = list.indexOf(pk);  list.remove(index);  TableModel t = new TableModel(((TableModel) pk).getId(), ((TableModel) pk).getType(),  ((TableModel) pk).getPrimaryForeignNoneKey(),((TableModel) pk).additional);  t.setAdditional("");  list.add(index, t);   }  });  }  int index = list.indexOf(this);  list.remove(this);  TableModel t = new TableModel(getId(),getType(),getPrimaryForeignNoneKey(),getAdditional());  t.setPrimaryKey(isPrimaryKey());  t.setForeignKey(isForeignKey());  t.setUnique(isUnique());  t.setNotNull(isNotNull());  t.setAdditional("U/NN");  list.add(index,t);  ObservableList data = FXCollections.*observableList*(list);  xTableView.setItems(data);  }   public TableModel updateData(XTableView xTableView){  List list = new ArrayList(xTableView.getItems());  int index = list.indexOf(this);  list.remove(this);  TableModel t = new TableModel(getId(),getType(),getPrimaryForeignNoneKey(),getAdditional());  t.setPrimaryKey(isPrimaryKey());  t.setForeignKey(isForeignKey());  t.setUnique(isUnique());  t.setNotNull(isNotNull());  list.add(index,t);  ObservableList data = FXCollections.*observableList*(list);  xTableView.setItems(data);  return t;  } } |

Zaprezentowana klasa to klasa modelu tabeli. Tu znajdują się operacje mające na celu aktualizację tabeli. Dodawanie wierszy. Wszystkie informacje dotyczące pól tabeli łącznie z kluczami. Jak i również przypisywanie różnych atrybutów.

|  |
| --- |
| public Point2D calculateTheShortestPoint(MoveableNodeModel moveableNodeModel, LineConnection lineConnection) {  Point2D startPosition = new Point2D(lineConnection.getStartX(), lineConnection.getStartY());  Map<Point2D, Double> distances = new HashMap<>();  int size = 10;  for (int i = 0; i < size; i++) {  Point2D left = new Point2D(moveableNodeModel.getAnchorPane().getBoundsInParent().getMinX(),  lineConnection.getEndY() + (moveableNodeModel.getAnchorPane().getHeight() \* i) / size);  distances.put(left, startPosition.distance(left));  Point2D top = new Point2D(lineConnection.getEndX() + (moveableNodeModel.getAnchorPane().getWidth() \* i) / size,  moveableNodeModel.getAnchorPane().getBoundsInParent().getMinY());  distances.put(top, startPosition.distance(top));  Point2D right = new Point2D(moveableNodeModel.getAnchorPane().getBoundsInParent().getMaxX(),  lineConnection.getEndY() + (moveableNodeModel.getAnchorPane().getHeight() \* i) / size);  distances.put(right, startPosition.distance(right));  Point2D bottom = new Point2D(lineConnection.getEndX() +  (moveableNodeModel.getAnchorPane().getWidth() \* i) / size, moveableNodeModel.getAnchorPane().getBoundsInParent().getMaxY());  distances.put(bottom, startPosition.distance(bottom));  }   return distances.entrySet().stream().min(Map.Entry.*comparingByValue*()).get().getKey(); } |

Jedną z ciekawszych metod jest wyszukiwanie najkrótszej drogi między tabelami. W tym celu skorzystałem z metody distance klasy Point2D. następnie dodaje do kolekcji wartości i zwracam, która wartość jest najmniejsza.

Jednym z najtrudniejszych zagadnień było opracowanie znaku dziedziczenia. Konkretnie należało określić punkt w jakiejś odległości od końca linii, następnie znaleźć równanie prostej prostopadłej do tej linii. Wyznaczyć pewne odległości równe i połączyć wszystkie punkty

|  |
| --- |
| Pair<Point2D,Point2D> pointAndEquation = calculateABOfPerpendicularLine(); Point2D pointFirst = calculateNewPointLine(pointAndEquation.getKey(),pointAndEquation.getValue(),20); Point2D pointSecond = calculateNewPointLine(pointAndEquation.getKey(),pointAndEquation.getValue(),-20); Point2D top = new Point2D(line.getEndX(),line.getEndY()); setPositionOfLineForTriangle(firstLine,pointFirst,pointSecond); setPositionOfLineForTriangle(secondLine,pointFirst,top); setPositionOfLineForTriangle(thirdLine,top,pointSecond);  ...  private void setPositionOfLineForTriangle(Line line,Point2D start,Point2D end){  line.setStartX(start.getX());  line.setStartY(start.getY());  line.setEndX(end.getX());  line.setEndY(end.getY()); }  private Pair<Point2D,Point2D> calculateABOfPerpendicularLine(){  Point2D start = new Point2D(line.getStartX(),line.getStartY());  Point2D end = new Point2D(line.getEndX(),line.getEndY());  double mainW = start.getX()\*1-1\*end.getX();  double aW = start.getY()\*1-1\*end.getY();  double bW = start.getX()\*end.getY()-start.getY()\*end.getX();  Point2D equation = new Point2D(aW/mainW,bW/mainW);  Point2D basePoint = calculateNewPointLine(equation,end,-20);  double a2 = -1/equation.getX();  double b2 = basePoint.getY()-basePoint.getX()\*a2;  return new Pair<Point2D,Point2D> (new Point2D(a2,b2),basePoint); } private Point2D calculateNewPointLine(Point2D equation,Point2D end,double offset){  double x =end.getX()+offset;  return new Point2D(x,equation.getX()\*x+equation.getY()); } |

Jak widać do obliczenia odpowiednich współrzędnych użyto metodę wyznacznikową.

Poniżej pokazuję metodę, która odpowiedzialna jest za tworzenie kodu SQL

|  |
| --- |
| public void buildSQL() {  StringBuilder queryBuilder = new StringBuilder();  lineConnections.forEach(lineConnection -> {  Label tableSecondName = findLabelAfterAnchorPane(lineConnection.getTableSecond());  Label tableFirstName = findLabelAfterAnchorPane(lineConnection.getTableFirst());  String textAreaString = textArea.getText();   if (lineConnection.getConnectionType().equals("dziedziczenie")) {  if (textArea.getText().contains("CREATE TABLE " + tableSecondName.getText())) {  String[] splittedTextArea = textAreaString.split("CREATE TABLE " + tableSecondName.getText() + " \\(");  String[] splittedPartTwo = splittedTextArea[1].split("\\);");  StringBuilder newStringBuilder = new StringBuilder(",\n");  Pair<List<TableModel>,StringBuilder> pair = getAllTableModelFromTableWithoutKeys(lineConnection,  newStringBuilder);  List<TableModel> tableModels = pair.getKey();  newStringBuilder = pair.getValue();  if (newStringBuilder.toString().trim().length() > 1) {  if (textArea.getText().contains("CREATE TABLE " + tableFirstName.getText())) {  splittedTextArea = textAreaString.split("CREATE TABLE " + tableFirstName.getText() + " \\(");  splittedPartTwo = splittedTextArea[1].split("\\);");  StringBuilder stringBuilderSecond = new StringBuilder(splittedPartTwo[0]);  stringBuilderSecond.append(newStringBuilder.toString());  textAreaString = textAreaString.replace(splittedPartTwo[0], stringBuilderSecond.toString());  textArea.setText(textAreaString);  }  }  }else {  StringBuilder query = new StringBuilder("");  query.append("CREATE TABLE " + tableFirstName.getText() + " (\n");  List<TableModel> tableModels = ((XTableView) (lineConnection.getTableFirst().getChildren().get(1))).getItems();  tableModels.forEach(row -> {  query.append(row.getId() + " " + row.getType());  if (row.isPrimaryKey()) query.append(" NOT NULL PRIMARY KEY");  if (!row.isPrimaryKey()) {  if (row.isUnique()) query.append(" UNIQUE");  if (row.isNotNull()) query.append(" NOT NULL");  }  if (!row.equals(tableModels.get(tableModels.size() - 1))) query.append(",\n");  });  StringBuilder newStringBuilder = new StringBuilder(",\n");  Pair<List<TableModel>,StringBuilder> pair = getAllTableModelFromTableWithoutKeys(lineConnection,  newStringBuilder);  newStringBuilder = pair.getValue();  if(newStringBuilder.toString().trim().length()>1){  query.append(newStringBuilder.toString());  }  query.append(");\n\n");  textArea.setText(textArea.getText()+query.toString());  }  }  }); } |

Za pomocą odpowiednich przypadków dopasowuję strategię budowanie kodu SQL.

# Wnioski

Projekt przebiegł bez zakłóceń. Założenia projektu zostały spełnione. Projekt był niezwykłym doświadczeniem, który pozwolił lepiej zrozumieć mechanizm tworzenia bazy danych. Podczas projektu wykorzystałem nabytą wiedzę z poprzedniego etapu studiów m.in. strategie dziedziczenia w przypadku diagramów ERD. Część związana z dziedziczeniem została opracowana na podstawie opracowania instrukcji do przedmiotu Projektowanie Aplikacji Internetowych 2 (PAI2) – JPA, którego autorem jest dr inż. Mariusz Bedla. Wszystkie przykłady zostały opracowane w taki sposób, aby wynikiem był kod SQL dla technologii ORACLE SQL. Wszystkie zapytania testowałem za pomocą strony: http://sqlfiddle.com