Podstawy baz danych

System zarządzania konferencjami

Dokumentacja

Sławomir Mucha, Patryk Mrukot

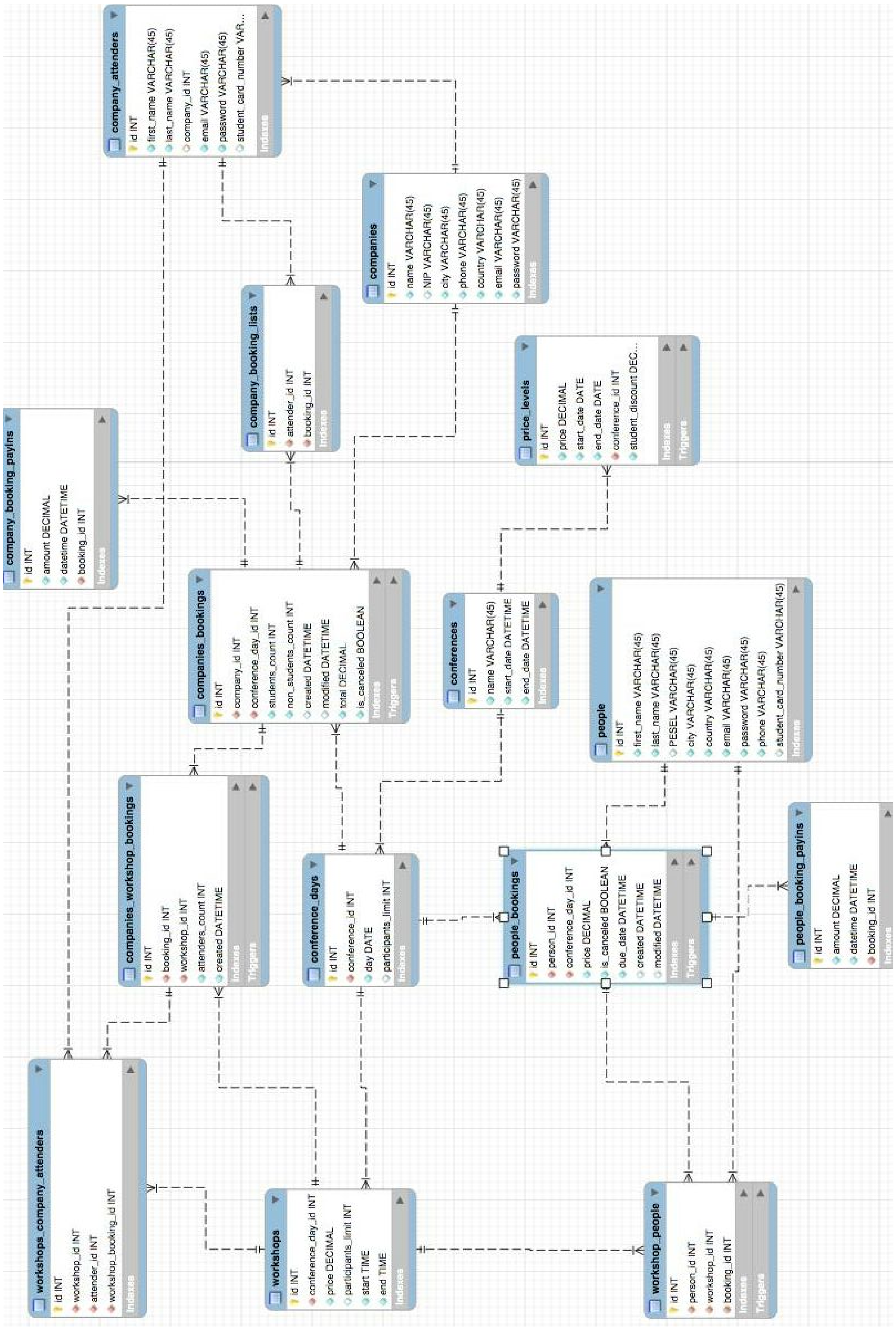
1. Opis funkcji systemu

Baza danych dotyczny firmy zajmującej się organizowaniem koferencji - jedno lub kilkudniowych. Podczas trwania konferencji mogą odbywać się jednodniowe warsztaty. Zarówno na konferencje jak i na warsztaty mogą zapisywać się osoby indywidualne jak i firmy - podając listę uczestników z danej firmy. Klienci (firmy oraz klienci indywidualni) rejestrują się za pomocą systemu www. Organizatorzy konferencji mają dostęp do list osobowych uczestników oraz informację o klientach najczęściej korzystających z usług.

Użytkownicy
- klient indywidualny: rejestracja w systemie, rezerwacja dnia konferencji, rezerwacja warsztatu, dokonywanie wpłat
- klient jako firma: rejestracja w systemie, rezerwacja dnia konferencji, uzupełnianie listy uczestników, zapis uczestników firmy na warsztaty
- administrator (superuser)
- organizator konferencji : CRUD konferencji, CRUD listy uzytkowników, akceptacja bookingów, CRUD dnia konferencji
- uczestnik - może sprawdzać na co jest zapisany

Wykorzystana baza danych: MySQL

2. Schemat bazy danych



3. Opis tabel

- conferences

```
CREATE TABLE IF NOT EXISTS `conferences`.`conferences` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  `start_date` DATETIME NOT NULL,
  `end_date` DATETIME NOT NULL,
  PRIMARY KEY (`id`))
```

Tabela opisuje odbywające się konferencje. Wyróżniamy id jako klucz podstawowy, name - nazwę konferencji oraz przedział czasowy trwania konferencji - atrybuty start_date oraz end_date

- price_levels

```
CREATE TABLE IF NOT EXISTS `conferences`.`price_levels` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `price` DECIMAL NOT NULL,
  `start_date` DATE NOT NULL,
  `end_date` DATE NOT NULL,
  `conference_id` INT NOT NULL,
  `student_discount` DECIMAL NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`),
  INDEX `conferences_fk_idx` (`conference_id` ASC),
  CONSTRAINT `conferences_fk`
    FOREIGN KEY (`conference_id`)
    REFERENCES `conferences`.`conferences` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuje progi cenowe opłat za poszczególne konferencje, dla poszczególnych przedziałów czasowych - im wcześniej, tym cena jest niższa. Wyróżniamy atrybuty: id - klucz podstawowy, price - opłata za udział w konferencji, start_date, end_date - przedział czasowy dla danego progu, conference_id - identyfikator konferencji.

- conference_days

```
CREATE TABLE IF NOT EXISTS `conferences`.`conference_days` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `conference_id` INT NOT NULL,
  `day` DATE NOT NULL,
  `participants_limit` INT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_conferences_idx` (`conference_id` ASC),
  CONSTRAINT `fk_conferences`
    FOREIGN KEY (`conference_id`)
    REFERENCES `conferences`.`conferences` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuje poszczególne dni dla danej konferencji. Atrybuty: id, conference_id, day - dzień konferencji, participants_limit - limit uczestników danego dnia konferencji , price - cena za dzień konferecji, discount - zniżka.

- workshops

```
CREATE TABLE IF NOT EXISTS `conferences`.`workshops` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `conference_day_id` INT NOT NULL,
  `price` DECIMAL NOT NULL,
  `participants_limit` INT NULL,
  `start` TIME NOT NULL,
  `end` TIME NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_conferences_idx` (`conference_day_id` ASC),
  CONSTRAINT `fk_conferences`
    FOREIGN KEY (`conference_day_id`)
    REFERENCES `conferences`.`conference_days` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuje warsztaty odbywające się w poszczególne dni konferencji , ich cenę oraz limit uczestników.

- people

```
CREATE TABLE IF NOT EXISTS `conferences`.`people` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `PESEL` VARCHAR(45) NULL,
  `city` VARCHAR(45) NOT NULL,
  `country` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `phone` VARCHAR(45) NOT NULL,
  `student_card_number` VARCHAR(45) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `PESEL_UNIQUE` (`PESEL` ASC),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC))
```

Tabela opisuje dane klientów indywidualnych. Atrybuty tabeli to first_name, last_name - imię i nazwisko uczestników, PESEL, city, country - adress, email oraz password - hasło.

- people_bookings

```
CREATE TABLE IF NOT EXISTS `conferences`.`people_bookings` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `person_id` INT NOT NULL,
  `conference_day_id` INT NOT NULL,
  `price` DECIMAL NOT NULL,
  `is_canceled` TINYINT(1) NOT NULL DEFAULT 0,
  `due_date` DATETIME NOT NULL,
  `created` DATETIME NULL,
  `modified` DATETIME NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_people_1_idx` (`person_id` ASC),
  INDEX `fk_conferences_2_idx` (`conference_day_id` ASC),
  CONSTRAINT `fk_people_1`
    FOREIGN KEY (`person_id`)
    REFERENCES `conferences`.`people` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_conferences_2`
    FOREIGN KEY (`conference_day_id`)
    REFERENCES `conferences`.`conference_days` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuję rezerwację poszczególnych klientów indywidualnych na poszczególne dni konferencji. Opisuję również wpłaconą kwotę, czy rezerwacja jest odwołana, oraz termin wygaśnięcia - due_date

- people_bookings_payins

```
CREATE TABLE IF NOT EXISTS `conferences`.`people_booking_payins` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `amount` DECIMAL NOT NULL,
  `datetime` DATETIME NOT NULL,
  `booking_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `people_booking_fk_idx` (`booking_id` ASC),
  CONSTRAINT `people_booking_fk`
    FOREIGN KEY (`booking_id`)
    REFERENCES `conferences`.`people_bookings` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuję wpłaty klientów na poszczególne rezerwację. Klient może dokonywać wpłat ratami (payin). Wyróżniamy atrybuty id, amount - wpłacona kwota, booking_id - id rezerwacji, oraz datetime - date wpłaty.

- workshop_people

```
CREATE TABLE IF NOT EXISTS `conferences`.`workshop_people` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `person_id` INT NOT NULL,
  `workshop_id` INT NOT NULL,
  `booking_id` INT NOT NULL,
  INDEX `fk_people_idx` (`person_id` ASC),
  INDEX `fk_bookings_idx` (`booking_id` ASC),
  INDEX `fk_workshops_idx` (`workshop_id` ASC),
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_people`
    FOREIGN KEY (`person_id`)
    REFERENCES `conferences`.`people` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_workshops`
    FOREIGN KEY (`workshop_id`)
    REFERENCES `conferences`.`workshops` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_bookings`
    FOREIGN KEY (`booking_id`)
    REFERENCES `conferences`.`people_bookings` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela przejściowa między tabelą **workshops** a tabelą **people** - jeden warsztat może mieć wielu uczestników, jeden klient może uczestniczyć w wielu warsztatach. Wyróżniamy atrybuty: person_id, workshop_id oraz booking_id

- *companies*

```
CREATE TABLE IF NOT EXISTS `conferences`.`companies` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  `NIP` VARCHAR(45) NULL,
  `city` VARCHAR(45) NOT NULL,
  `phone` VARCHAR(45) NOT NULL,
  `country` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `NIP_UNIQUE` (`NIP` ASC),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC))
```

Tabela opisuję firmy, które zapisują pracowników na konferencję. Tabela ta zawiera informację na temat firm takie jak: id, name, NIP, city, phone, country, email czy password

- *company_attenders*

```
CREATE TABLE IF NOT EXISTS `conferences`.`company_attenders` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `company_id` INT NULL,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `student_card_number` VARCHAR(45) NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_companies_idx` (`company_id` ASC),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC),
  CONSTRAINT `fk_companies`
    FOREIGN KEY (`company_id`)
    REFERENCES `conferences`.`companies` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuję uczestników zapisanych na konferencję z danej firmy. Posiada dane na temat pracowników, takie jak: first_name, last_name, company_id, email, password czy student_card_number (w celach uzyskania zniżki)

- *companies_bookings*

```
CREATE TABLE IF NOT EXISTS `conferences`.`companies_bookings` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `company_id` INT NOT NULL,
  `conference_day_id` INT NOT NULL,
  `students_count` INT NOT NULL,
  `non_students_count` INT NOT NULL DEFAULT 0,
  `created` DATETIME NULL,
  `modified` DATETIME NULL,
  `total` DECIMAL NOT NULL,
  `is_canceled` TINYINT(1) NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`),
  INDEX `fk_conference_days_idx` (`conference_day_id` ASC),
  CONSTRAINT `fk_companies_1`
    FOREIGN KEY (`company_id`)
    REFERENCES `conferences`.`companies` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_conference_days`
    FOREIGN KEY (`conference_day_id`)
    REFERENCES `conferences`.`conference_days` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela opisuję rezerwację dokonaną przez firmę na dany dzień konferencji. Posiada informację na temat ilości uczestników - studentów i nie-studentów, kiedy została utworzona rezerwacja, kiedy zsotałą zmodyfikowana, oraz czy rejestracja została odwołana.

- *companies_bookings_list*

```
CREATE TABLE IF NOT EXISTS `conferences`.`company_booking_lists` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `attender_id` INT NOT NULL,
  `booking_id` INT NOT NULL,
  INDEX `fk_company_attenders_idx` (`attender_id` ASC),
  INDEX `fk_companies_bookings_idx` (`booking_id` ASC),
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_company_attenders`
    FOREIGN KEY (`attender_id`)
    REFERENCES `conferences`.`company_attenders` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_companies_bookings`
    FOREIGN KEY (`booking_id`)
    REFERENCES `conferences`.`companies_bookings` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela przejściowa między tabelami **company_attenders** a **companies_bookings**, zawiera id uczestnika oraz id rezerwacji

- *companies_booking_payins*

```
CREATE TABLE IF NOT EXISTS `conferences`.`company_booking_payins` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `amount` DECIMAL NOT NULL,
  `datetime` DATETIME NOT NULL,
  `booking_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `company_booking_id_idx` (`booking_id` ASC),
  CONSTRAINT `company_booking_id`
    FOREIGN KEY (`booking_id`)
    REFERENCES `conferences`.`companies_bookings` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela zawiera informację na temat wpłat dokonanych przez firmę na daną rezerwację. Zakłada się, że wpłaty można dokonywać w ratach. Tabela zawiera informację na temat wpłaconej kwoty oraz timestamp

- companies_workshop_bookings

```
CREATE TABLE IF NOT EXISTS `conferences`.`companies_workshop_bookings` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `booking_id` INT NOT NULL,
  `workshop_id` INT NOT NULL,
  `attenders_count` INT NOT NULL,
  `created` DATETIME NOT NULL,
  INDEX `companies_bookings_fk_idx` (`booking_id` ASC),
  UNIQUE INDEX `unique_booking` (`booking_id` ASC, `workshop_id` ASC),
  PRIMARY KEY (`id`),
  INDEX `workshops_fk_idx` (`workshop_id` ASC),
  CONSTRAINT `companies_bookings_fk`
    FOREIGN KEY (`booking_id`)
    REFERENCES `conferences`.`companies_bookings` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `workshops_fk`
    FOREIGN KEY (`workshop_id`)
    REFERENCES `conferences`.`workshops` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela zawiera informację na temat rezerwacji uczestników przez firmy na poszcególne warsztaty. Zawiera informację na temat daty utworzenia rezerwacji oraz liczby uczestników.

- workshops_company_attenders

```
CREATE TABLE IF NOT EXISTS `conferences`.`workshops_company_attenders` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `workshop_id` INT NOT NULL,
  `attender_id` INT NOT NULL,
  `workshop_booking_id` INT NOT NULL,
  INDEX `fk_workshops_idx` (`workshop_id` ASC),
  INDEX `fk_company_attenders_idx` (`attender_id` ASC),
  PRIMARY KEY (`id`),
  INDEX `fk_bookings_idx` (`workshop_booking_id` ASC),
  CONSTRAINT `fk_company_attenders`
    FOREIGN KEY (`attender_id`)
    REFERENCES `conferences`.`company_attenders` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_workshops`
    FOREIGN KEY (`workshop_id`)
    REFERENCES `conferences`.`workshops` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_bookings`
    FOREIGN KEY (`workshop_booking_id`)
    REFERENCES `conferences`.`companies_workshop_bookings` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

Tabela przejściowa między tabelami **company_attenders** a **workshops**, zawiera workshop_id, attender_id oraz workshop_booking_id.

4. Widoki

- ***conferences_pricing***

Widok jest zestawieniem progów cenowych dla wszystkich konferencji. W widoku mamy informacji na temat ceny w danym przedziale czasowym, zniżki studenckiej, przedział czasowy w którym obowiązuje dana cena oraz nazwę konferencji

```
CREATE VIEW `conferences`.`conferences_pricing` AS
   SELECT
      `conferences`.`price_levels`.`price` AS `price`,
      `conferences`.`price_levels`.`student_discount` AS `student_discount`,
      `conferences`.`price_levels`.`start_date` AS `start_date`,
      `conferences`.`price_levels`.`end_date` AS `end_date`,
      `conferences`.`conferences`.`name` AS `name`
   FROM
      (`conferences`.`price_levels`
      JOIN `conferences`.`conferences` ON ((`conferences`.`conferences`.`id` =
`conferences`.`price_levels`.`conference_id`)))
   ORDER BY `conferences`.`conferences`.`name`;
```

- ***workshops_pricing***

Widok jest zestawieniem kosztów warsztatów. Uwzględnia on nazwe konferencji w ramach której odbywa się warsztat, dzień, przedział czasowy oraz cene warsztatu.

```
CREATE VIEW `conferences`.`workshops_pricing` AS
   SELECT
      `conferences`.`conferences`.`name` AS `name`,
      `conferences`.`conference_days`.`day` AS `day`,
      `conferences`.`workshops`.`start` AS `start`,
      `conferences`.`workshops`.`end` AS `end`,
      CONCAT(CAST(`conferences`.`workshops`.`price` AS CHAR (50) CHARSET UTF8),
            'PLN') AS `cost`,
      CONCAT(CAST((HOUR(`conferences`.`workshops`.`end`) -
HOUR(`conferences`.`workshops`.`start`))
               AS CHAR (50) CHARSET UTF8),
            'h') AS `length`
   FROM
      ((`conferences`.`workshops`
      JOIN `conferences`.`conference_days` ON ((`conferences`.`conference_days`.`id` =
`conferences`.`workshops`.`conference_day_id`)))
      JOIN `conferences`.`conferences` ON ((`conferences`.`conferences`.`id` =
`conferences`.`conference_days`.`conference_id`)))
   ORDER BY `conferences`.`conferences`.`name` , `conferences`.`conference_days`.`day` ,
`conferences`.`workshops`.`start`;
```

- ***free_workshops***

Widok zawiera dane na temat warsztatów odbywających się za darmo. Zawiera on nazwę konferencji, date rozpoczęcia oraz zakończenia warsztatu oraz jego długość.

```
VIEW `conferences`.`free_workshops` AS
   SELECT
       `conferences`.`conferences`.`name` AS `name`,
       `conferences`.`conference_days`.`day` AS `day`,
       `conferences`.`workshops`.`start` AS `start`,
       `conferences`.`workshops`.`end` AS `end`,
       CONCAT(CAST((HOUR(`conferences`.`workshops`.`end`) -
HOUR(`conferences`.`workshops`.`start`))
               AS CHAR (50) CHARSET UTF8), 'h') AS `length`
   FROM
       ((`conferences`.`workshops`
       JOIN `conferences`.`conference_days` ON ((`conferences`.`conference_days`.`id` =
`conferences`.`workshops`.`conference_day_id`)))
       JOIN `conferences`.`conferences` ON ((`conferences`.`conferences`.`id` =
`conferences`.`conference_days`.`conference_id`)))
   WHERE
       (`conferences`.`workshops`.`price` = 0)
   ORDER BY `conferences`.`conferences`.`name` , `conferences`.`conference_days`.`day` ,
`conferences`.`workshops`.`start`;
```

- ***most_popular_conferences***

Widok wyświetla najpopularniejsze koferencje (posortowane od najpopularniejszej) oraz całkowitą liczbę uczestników.

```
CREATE VIEW `conferences`.`most_popular_conferences` AS
   SELECT
       (SUM((`cb`.`students_count` + `cb`.`non_students_count`)) + COUNT(`pb`.`person_id`)) AS
`total_participants`,
       `c`.`name` AS `name`
   FROM
       (((`conferences`.`companies_bookings` `cb`
       JOIN `conferences`.`conference_days` `cd` ON ((`cd`.`id` = `cb`.`conference_day_id`)))
       JOIN `conferences`.`conferences` `c` ON ((`c`.`id` = `cd`.`conference_id`)))
       JOIN `conferences`.`people_bookings` `pb` ON ((`pb`.`conference_day_id` = `cd`.`id`)))
   GROUP BY `c`.`name`
   ORDER BY (SUM((`cb`.`students_count` + `cb`.`non_students_count`)) +
COUNT(`pb`.`person_id`)) DESC
   LIMIT 10;
```

- *students_personal_data*

Widok pokazuje dane osobowe wszystkich studentów, zarówno idących na konferencję/warsztaty indywidualnie jak i tych zapisanych przez firmę.

```
CREATE VIEW `conferences`.`students_personal_data` AS
   SELECT
      `conferences`.`company_attenders`.`first_name` AS `first_name`,
      `conferences`.`company_attenders`.`last_name` AS `last_name`,
      `conferences`.`company_attenders`.`student_card_number` AS `student_card_number`,
      `conferences`.`company_attenders`.`email` AS `email`
   FROM
      `conferences`.`company_attenders`
   WHERE
      (`conferences`.`company_attenders`.`student_card_number` IS NOT NULL)
   UNION SELECT
      `conferences`.`people`.`first_name` AS `first_name`,
      `conferences`.`people`.`last_name` AS `last_name`,
      `conferences`.`people`.`student_card_number` AS `student_card_number`,
      `conferences`.`people`.`email` AS `email`
   FROM
      `conferences`.`people`
   WHERE
      (`conferences`.`people`.`student_card_number` IS NOT NULL);
```

- *top_donators*

Widok zawiera imiona i nazwiska osób, które przez cały okres działalności firmy organizującej konferencję wydali ponad 25000 złotych.

```
VIEW `conferences`.`top_donators` AS
   SELECT
      SUM(`conferences`.`people_booking_payins`.`amount`) AS `total_paid`,
      CONCAT(`conferences`.`people`.`first_name`,
            ' ',
            `conferences`.`people`.`last_name`) AS `person`
   FROM
      ((`conferences`.`people_booking_payins`
      JOIN `conferences`.`people_bookings` ON ((`conferences`.`people_bookings`.`id` =
`conferences`.`people_booking_payins`.`booking_id`)))
      JOIN `conferences`.`people` ON ((`conferences`.`people`.`id` =
`conferences`.`people_bookings`.`person_id`)))
   GROUP BY `conferences`.`people_bookings`.`person_id`
   HAVING (SUM(`conferences`.`people_booking_payins`.`amount`) > 25000)
   ORDER BY SUM(`conferences`.`people_booking_payins`.`amount`) DESC;
```

- *cancelled_companies_bookings_employees_personal_data*

Widok zawiera dane osobowe pracowników zarejestrowanych przez firmę, których rezerwacja została anulowana.

```
VIEW `conferences`.`cancelled_companies_bookings_employees_personal_data` AS
  SELECT
    `c`.`name` AS `name`,
    `ca`.`first_name` AS `first_name`,
    `ca`.`last_name` AS `last_name`,
    `ca`.`email` AS `email`
  FROM
    (((((`conferences`.`conferences` `c`
    JOIN `conferences`.`conference_days` `cd` ON ((`cd`.`conference_id` = `c`.`id`)))
    JOIN `conferences`.`companies_bookings` `cb` ON ((`cb`.`conference_day_id` = `cd`.`id`)))
    JOIN `conferences`.`company_booking_lists` `cbl` ON ((`cbl`.`booking_id` = `cb`.`id`)))
    JOIN `conferences`.`company_attenders` `ca` ON ((`ca`.`id` = `cbl`.`attender_id`)))
    JOIN `conferences`.`companies` `cmp` ON ((`ca`.`company_id` = `cmp`.`id`)))
  WHERE
    (`cb`.`is_canceled` = 1)
  GROUP BY `ca`.`first_name` , `ca`.`last_name`
  ORDER BY `ca`.`last_name` , `ca`.`first_name`;
```

- ***workshop_limits***

Widok zwiera łączną liczbe zapisanych osób na dane warsztaty - zarówno klientów indywidualnych jak i tych zarejestrowanych przez firmę. Również pokazuje on limity uczestników danych warsztatów.

```
CREATE VIEW
 `workshop_limits` AS
   SELECT
      `w`.`participants_limit` AS `participants_limit`,
      `c`.`name` AS `name`,
      ((SELECT
            SUM(`cwb`.`attenders_count`)
         FROM
            (`companies_workshop_bookings` `cwb`
            JOIN `companies_bookings` `cb` ON ((`cb`.`id` = `cwb`.`booking_id`)))
         WHERE
            ((`cwb`.`workshop_id` = `w`.`id`)
               AND (NOT (`cb`.`is_canceled`)))
         GROUP BY `cwb`.`workshop_id`) + (SELECT
            COUNT(`wp`.`id`)
         FROM
            (`workshop_people` `wp`
            JOIN `people_bookings` `pb` ON ((`pb`.`id` = `wp`.`booking_id`)))
         WHERE
            ((`wp`.`workshop_id` = `w`.`id`)
               AND (NOT (`pb`.`is_canceled`))))) AS `sum`
   FROM
      ((`workshops` `w`
      JOIN `conference_days` `cd` ON ((`cd`.`id` = `w`.`conference_day_id`)))
      JOIN `conferences` `c` ON ((`c`.`id` = `cd`.`conference_id`)))
   ORDER BY `w`.`participants_limit` DESC
```

- ***conferences_limits***

Widok zawiera zsumowane limity ze wszystkich dni konferencji oraz całkowitą liczbę uczestników.

```
CREATE
  ALGORITHM = UNDEFINED
  DEFINER = `root`@`localhost`
  SQL SECURITY DEFINER
VIEW `conferences_limit` AS
  SELECT
    `c`.`name` AS `name`,
    SUM(`cd`.`participants_limit`) AS `total_participant_limit`,
    SUM(((SELECT
         COUNT(`pb`.`id`)
      FROM
         `people_bookings` `pb`
      WHERE
         ((`pb`.`conference_day_id` = `cd`.`id`)
           AND (NOT (`pb`.`is_canceled`)))) + (SELECT
         (SUM(`cb`.`students_count`) + SUM(`cb`.`non_students_count`))
      FROM
         `companies_bookings` `cb`
      WHERE
         (`cb`.`conference_day_id` = `cd`.`id`)
      GROUP BY `cb`.`conference_day_id`))) AS `total_participants`
  FROM
    (`conference_days` `cd`
    JOIN `conferences` `c` ON ((`cd`.`conference_id` = `c`.`id`)))
  GROUP BY `cd`.`participants_limit` , ((SELECT
      COUNT(`pb`.`id`)
    FROM
      `people_bookings` `pb`
    WHERE
      ((`pb`.`conference_day_id` = `cd`.`id`)
         AND (NOT (`pb`.`is_canceled`)))) + (SELECT
      (SUM(`cb`.`students_count`) + SUM(`cb`.`non_students_count`))
    FROM
      `companies_bookings` `cb`
    WHERE
      (`cb`.`conference_day_id` = `cd`.`id`)
    GROUP BY `cb`.`conference_day_id`))
```

5. Triggery

- *companies_bookings_BEFORE_INSERT*

Trigger zapobiegający przekroczeniu limitu uczestników danego dnia konferencji przy wykonywaniu operacji insert na tabeli conpanies_bookings. Wyliczana jest obecna liczba uczestników (łącznie klientów indywidualnych oraz zarejestrowanych przez firmę) i porównywana jest z limitem osób. Jeżeli operacja insert nie powiedzie się otrzymamy komunikat ‘Conference day participants limit exceeded!’

```
CREATE TRIGGER `conferences`.`companies_bookings_BEFORE_INSERT`
BEFORE INSERT ON `conferences`.`companies_bookings`
FOR EACH ROW
BEGIN
	DECLARE COMPANY_BOOKINGS INT;
   DECLARE PEOPLE_BOOKINGS INT;
   DECLARE PARTICIPANTS_LIMIT INT;

	SET COMPANY_BOOKINGS = (SELECT
SUM(cb.students_count)+SUM(cb.non_students_count)
									FROM
companies_bookings AS cb
		     WHERE cb.conference_day_id = NEW.conference_day_id
		     GROUP BY cb.conference_day_id);

	SET PEOPLE_BOOKINGS = (SELECT COUNT(pb.id)
								FROM people_bookings AS pb
		    WHERE pb.conference_day_id = NEW.conference_day_id AND NOT
pb.is_canceled);

	SET PARTICIPANTS_LIMIT = (SELECT cd.participants_limit
									FROM conference_days
AS cd
									WHERE cd.id =
NEW.conference_day_id);

	IF PARTICIPANTS_LIMIT IS NOT NULL AND PARTICIPANTS_LIMIT <=
COMPANY_BOOKINGS + PEOPLE_BOOKINGS THEN
		signal sqlstate '45000' set message_text = 'Conference day participants limit
exceeded!';
	END IF;
END$$
```

- *companies_workshop_bookings_AFTER_INSERT*

Trigger aktualizujący liczbę uczestników danych warsztatów po wykonaniu operacji insert na tabeli companies_workshop_bookings.

```
CREATE TRIGGER `conferences`.`companies_workshop_bookings_AFTER_INSERT`
AFTER INSERT ON `conferences`.`companies_workshop_bookings`
FOR EACH ROW
BEGIN
	DECLARE WORKSHOP_PRICE INT;
	SET WORKSHOP_PRICE = (SELECT w.price FROM workshops as w WHERE w.id
= NEW.workshop_id);
	UPDATE companies_bookings SET total = total + WORKSHOP_PRICE *
NEW.attenders_count WHERE id = NEW.booking_id;
END$$
```

- *companies_workshop_bookings_BEFORE_INSERT*

Trigger sprawdzający przed wykonaniem operacji insert na tabeli companies_workshop_bookings czy nie został przekroczony limit zapisanych na warsztaty. Analogicznie do dnia konferencji wyliczana jest liczba uczestników którzy są klientami indywidualnymi oraz tych zapisanych przez firmę po czym zostaję ona porównana z limitem. Jeżeli trigger zadziała otrzymamy komunikat 'Workshop participants limit exceeded!'

```
CREATE TRIGGER `conferences`.`companies_workshop_bookings_BEFORE_INSERT`
BEFORE INSERT ON `conferences`.`companies_workshop_bookings`
FOR EACH ROW
BEGIN
   DECLARE COMPANY_BOOKINGS INT;
   DECLARE PEOPLE_BOOKINGS INT;
   DECLARE PARTICIPANTS_LIMIT INT;

   SET COMPANY_BOOKINGS = (SELECT SUM(cwb.attenders_count)
                                                            FROM
companies_workshop_bookings AS cwb
                  JOIN companies_bookings AS cb ON cb.id = cwb.booking_id
                  WHERE cwb.workshop_id = NEW.workshop_id AND NOT cb.is_canceled
                  GROUP BY cwb.workshop_id);
	SET PEOPLE_BOOKINGS = (SELECT COUNT(wp.id)
                                                      FROM workshop_people AS wp
                                                      JOIN people_bookings AS pb ON
pb.id = wp.booking_id
```

```
                WHERE wp.workshop_id = NEW.workshop_id AND NOT pb.is_canceled);
        SET PARTICIPANTS_LIMIT = (SELECT w.participants_limit
                                                            FROM workshops AS w
                                                            WHERE w.id =
NEW.workshop_id);
        IF PARTICIPANTS_LIMIT IS NOT NULL AND PARTICIPANTS_LIMIT <
COMPANY_BOOKINGS + PEOPLE_BOOKINGS + NEW.attenders_count THEN
                signal sqlstate '45000' set message_text = 'Workshop participants limit
exceeded!';
        END IF;
END$$
```

- *people_bookings_BEFORE_INSERT*

Trigger analogiczny, sprawdzający czy nie został przekroczony limit uczestników danego dnia konferencji tym razem przy insercie do tabeli people_bookings. Otrzymamy komunikat ‘Conference day participants limit exceeded!’

```
CREATE TRIGGER `conferences`.`people_bookings_BEFORE_INSERT`
BEFORE INSERT ON `conferences`.`people_bookings`
FOR EACH ROW
BEGIN
        DECLARE COMPANY_BOOKINGS INT;
  DECLARE PEOPLE_BOOKINGS INT;
  DECLARE PARTICIPANTS_LIMIT INT;

        SET COMPANY_BOOKINGS = (SELECT
SUM(cb.students_count)+SUM(cb.non_students_count)
                                                           FROM
companies_bookings AS cb
                WHERE cb.conference_day_id = NEW.conference_day_id
                GROUP BY cb.id);

        SET PEOPLE_BOOKINGS = (SELECT COUNT(pb.id)
                                                    FROM people_bookings AS pb
                WHERE pb.conference_day_id = NEW.conference_day_id AND NOT
pb.is_canceled);

        SET PARTICIPANTS_LIMIT = (SELECT cd.participants_limit
                                                            FROM conference_days
AS cd
                                                            WHERE cd.id =
NEW.conference_day_id);
```

```
		IF PARTICIPANTS_LIMIT IS NOT NULL AND PARTICIPANTS_LIMIT <=
COMPANY_BOOKINGS + PEOPLE_BOOKINGS THEN
				signal sqlstate '45000' set message_text = 'Conference day participants limit
exceeded!';
		END IF;
END$$
```

- ***price_levels_BEFORE_INSERT***

Trigger sprawdzający czy przy wykonywaniu operacji insert do tabeli price_levels przedziały nie nachodzą na siebie. Otrzymamy komunikat 'Price level interval overlaps existing one!'

```
CREATE TRIGGER `conferences`.`price_levels_BEFORE_INSERT`
BEFORE INSERT ON `conferences`.`price_levels`
FOR EACH ROW
BEGIN
		IF EXISTS(SELECT pl.id FROM price_levels AS pl
						WHERE pl.start_date < NEW.end_date AND pl.end_date >
NEW.start_date AND pl.conference_id = NEW.conference_id)
		THEN
				signal sqlstate '45000' set message_text = 'Price level interval overlaps
existing one!';
   END IF;
END$$
```

- ***workshop_people_AFTER_DELETE***

Trigger aktualizujący cene (price) w tabeli people_bookings po wykonaniu operacji delete na tabeli people_bookings.

```
CREATE TRIGGER `conferences`.`workshop_people_AFTER_DELETE`
AFTER DELETE ON `conferences`.`workshop_people`
FOR EACH ROW
BEGIN
		DECLARE WORKSHOP_PRICE INT;
		SET WORKSHOP_PRICE = (SELECT w.price FROM workshops as w WHERE w.id
= OLD.workshop_id);
		UPDATE people_bookings SET price = price - WORKSHOP_PRICE WHERE id =
OLD.booking_id;
END$$
```

- ***workshop_people_AFTER_INSERT***

Trigger analogiczny, aktualizujący cene (price) w tabeli people_bookings po wykonaniu operacji insert na tabeli people_bookings.

```
TRIGGER `conferences`.`workshop_people_AFTER_INSERT`
AFTER INSERT ON `conferences`.`workshop_people`
FOR EACH ROW
BEGIN
		DECLARE WORKSHOP_PRICE INT;
		SET WORKSHOP_PRICE = (SELECT w.price FROM workshops as w WHERE w.id
= NEW.workshop_id);
		UPDATE people_bookings SET price = price + WORKSHOP_PRICE WHERE id =
NEW.booking_id;
END$$
```

- ***workshop_people_BEFORE_INSERT***

Trigger przed wykonaniem operacji insert na workshop_people pilnujący:

- czy osoba która jest zapisana na warsztat jest również zapisana na konferencję w tym samym dniu, komunikat 'Person is not attending workshop conference day!'
- czy nie został przekroczony limit uczestników po operacji insert, komunikat 'Workshop participants limit exceeded!'
- czy osoba nie uczestniczy w innym warsztacie w tym samym czasie, komunikat 'Person is attending different workshop at the same time!’

```
CREATE TRIGGER `conferences`.`workshop_people_BEFORE_INSERT`
BEFORE INSERT ON `conferences`.`workshop_people`
FOR EACH ROW
BEGIN
  DECLARE WORKSHOP_START TIME;
  DECLARE WORKSHOP_END TIME;
  DECLARE COMPANY_BOOKINGS INT;
  DECLARE PEOPLE_BOOKINGS INT;
  DECLARE PARTICIPANTS_LIMIT INT;

		IF NOT EXISTS(SELECT w.id FROM workshops AS w
							JOIN people_bookings AS pb ON pb.conference_day_id
= w.conference_day_id
							WHERE w.id = NEW.workshop_id AND pb.id =
NEW.booking_id)
```

```
            THEN
                        signal sqlstate '45000' set message_text = 'Person is not attending workshop
conference day!';
   END IF;

   SET COMPANY_BOOKINGS = (SELECT SUM(cwb.attenders_count)
                                                                                FROM
companies_workshop_bookings AS cwb
                    JOIN companies_bookings AS cb ON cb.id = cwb.booking_id
                    WHERE cwb.workshop_id = NEW.workshop_id AND NOT cb.is_canceled
                    GROUP BY cwb.workshop_id);
            SET PEOPLE_BOOKINGS = (SELECT COUNT(wp.id)
                                                                        FROM workshop_people AS wp
                                                                        JOIN people_bookings AS pb ON
pb.id = wp.booking_id
                    WHERE wp.workshop_id = NEW.workshop_id AND NOT pb.is_canceled);
            SET PARTICIPANTS_LIMIT = (SELECT w.participants_limit
                                                                                FROM workshops AS w
                                                                                WHERE w.id =
NEW.workshop_id);

            IF PARTICIPANTS_LIMIT IS NOT NULL AND PARTICIPANTS_LIMIT <=
COMPANY_BOOKINGS + PEOPLE_BOOKINGS THEN
                        signal sqlstate '45000' set message_text = 'Workshop participants limit
exceeded!';
            END IF;

   SET WORKSHOP_START = (SELECT w.start FROM workshops as w WHERE w.id =
NEW.workshop_id);
   SET WORKSHOP_END = (SELECT w.end FROM workshops as w WHERE w.id =
NEW.workshop_id);

   IF EXISTS(SELECT w.id FROM workshops AS w
                                    JOIN workshop_people AS wp ON w.id = wp.workshop_id
                                    WHERE wp.person_id = NEW.person_id AND w.start <
WORKSHOP_END AND w.end > WORKSHOP_START)
            THEN
                        signal sqlstate '45000' set message_text = 'Person is attending different
workshop at the same time!';
            END IF;
END$$
```

- *company_attenders_BEFORE_INSERT*

Trigger sprawdzający poprawność emaila oraz numeru telefonu za pomocą wyrażeń regularnych

```
CREATE DEFINER = CURRENT_USER TRIGGER
`conferences`.`people_BEFORE_INSERT` BEFORE INSERT ON `people` FOR EACH ROW
BEGIN
IF email NOT LIKE '[a-zA-Z0-9_\-]+@([a-zA-Z0-9_\-]+\.)+(com|org|edu|nz|au)' THEN
                        signal sqlstate '45000' set message_text = 'Wrong email pattern';
            END IF;
   IF phone NOT LIKE '[+](\d[-\s]?){6,11}\d' THEN
                        signal sqlstate '45000' set message_text = 'Wrong phone pattern';
            END IF;
END
```

6. Procedury

- *add_company_booking_workshop*

Procedura służąca do dodawania rezerwacji firmy na dany warsztat.

```
CREATE PROCEDURE `add_company_booking_workshop` (IN workshop_id INT(11), IN
booking_id INT(11), IN attenders_count INT(11))
BEGIN
            declare time_now DATETIME;
   set time_now = NOW();
            insert into companies_workshop_bookings (booking_id, workshop_id,
attenders_count, created) values (booking_id, workshop_id, attenders_count, time_now);
END
```

- *add_person_booking_workshop*

Procedura służąca do dodawania rezerwacji klienta indywidualnego na warsztat

```
CREATE PROCEDURE `add_person_booking` (IN person_id INT(11), IN workshop_id
INT(11),

            IN booking_id INT(11))
BEGIN
            insert into workshop_people (person_id, workshop_id, booking_id) values (person_id,
workshop_id, booking_id);
END
```

- ***add_company_booking_payin***

Procedura do dodawania wpłat dokonanych przez firmę.

```
CREATE PROCEDURE `add_company_booking_payin` (IN amount DECIMAL(10,0),
IN booking_id INT(11))

BEGIN
		declare time_now DATETIME;
   set time_now = NOW();
		insert into company_booking_payins (amount, datetime, booking_id) values
(amount, time_now, booking_id);
   update companies_bookings
   set total = total + amount
   where companies_bookings.id = booking_id;

END
```

- ***add_person_booking_payin***

Procedura do dodawania wpłat dokonanych przez klientów indywidualnych

```
CREATE PROCEDURE `add_person_booking_payin` (IN amount DECIMAL(10,0),
IN datetime DATETIME,

		IN booking_id INT(11))
BEGIN
		insert into people_booking_payins (amount, datetime, booking_id) values
(amount, datetime, booking_id);
END
```

- ***cancel_conference***

Procedura służąca do anulowania konferencji.

```
CREATE PROCEDURE `cancel_conference` (IN id INT(11))
BEGIN
		update conference_days
   set is_canceled = 1
   where conference_id = id;
END
```

- ***add_conference***

```
CREATE PROCEDURE `add_conference` (IN name varchar(255), IN start_date
DATETIME, IN end_date DATETIME)
BEGIN
            insert into conferences (name, start_date, end_date)
   values (name, start_date, end_date);
END
```

- ***add_company_attender***

```
CREATE PROCEDURE `add_company_attender` (IN first_name varchar(120), IN
last_name varchar(120),
IN company_id INT(11), IN email varchar(255), IN password varchar(255), IN
student_card_number varchar(10))
BEGIN
            insert into company_attenders (first_name, last_name, company_id, email,
password, student_card_number)
   values (first_name, last_name, company_id, email, password, student_card_number);
END
```

- ***add_conference_day***

```
CREATE PROCEDURE `add_conference_day` (IN conference_id INT, IN day DATE,
IN participants_limit INT)
BEGIN
            insert into conference_days (conference_id, day, participants_limit)
   values (conference_id, day, participants_limit);
END
```

- ***add_price_level***

```
CREATE PROCEDURE `add_price_level` (IN price DECIMAL, IN start_date DATE,
IN end_date DATE, IN conference_id INT, IN student_discount DECIMAL)
BEGIN
            insert into price_levels (price, start_date, end_date, conference_id,
student_discount)
   values (price, start_date, end_date, conference_id, student_discount);
END
```

- ***add_person***

```
CREATE PROCEDURE `add_person` (IN first_name varchar(120), IN last_name
varchar(120),
IN pesel varchar(11), IN city varchar(120), IN country varchar(2), IN email
varchar(255),
IN password varchar(255), IN phone varchar(20), IN student_card_number
varchar(10))
BEGIN
            insert into people (first_name, last_name, pesel, city, country, email, password,
phone, student_card_number)
   values (first_name, last_name, pesel, city, country, email, password, phone,
student_card_number);
END
```

- *add_workshop*

```
CREATE PROCEDURE `add_person` (IN first_name varchar(120), IN last_name
varchar(120),
IN pesel varchar(11), IN city varchar(120), IN country varchar(2), IN email
varchar(255),
IN password varchar(255), IN phone varchar(20), IN student_card_number
varchar(10))
BEGIN
            insert into people (first_name, last_name, pesel, city, country, email, password,
phone, student_card_number)
   values (first_name, last_name, pesel, city, country, email, password, phone,
student_card_number);
END
```

- *enroll_person_for_a_conference_day*

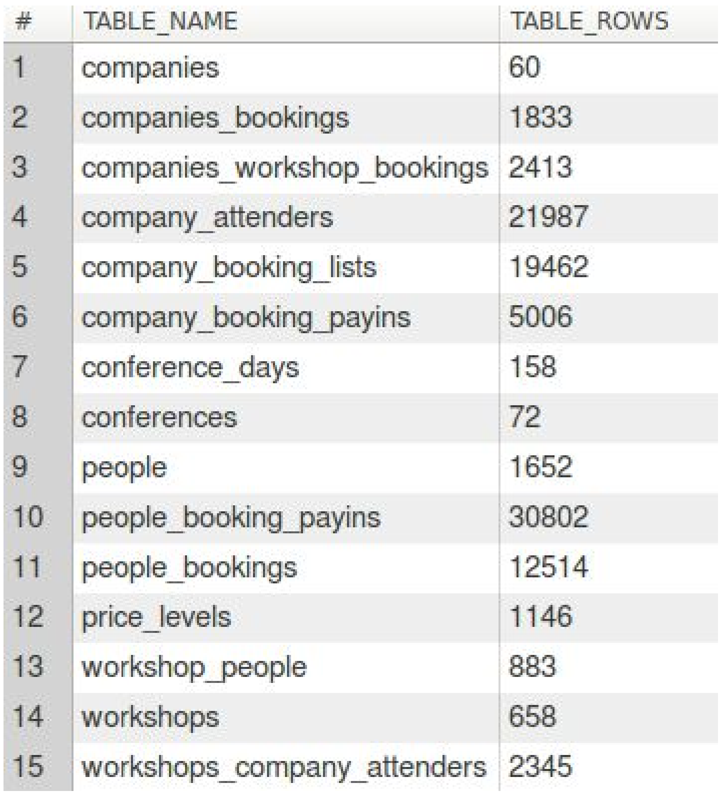
```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`enroll_person_for_conference_day`(
            IN conference_day_id INT,
   IN person_id INT,
   IN enrollment_date DATE
)
BEGIN

   DECLARE PRICE DECIMAL;
   DECLARE STUDENT_DISCOUNT INT;
   DECLARE STUDENT_CARD_NUMBER VARCHAR(10);

   SET STUDENT_CARD_NUMBER = (SELECT p.student_card_number FROM
people AS p WHERE p.id = person_id);
   SET PRICE = (SELECT pl.price FROM price_levels AS pl
             JOIN conferences AS c ON c.id = pl.conference_id
             JOIN conference_days AS cd ON cd.conference_id = c.id
             WHERE enrollment_date >= pl.start_date
             AND enrollment_date <= pl.end_date
             AND cd.id = conference_day_id);
   SET STUDENT_DISCOUNT = (SELECT pl.student_discount FROM price_levels AS
pl
                    JOIN conferences AS c ON c.id = pl.conference_id
                    JOIN conference_days AS cd ON cd.conference_id = c.id
                    WHERE enrollment_date >= pl.start_date
                    AND enrollment_date <= pl.end_date
                    AND cd.id = conference_day_id);

            IF STUDENT_CARD_NUMBER IS NOT NULL THEN
                        SET PRICE = PRICE * (1 - STUDENT_DISCOUNT/100);
            END IF;

            INSERT INTO people_bookings (
                        person_id,
                        conference_day_id,
                        price,
                        is_canceled,
                        due_date,
                        created
            ) VALUES (
                        person_id,
                        conference_day_id,
                        PRICE,
                        FALSE,
                        DATE_ADD(NOW(), INTERVAL 1 WEEK),
                        NOW()
            );
END
```

7. Generator danych

Generator danych został zaimplementowany w języku Python, z wykorzystaniem framework'a Django do zmapowania tabel na obiekty oraz biblioteki Faker

8. Dane statystyczne

```
SELECT TABLE_NAME, TABLE_ROWS
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'conferences'
```



#	TABLE_NAME	TABLE_ROWS
1	companies	60
2	companies_bookings	1833
3	companies_workshop_bookings	2413
4	company_attenders	21987
5	company_booking_lists	19462
6	company_booking_payins	5006
7	conference_days	158
8	conferences	72
9	people	1652
10	people_booking_payins	30802
11	people_bookings	12514
12	price_levels	1146
13	workshop_people	883
14	workshops	658
15	workshops_company_attenders	2345

9. Uprawnienia

- administrator (superuser) - wszystko
- klient indywidualny - rejestracja w systemie, booking dnia konferencji, booking warsztatu, wpłata
- klient jako firma - rejestracja w systemie, booking dnia konferencji, uzupełnianie listy uczestników, zapis uczestników firmy na warsztaty
- organizator konferencji - CRUD konferencji, CRUD listy uzytkowników, akceptacja bookingów, CRUD dnia konferencji
- uczestnik - może sprawdzać na jakie konferencje/warsztaty jest zapisany