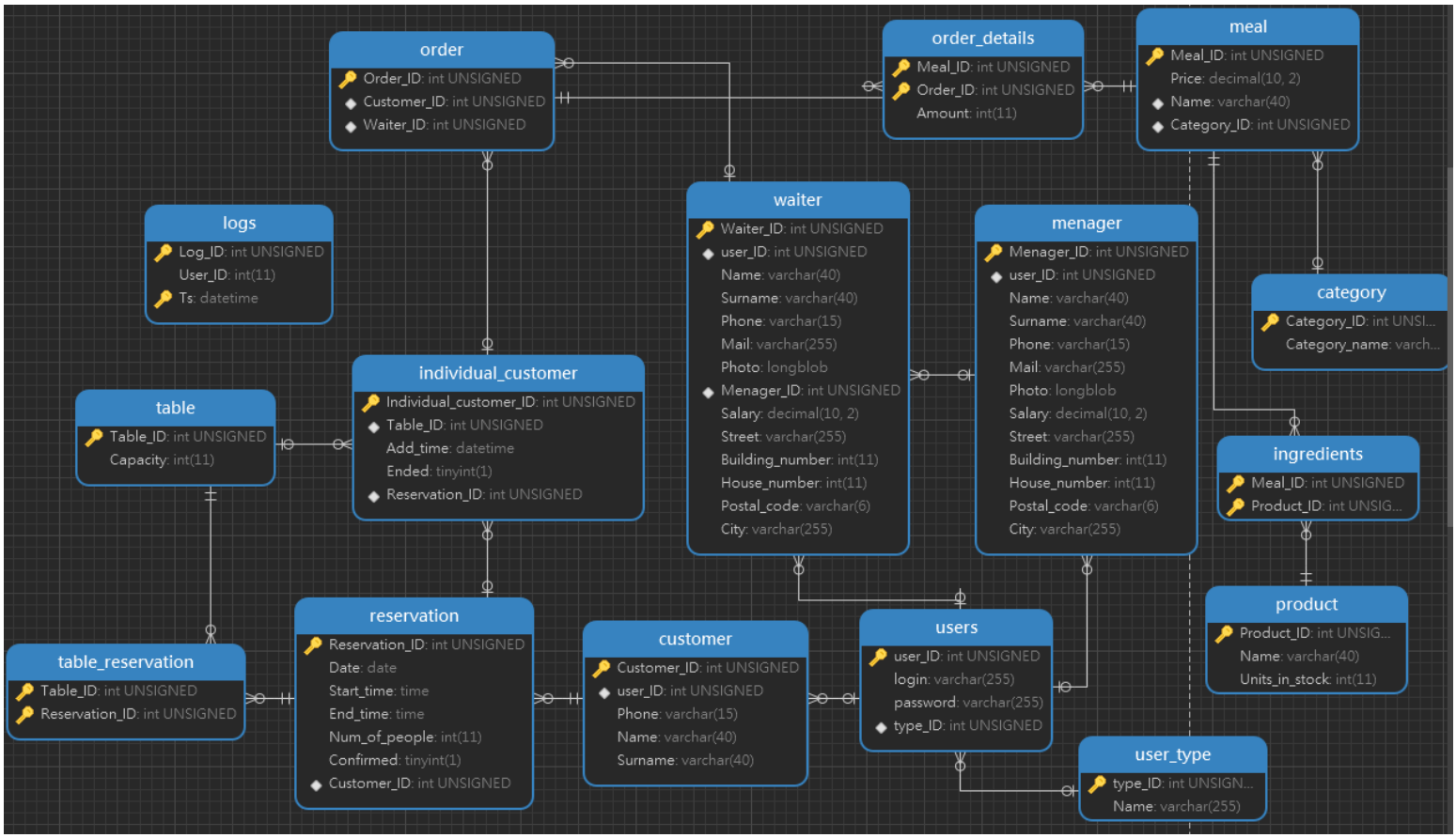
Podstawy baz danych

Projekt - restauracja

Bartosz Kniaziewicz
Dominik Kieljan

Diagram ERD bazy danych restauracji :



Opis poszczególnych tabel:

1) Menager
   - tabela zawiera dane dotyczące managera takie jak: imię, nazwisko, adres, mail jak i zdjęcie managera.
   - menager jest zdolny zarządzać kelnerami którzy są przypisani do niego za pomocą odpowiedniego numeru id, czy dodawać nowe posiłki do menu.

```
CREATE TABLE `Menager` (
  `Menager_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_ID` int(10) unsigned DEFAULT NULL,
  `Name` varchar(40) NOT NULL,
  `Surname` varchar(40) NOT NULL,
  `Phone` varchar(15) NOT NULL,
  `Mail` varchar(255) DEFAULT NULL,
  `Photo` longblob DEFAULT NULL,
  `Salary` decimal(10,2) NOT NULL,
  `Street` varchar(255) DEFAULT NULL,
  `Building_number` int(11) DEFAULT NULL,
  `House_number` int(11) DEFAULT NULL,
  `Postal_code` varchar(6) DEFAULT NULL,
  `City` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`Menager_ID`),
  KEY `user_ID` (`user_ID`),
  CONSTRAINT `menager_ibfk_1` FOREIGN KEY (`user_ID`) REFERENCES `Users` (`user_ID`),
  CONSTRAINT `mail_check` CHECK (`Mail` like '%@%.%'),
  CONSTRAINT `phone_check` CHECK (`Phone` regexp '\\+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]')
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb3;

CREATE DEFINER=`root`@`localhost` TRIGGER `first_letter_name` BEFORE INSERT ON `Menager` FOR EACH ROW IF(NEW.Name REGEXP '^[a-zA-Z.]+$')
THEN
  SET NEW.Name = CONCAT(UCASE(LEFT(NEW.Name,1)), LCASE(SUBSTR(NEW.Name,2)));
ELSE
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong input in Name field';
END IF;

CREATE DEFINER=`root`@`localhost` TRIGGER `first_letter_surname` BEFORE INSERT ON `Menager` FOR EACH ROW IF(NEW.Surname REGEXP '^[a-zA-Z.]+$')
THEN
  SET NEW.Surname = CONCAT(UCASE(LEFT(NEW.Surname,1)), LCASE(SUBSTR(NEW.Surname,2)));
ELSE
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong input in Surname field';
END IF;
```

2) Waiter

- tabela zawiera dane dotyczące kelnera takie jak: imię, nazwisko, adres, mail jak i zdjęcie kelnera.
- kelner jest zdolny do zarządzania zamówieniami.

```
CREATE TABLE `Waiter` (
  `Waiter_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_ID` int(10) unsigned DEFAULT NULL,
  `Name` varchar(40) NOT NULL,
  `Surname` varchar(40) NOT NULL,
  `Phone` varchar(15) NOT NULL,
  `Mail` varchar(255) DEFAULT NULL,
  `Photo` longblob DEFAULT NULL,
  `Menager_ID` int(10) unsigned DEFAULT NULL,
  `Salary` decimal(10,2) NOT NULL,
  `Street` varchar(255) DEFAULT NULL,
  `Building_number` int(11) DEFAULT NULL,
  `House_number` int(11) DEFAULT NULL,
  `Postal_code` varchar(6) DEFAULT NULL,
  `City` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`Waiter_ID`),
  KEY `Menager_ID` (`Menager_ID`),
  KEY `idx_Waiter_User` (`user_ID`),
  CONSTRAINT `Waiter_ibfk_1` FOREIGN KEY (`Menager_ID`) REFERENCES `Menager` (`Menager_ID`),
  CONSTRAINT `idx_Waiter_User` FOREIGN KEY (`user_ID`) REFERENCES `Users` (`user_ID`),
  CONSTRAINT `mail_check` CHECK (`Mail` like '%@%.%'),
  CONSTRAINT `phone_check` CHECK (`Phone` regexp '\\+[0-9][0-9] [0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]'),
  CONSTRAINT `salary_check` CHECK (`Salary` > 0)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8mb3;

CREATE DEFINER=`root`@`localhost` TRIGGER `first_letter_surname_w` BEFORE INSERT ON `Waiter` FOR EACH ROW IF(NEW.Surname REGEXP '^[a-zA-Z.]+$')
THEN
  SET NEW.Surname = CONCAT(UCASE(LEFT(NEW.Surname,1)), LCASE(SUBSTR(NEW.Surname,2)));
ELSE
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong input in Surname field';
END IF;

CREATE DEFINER=`root`@`localhost` TRIGGER `first_letter_name_w` BEFORE INSERT ON `Waiter` FOR EACH ROW IF(NEW.Name REGEXP '^[a-zA-Z.]+$')
THEN
  SET NEW.Name = CONCAT(UCASE(LEFT(NEW.Name,1)), LCASE(SUBSTR(NEW.Name,2)));
ELSE
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong input in Name field';
END IF;
```

3) Customer

- tabela zawiera dane dotyczące klienta takie jak: imię, nazwisko i numer telefonu.
- klient jest w stanie dokonać rezerwacji stolika.

```
CREATE TABLE `Customer` (
  `Customer_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `user_ID` int(10) unsigned DEFAULT NULL,
  `Phone` varchar(15) NOT NULL,
  `Name` varchar(40) DEFAULT NULL,
  `Surname` varchar(40) NOT NULL,
  PRIMARY KEY (`Customer_ID`),
  KEY `user_ID` (`user_ID`),
  CONSTRAINT `customer_ibfk_1` FOREIGN KEY (`user_ID`) REFERENCES `Users` (`user_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8mb3;
```

4)Reservation

- tabela zawiera dane dotyczące rezerwacji takie jak: data, czas rozpoczęcia rezerwacji, czas zakończenia rezerwacji, status potwierdzenia, liczba osób, id klienta składającego rezerwacje.
- rezerwacja jest połączona z tabelą stolików tak aby można było sprawdzić czy rezerwacje można wykonać

```
CREATE TABLE `Reservation` (
  `Reservation_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `Date` date NOT NULL,
  `Start_time` time NOT NULL,
  `End_time` time DEFAULT addtime(`Start_time`,'02:00:00'),
  `Num_of_people` int(11) NOT NULL,
  `Confirmed` tinyint(1) DEFAULT 0,
  `Customer_ID` int(10) unsigned NOT NULL,
  PRIMARY KEY (`Reservation_ID`),
  KEY `Customer_ID` (`Customer_ID`),
  CONSTRAINT `Reservation_ibfk_1` FOREIGN KEY (`Customer_ID`) REFERENCES `Customer` (`Customer_ID`),
  CONSTRAINT `time_check` CHECK (`Start_time` < `End_time`),
  CONSTRAINT `confirmation_check` CHECK (`Confirmed` = 0 or `Confirmed` = 1),
  CONSTRAINT `people_check` CHECK (`Num_of_people` > 0)
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=utf8mb3;
```

5) Individual Customer

- tabela zawiera dane dotyczące klienta zarówno tego przypisanego do rezerwacji jak i indywidualnego:
  id stolika, id rezerwacji, czas rozpoczęcia zamówienia indywidualnego klienta, status zakończenia

```
CREATE TABLE `Individual_customer` (
  `Individual_customer_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `Table_ID` int(10) unsigned DEFAULT NULL,
  `Add_time` datetime DEFAULT curtime(),
  `Ended` tinyint(1) DEFAULT 0,
  `Reservation_ID` int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (`Individual_customer_ID`),
  KEY `Table_ID` (`Table_ID`),
  KEY `idx_Individual_customer_Reservation` (`Reservation_ID`),
  CONSTRAINT `Individual_customer_ibfk_1` FOREIGN KEY (`Table_ID`) REFERENCES `Table` (`Table_ID`),
  CONSTRAINT `idx_Individual_customer_Reservation` FOREIGN KEY (`Reservation_ID`) REFERENCES `Reservation` (`Reservation_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=21 DEFAULT CHARSET=utf8mb3 WITH SYSTEM VERSIONING;
```

6) Order

- tabela zawiera takie dane jak:
  id klienta, który jest założycielem zamówienia jak i id kelnera, który się zajmuje zamówieniem.

```
CREATE TABLE `Order` (
  `Order_ID` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `Customer_ID` int(10) unsigned DEFAULT NULL,
  `Waiter_ID` int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (`Order_ID`),
  KEY `Customer_ID` (`Customer_ID`),
  KEY `Waiter_ID` (`Waiter_ID`),
  CONSTRAINT `Order_ibfk_1` FOREIGN KEY (`Customer_ID`) REFERENCES `Individual_customer` (`Individual_customer_ID`) ON DELETE SET NULL,
  CONSTRAINT `Order_ibfk_2` FOREIGN KEY (`Waiter_ID`) REFERENCES `Waiter` (`Waiter_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8mb3;
```

7) Meal

- tabela zawiera dane dotyczące posiłku takie jak: cena, nazwa, id kategorii .

```
CREATE TABLE `Meal` (
  `Meal_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `Price` decimal(10,2) NOT NULL,
  `Name` varchar(40) NOT NULL,
  `Category_ID` int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (`Meal_ID`),
  KEY `Category_ID` (`Category_ID`),
  FULLTEXT KEY `Name` (`Name`),
  CONSTRAINT `Meal_ibfk_1` FOREIGN KEY (`Category_ID`) REFERENCES `Category` (`Category_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=47 DEFAULT CHARSET=utf8mb3;
```

8) Category

- tabela zawiera dane odnośnie kategorii dań jakie mogą być np : deser, przystawka, główne danie, zupy, napoje itp.

```
CREATE TABLE `Category` (
  `Category_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `Category_name` varchar(40) DEFAULT NULL,
  PRIMARY KEY (`Category_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb3;
```

9) Ingredients

- tabela zawiera dane odnośnie składników które znajdują się w konkretnym posiłku.

```
CREATE TABLE `Ingredients` (
  `Meal_ID` int(10) unsigned NOT NULL,
  `Product_ID` int(10) unsigned NOT NULL,
  PRIMARY KEY (`Meal_ID`,`Product_ID`),
  KEY `Product_ID` (`Product_ID`),
  CONSTRAINT `Ingredients_ibfk_1` FOREIGN KEY (`Meal_ID`) REFERENCES `Meal` (`Meal_ID`),
  CONSTRAINT `Ingredients_ibfk_2` FOREIGN KEY (`Product_ID`) REFERENCES `Product` (`Product_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

10) Product

- tabela zawiera dane dotyczące produktów takie jak : nazwa, ilość na magazynie.

```
CREATE TABLE `Product` (
  `Product_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `Name` varchar(40) NOT NULL,
  `Units_in_stock` int(11) NOT NULL,
  PRIMARY KEY (`Product_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=utf8mb3;

CREATE DEFINER=`root`@`localhost` TRIGGER `Units_check` BEFORE UPDATE ON `Product` FOR EACH ROW IF(NEW.Units_in_stock < 0)
THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'There are not enough products!';
END IF;
```

11) Logs

- tabela zawiera historię akcji dokonywanych przez kelnerów, managerów oraz klientów.

```
CREATE TABLE `Logs` (
  `Log_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `User_ID` int(11) DEFAULT NULL,
  `Ts` datetime NOT NULL DEFAULT current_timestamp(),
  PRIMARY KEY (`Log_ID`,`Ts`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4
 PARTITION BY RANGE (year(`Ts`))
(PARTITION `mniejsze_od_2022` VALUES LESS THAN (2022) ENGINE = InnoDB,
 PARTITION `mniejsze_od_2023` VALUES LESS THAN (2023) ENGINE = InnoDB,
 PARTITION `wieksze_od_2022` VALUES LESS THAN MAXVALUE ENGINE = InnoDB);
```

12) Users

- tabela zawiera dane dotyczące logowania się danych pracowników do systemu takie jak:
  login, hasło jak i typ konta.

```
CREATE TABLE `Users` (
  `user_ID` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `login` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `type_ID` int(11) unsigned DEFAULT NULL,
  PRIMARY KEY (`user_ID`),
  KEY `type_ID` (`type_ID`),
  CONSTRAINT `users_ibfk_1` FOREIGN KEY (`type_ID`) REFERENCES `User_type` (`type_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=utf8mb4 WITH SYSTEM VERSIONING;
```

13) Table_reservation

- tabela łączy rezerwacje ze stolikiem zawiera dane odnośnie rezerwacji jak i stolika

```
CREATE TABLE `Table_reservation` (
  `Table_ID` int(10) unsigned NOT NULL,
  `Reservation_ID` int(10) unsigned NOT NULL,
  PRIMARY KEY (`Table_ID`,`Reservation_ID`),
  KEY `Reservation_ID` (`Reservation_ID`),
  CONSTRAINT `Table_reservation_ibfk_1` FOREIGN KEY (`Table_ID`) REFERENCES `Table` (`Table_ID`),
  CONSTRAINT `Table_reservation_ibfk_2` FOREIGN KEY (`Reservation_ID`) REFERENCES `Reservation` (`Reservation_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

14) Table

- tabela zawiera dane odnośnie stolików takie jak: pojemność i id stolika

```
CREATE TABLE `Table` (
  `Table_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `Capacity` int(11) NOT NULL,
  PRIMARY KEY (`Table_ID`),
  CONSTRAINT `capacity_check` CHECK (`Capacity` > 0)
) ENGINE=InnoDB AUTO_INCREMENT=33 DEFAULT CHARSET=utf8mb3;
```

15) Order_details

- tabela zawiera dane dotyczące zamówienia takie jak: id posiłku, id zamówienia jak i ilość posiłków

```
CREATE TABLE `Order_details` (
  `Meal_ID` int(10) unsigned NOT NULL,
  `Order_ID` int(10) unsigned NOT NULL,
  `Amount` int(11) DEFAULT NULL,
  PRIMARY KEY (`Meal_ID`,`Order_ID`),
  KEY `Order_ID` (`Order_ID`),
  CONSTRAINT `Order_details_ibfk_1` FOREIGN KEY (`Meal_ID`) REFERENCES `Meal` (`Meal_ID`),
  CONSTRAINT `Order_details_ibfk_2` FOREIGN KEY (`Order_ID`) REFERENCES `Order` (`Order_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

16) User_type

- tabela zawiera dane dotyczące typu konta takie jak : id typu konta i nazwa typu konta

```
CREATE TABLE `User_type` (
  `type_ID` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `Name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`type_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4;
```

Procedury

Add_customer

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_customer`(IN vlogin varchar(256), IN vpassword varchar(256),
  IN vname varchar(256), IN vsurname varchar(256), IN vphone varchar(256))
BEGIN
  DECLARE vuser_ID INT;

  IF vlogin IS NULL OR LENGTH(vlogin) = 0
  THEN
   SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano loginu';
  END IF;

  IF vpassword IS NULL OR LENGTH(vpassword) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano hasla';
  END IF;

  IF vname IS NULL OR LENGTH(vname) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano imienia';
  END IF;

  IF vsurname IS NULL OR LENGTH(vsurname) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano nazwiska';
  END IF;

  IF vphone IS NULL OR LENGTH(vphone) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano numeru telefonu';
  END IF;

  INSERT INTO Users (login, password, type_ID)
  VALUES (vlogin, vpassword, 3);

  SET vuser_ID = LAST_INSERT_ID();

  INSERT INTO Customer (user_ID, Name, Surname, Phone)
  VALUES (vuser_ID, vname, vsurname, vphone);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Na podstawie podanego loginu, hasła, imienia, nazwiska i numeru telefonu tworzy konto użytkownika w tabeli Users i przypisuje do niego dane o kliencie umieszczone w tabeli Customer.

Add_waiter

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_waiter`(IN vlogin varchar(256), IN vpassword varchar(256),
  IN vname varchar(256), IN vsurname varchar(256), IN vphone varchar(256), IN vsalary decimal(10,2))
BEGIN
  DECLARE vuser_ID INT;

  IF vlogin IS NULL OR LENGTH(vlogin) = 0
  THEN
   SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano loginu';
  END IF;

  IF vpassword IS NULL OR LENGTH(vpassword) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano hasla';
  END IF;

  IF vname IS NULL OR LENGTH(vname) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano imienia';
  END IF;

  IF vsurname IS NULL OR LENGTH(vsurname) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano nazwiska';
  END IF;

  IF vphone IS NULL OR LENGTH(vphone) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano numeru telefonu';
  END IF;

  INSERT INTO Users (login, password, type_ID)
  VALUES (vlogin, vpassword, 2);

  SET vuser_ID = LAST_INSERT_ID();

  INSERT INTO Waiter (user_ID, Name, Surname, Phone, Salary)
  VALUES (vuser_ID, vname, vsurname, vphone, vsalary);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Działa analogicznie jak Add_customer z tym, że dodatkowo podajemy pensję kelnera.

Add_menager

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_manager`(IN vlogin varchar(256), IN vpassword varchar(256),
  IN vname varchar(256), IN vsurname varchar(256), IN vphone varchar(256), IN vsalary decimal(10,2))
BEGIN
  DECLARE vuser_ID INT;

  IF vlogin IS NULL OR LENGTH(vlogin) = 0
  THEN
   SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano loginu';
  END IF;

  IF vpassword IS NULL OR LENGTH(vpassword) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano hasla';
  END IF;

  IF vname IS NULL OR LENGTH(vname) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano imienia';
  END IF;

  IF vsurname IS NULL OR LENGTH(vsurname) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano nazwiska';
  END IF;

  IF vphone IS NULL OR LENGTH(vphone) = 0
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Nie podano numeru telefonu';
  END IF;

  INSERT INTO Users (login, password, type_ID)
  VALUES (vlogin, vpassword, 1);

  SET vuser_ID = LAST_INSERT_ID();

  INSERT INTO Menager (user_ID, Name, Surname, Phone, Salary)
  VALUES (vuser_ID, vname, vsurname, vphone, vsalary);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Procedura dodająca nowego menadżera, działa podobnie do Add_waiter.

Add_meal

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_meal`(IN vname varchar(40), IN vprice decimal(10,2),
  IN vcategory INT, IN vmanager_ID INT)
BEGIN
  DECLARE vuser_ID INT;

  IF(CHAR_LENGTH(vname) > 0, vprice > 0, vcategory > 0, vcategory < 8)
  THEN
  INSERT INTO Meal(Price, Name, Category_ID)
  VALUES (vprice,vname,vcategory);
  END IF;

  SET vuser_ID = (SELECT user_ID FROM Menager WHERE Menager_ID = vmanager_ID);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Procedura dodaje nowy posiłek do menu.

Add_new_ingredients

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_new_ingredients`(IN vmeal_ID INT, IN vproduct_ID INT, IN vmanager_ID INT)
BEGIN
  DECLARE vuser_ID INT;

  INSERT INTO Ingredients(Meal_ID, Product_ID)
  VALUES (vmeal_ID, vproduct_ID);

  SET vuser_ID = (SELECT user_ID FROM Menager WHERE Menager_ID = vmanager_ID);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Dodanie nowych składników do pozycji z menu o danym ID.

Add_individual_customer

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_individual_customer`(IN vTable_ID INT)
BEGIN
  INSERT INTO Individual_customer (Table_ID)
  VALUES (vTable_ID);
END
```

Dodanie nowego klienta indywidualnego, musimy podać ID wolnego stolika, a takie możemy wyświetlić przy pomocy widoku free_tables_now.

Add_order

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_order`(IN vcustomer_ID int, IN vwaiter_ID int)
BEGIN
  DECLARE vuser_ID INT;

  INSERT INTO `Order` (Customer_ID, Waiter_ID)
  VALUES (vcustomer_ID, vwaiter_ID);

  SET vuser_ID = (SELECT user_ID FROM Waiter WHERE Waiter_ID = vwaiter_ID);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Utworzenie nowego zamówienia dla klienta indywidualnego.

Add_meal_to_order

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_meal_to_order`(IN vorder_ID int, IN vmeal_ID int, IN vamount int)
BEGIN
  DECLARE prod_ID INT;
  DECLARE done INT;
  DECLARE vuser_ID INT;

  DECLARE productInfo CURSOR FOR
  SELECT Product_ID FROM Ingredients
  WHERE Meal_ID = vmeal_ID;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN productInfo;

  update_prod: LOOP
  FETCH productInfo INTO prod_ID;

  IF done THEN
    LEAVE update_prod;
  END IF;

  UPDATE Product
  SET Units_in_stock = Units_in_stock - vamount
  WHERE Product_ID = prod_ID;

END LOOP update_prod;

CLOSE productInfo;

  INSERT INTO Order_details
  VALUES (vmeal_ID, vorder_ID, vamount);

  SET vuser_ID = (SELECT user_ID FROM Waiter WHERE Waiter_ID = (SELECT Waiter_ID FROM `Order` WHERE Order_ID = vorder_ID));

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);

END
```

Dodanie do zamówienia pozycji z menu o danym ID w podanej ilości. Dodatkowo modyfikuje tabele ze składnikami, po dodaniu do zamówienia zmniejsza się ilość dostępnych składników.

Add_reservation

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_reservation`(IN vdate date, IN vstart time, IN vend time,
  IN vcustomers_num INT, IN vcustomer_ID INT)
BEGIN

  DECLARE vuser_ID INT;

  IF vend IS NULL OR LENGTH(vend) = 0 THEN
  SET vend := ADDTIME(vstart,'02:00:00');
  END IF;

  IF(vdate > CURDATE())
  THEN
    INSERT INTO Reservation (Date, Start_time, End_time, Num_of_people, Confirmed, Customer_ID)
    VALUES (vdate, vstart, vend, vcustomers_num, 0, vcustomer_ID);
  ELSE
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Rezerawacja musi być wykonana z przynajmiej jednodniowym wyprzedzeniem!';
  END IF;

  SET vuser_ID = (SELECT user_ID FROM Customer WHERE Customer_ID = vcustomer_ID);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Dodanie nowej rezerwacji dla klienta o danym ID. Konieczne jest podanie daty rezerwacji (pamiętając o przynajmniej jednodniowym wyprzedzeniu) oraz o godzinie rozpoczęcia, jeśli nie podamy godziny zakończenia rezerwacja jest dokonywana na dwie godziny.

Add_table_to_reservation

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_table_to_reservation`(IN vReservation_ID int, IN vTable_ID int)
BEGIN

  DECLARE start_hour TIME;
  DECLARE res_date DATE;
  DECLARE customers_num INT;

  IF(vTable_ID IS NULL)
  THEN
    START TRANSACTION;

    SET start_hour = (SELECT Start_time FROM Reservation WHERE Reservation_ID = vReservation_ID);
    SET res_date = (SELECT Date FROM Reservation WHERE Reservation_ID = vReservation_ID);
    SET customers_num = (SELECT Num_of_people FROM Reservation WHERE Reservation_ID = vReservation_ID FOR UPDATE);

    SET vTable_ID = Get_table_for_reservation(res_date, start_hour, customers_num);

     IF(vTable_ID IS NULL)
     THEN
      SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Brak wolnych stolików!';
      ROLLBACK;
     ELSE
      INSERT INTO Table_reservation
      VALUES (vTable_ID, vReservation_ID);

      UPDATE Reservation SET Confirmed = 1 WHERE Reservation_ID = vReservation_ID;
      COMMIT;
     END IF;
  ELSE
    INSERT INTO Table_reservation
    VALUES (vTable_ID, vReservation_ID);

    UPDATE Reservation SET Confirmed = 1 WHERE Reservation_ID = vReservation_ID;
  END IF;

END
```

Dodanie stolika odpowiedniego dla danej rezerwacji, jeśli nie podano ID stolika to jest on wyszukiwany przy pomocy funkcji Get_table_for_reservation.

Add_order_to_reservation

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Add_order_to_reservation`(IN vReservation_ID int, IN vWaiter_ID int)
BEGIN
  DECLARE orders_num INT; -- aktualna liczba zamowien przypisana do rezerwacji
  DECLARE orders_max INT; -- maksymalna liczba zamowien dla danej rezerwacji = liczbie osob
  DECLARE vTable_ID INT;
  DECLARE vuser_ID INT;

  SET orders_num = (SELECT COUNT(*) FROM Individual_customer WHERE Reservation_ID = vReservation_ID);
  SET orders_max = (SELECT Num_of_people FROM Reservation WHERE Reservation_ID = vReservation_ID);
  SET vTAble_ID = (SELECT Table_ID FROM Table_reservation WHERE Reservation_ID = vReservation_ID);

  IF (orders_num = orders_max)
  THEN
    SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO = 30001, MESSAGE_TEXT = 'Osiagnieto limit';
  ELSE
    INSERT INTO Individual_customer (Table_ID, Reservation_ID)
    VALUES (vTable_ID, vReservation_ID);

    INSERT INTO `Order` (Customer_ID, Waiter_ID)
    VALUES (LAST_INSERT_ID(), vWaiter_ID);
END IF;

  SET vuser_ID = (SELECT user_ID FROM Waiter WHERE Waiter_ID = vwaiter_ID);

  INSERT INTO Logs (User_ID)
  VALUES (vuser_ID);
END
```

Każda osoba z rezerwacji jest traktowana jako osobny klient. Procedura tworzy nowy rekord w tabeli Individual_customer i przypisuje mu ID rezerwacji. Do rezerwacji nie możemy przypisać więcej zamówień niż liczba osób na które dokonana jest rezerwacja.

SumUp_order

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `SumUp_order`(IN vOrder_ID int)
BEGIN
  DECLARE quantity INT;
  DECLARE calculated_price DECIMAL(10,2);
  DECLARE meal_name VARCHAR(255);
  DECLARE cus_ID INT;

  SELECT ml.`Name`, ml.Price, od.Amount FROM Meal AS ml
  INNER JOIN Order_details as od ON ml.Meal_ID = od.Meal_ID
  WHERE od.Order_ID = vOrder_ID
  GROUP BY Name, Price, Amount;

  SET calculated_price = Order_total_price(vOrder_ID);

  SELECT calculated_price;

  SET cus_ID = (SELECT Customer_ID FROM `Order` WHERE Order_ID = vOrder_ID);

  UPDATE Individual_customer SET Ended = 1 WHERE Individual_customer_ID;
END
```

Podsumowanie zamówienia o danym ID. Wyświetla nazwy produktów, które zostały zamówione, ich cenę oraz ilość. Dodatkowo wyświetlana jest cena całego zamówienia obliczona przy pomocy funkcji Order_total_price;

Get_reservation_total_value

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Get_reservation_total_value`(IN vReservation_ID int)
BEGIN
  DECLARE done INT;
  DECLARE cus_ID INT;
  DECLARE ord_ID INT;
  DECLARE part_sum DECIMAL(10,2) DEFAULT 0.00;
  DECLARE total_sum DECIMAL(10,2) DEFAULT 0.00;

  DECLARE customersInfo CURSOR FOR
  SELECT Individual_customer_ID FROM Individual_customer WHERE Reservation_ID = vReservation_ID;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN customersInfo;

    sum_up: LOOP
    FETCH customersInfo INTO cus_ID;

    IF done THEN
      LEAVE sum_up;
    END IF;

    SET ord_ID = (SELECT Order_ID FROM `Order` WHERE Customer_ID = cus_ID);
    SET part_sum = Order_total_price(ord_ID);
    IF (part_sum IS NOT NULL)
    THEN
      SET total_sum = total_sum + part_sum;
    END IF;

  END LOOP sum_up;

  CLOSE customersInfo;

  SELECT total_sum;
END
```

Wyświetla zsumowaną wartość każdego z zamówień przypisanych do rezerwacji.

Find_in_menu

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Find_in_menu`(IN vname varchar(256))
BEGIN
  SELECT Name, Price FROM Meal WHERE MATCH(Name) against(vname);
END
```

Wyszukiwanie pozycji w menu wykorzystujące indeks Full-Text.

Funkcje

Order_total_price

```
CREATE DEFINER=`root`@`localhost` FUNCTION `Order_total_price`(vOrder_ID int) RETURNS decimal(10,2)
BEGIN
  DECLARE result DECIMAL(10,2);

  SET result = (SELECT SUM(od.Amount * ml.Price) FROM Order_details AS od
                INNER JOIN Meal ml ON ml.Meal_ID = od.Meal_ID
                WHERE od.Order_ID = vOrder_ID);
RETURN result;
END
```

Oblicza całkowitą wartość zamówienia o podanym ID.

Get_table_for_reservation

```
CREATE DEFINER=`root`@`localhost` FUNCTION `Get_table_for_reservation`(res_date date,  start_hour time,  clients_num int) RETURNS int(11)
BEGIN
  DECLARE founded_ID INT;

  SET founded_ID = (SELECT t.Table_ID FROM `Table` AS t
  WHERE t.Capacity >= clients_num
  AND t.Table_ID NOT IN (
    SELECT tr.Table_ID FROM `Table_reservation` AS tr
    INNER JOIN `Reservation` AS r ON r.Reservation_ID = tr.Reservation_ID
    WHERE r.Date = res_date
    AND NOT ((r.Start_time <= start_hour AND r.End_time <= start_hour)
             OR (r.Start_time >= ADDTIME(start_hour, '02:00:00') AND r.End_time >= ADDTIME(start_hour,'02:00:00'))))
  LIMIT 1);

RETURN founded_ID;
END
```

Wyszukuje wolnego stolika dla podanej ilości osób i zwraca jego ID.

Widoki

free_tables_now

```
(SELECT t.Table_ID FROM `Table` AS t
  WHERE t.Table_ID NOT IN (
    SELECT tr.Table_ID FROM `Table_reservation` AS tr
    INNER JOIN `Reservation` AS r ON r.Reservation_ID = tr.Reservation_ID
    WHERE r.Date = CURDATE()
    AND NOT ((r.Start_time <= CURTIME() AND r.End_time <= CURTIME())
             OR (r.Start_time >= ADDTIME(CURTIME(), '02:00:00') AND r.End_time >= ADDTIME(CURTIME(),'02:00:00')))
    ))
EXCEPT
(SELECT Table_ID FROM Individual_customer
WHERE Ended = 0 AND DATE(Add_time) = CURDATE());
```

Wyświetla stoliki które są wolne w danym momencie i będą wolne przez następne dwie godziny.

Show_reserver_tables

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Show_reserved_tables`(IN vdate date)
BEGIN
  SELECT tab.Table_ID, res.Start_time, res.End_time, res.Date FROM Reservation res
  INNER JOIN Table_reservation tab_res ON tab_res.Reservation_ID = res.Reservation_ID
  INNER JOIN `Table` tab ON tab.Table_ID = tab_res.Table_ID
  WHERE res.Date = vdate;
END
```

Wyświetla listę stolików oraz godziny w jakich będą zarezerwowane w danym dniu.

Show_ingredients_of_meal

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ShowIngredientsOfMeal`(IN meal_ID int)
BEGIN
  SELECT  p.Name AS 'Składniki' FROM Ingredients i
  INNER JOIN Product p ON p.Product_ID = i.Product_ID
  WHERE i.Meal_ID = meal_ID;
END
```

Wyświetla składniki pozycji z menu o danym ID.

Wyzwalacze

first_letter_name

```
IF(NEW.Name REGEXP '^[a-zA-Z.]+$')
THEN
  SET NEW.Name = CONCAT(UCASE(LEFT(NEW.Name,1)), LCASE(SUBSTR(NEW.Name,2)));
ELSE
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wrong input in Name field';
END IF
```

Sprawdza czy wartość którą podajemy jako imię napewno zawiera tylko litery i dodatkowo ustawia pierwszą literę na dużą a resztę na małe. Wykorzystywany np. w tabelach Waiter i Menager.
Analogiczny trigger sprawdza nazwisko.

Units_check

```
IF(NEW.Units_in_stock < 0)
THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'There are not enough products!';
END IF
```

Jeśli dodanie pozycji z menu do zamówienia, spowoduje że ilość któregoś produktu spadnie poniżej 0 wtedy dodanie tego posiłku zostaje przerwane.

Zdarzenia

Clear_individual_customers

```
CREATE DEFINER=`root`@`localhost` EVENT `Clear_individual_customers`
ON SCHEDULE EVERY 1 DAY STARTS '2022-06-14 14:48:00' ENDS '2022-09-14 14:40:00'
ON COMPLETION PRESERVE
ENABLE
DO DELETE FROM Individual_customer WHERE DATEDIFF(NOW(), Add_time) > 90;
```

Usuwa z tabeli klientów którzy składali zamówienie ponad 3 miesiące temu.

Monday_delivery

```
CREATE DEFINER=`root`@`localhost` EVENT `Monday_delivery`
ON SCHEDULE EVERY 7 DAY STARTS '2022-06-06 15:45:00' ENDS '2022-09-06 15:45:00'
ON COMPLETION NOT PRESERVE
ENABLE
DO UPDATE Product SET Units_in_stock = Units_in_stock + 10;
```

Co poniedziałek uzupełnia stan magazynu.