System bazodanowy dla zespołu regionalnego ZPiT Mietniowiacy[[1]](#footnote-1) (do wypożyczania strojów)

Autorzy KKK: Kaczor Filip, Kowalski Jakub, Kogut Jakub

Spis Treści

[Spis pojęć 7](#_Toc187234340)

[Opis projektu 7](#_Toc187234341)

[Wymagania Klient 7](#_Toc187234342)

[Wymagania Strój 7](#_Toc187234343)

[Wymagania User 8](#_Toc187234344)

[Wymagania Kostiumolog 8](#_Toc187234345)

[Wymagania Członka Chóru 8](#_Toc187234346)

[Wymagania Członka Kapeli 8](#_Toc187234347)

[Wymagania Członka Baletu 8](#_Toc187234348)

[Schemat bazy danych 9](#_Toc187234349)

[Tabele 10](#_Toc187234350)

[Countries 10](#_Toc187234351)

[Regions 11](#_Toc187234352)

[Settlements 12](#_Toc187234353)

[Locations 13](#_Toc187234354)

[Genders 14](#_Toc187234355)

[Users 15](#_Toc187234356)

[Roles 18](#_Toc187234357)

[Types\_of\_voices 19](#_Toc187234358)

[Types\_of\_instruments 20](#_Toc187234359)

[Dances 21](#_Toc187234360)

[Costumiers 22](#_Toc187234361)

[Singers 24](#_Toc187234362)

[Singer\_voices 26](#_Toc187234363)

[Musicians 28](#_Toc187234364)

[Musician\_instrument 30](#_Toc187234365)

[Dancers 32](#_Toc187234366)

[Dancer\_dance 34](#_Toc187234367)

[Colors 35](#_Toc187234368)

[Collections 36](#_Toc187234369)

[Costumes\_items 37](#_Toc187234370)

[Patterns 39](#_Toc187234371)

[Head\_accessory\_categories 40](#_Toc187234372)

[Head\_accessories 41](#_Toc187234373)

[Aprons 43](#_Toc187234374)

[Caftans 45](#_Toc187234375)

[Petticoats 47](#_Toc187234376)

[Corsets 49](#_Toc187234377)

[Skirts 51](#_Toc187234378)

[Belts 53](#_Toc187234379)

[Shirts 55](#_Toc187234380)

[Pants 57](#_Toc187234381)

[Boots 59](#_Toc187234382)

[Neck\_accessories 60](#_Toc187234383)

[Costumes 62](#_Toc187234384)

[States\_of\_requests 67](#_Toc187234385)

[Requests 68](#_Toc187234386)

[Rental\_costume\_item\_requests 70](#_Toc187234387)

[Return\_costume\_item\_requests 72](#_Toc187234388)

[Borrow\_costume\_item\_requests 74](#_Toc187234389)

[Notifications 76](#_Toc187234390)

[Rentals 78](#_Toc187234391)

[Wyzwalacze 80](#_Toc187234392)

[prevent\_invalid\_costume\_item\_insert 80](#_Toc187234393)

[prevent\_invalid\_costume\_insert\_update 83](#_Toc187234394)

[prevent\_invalid\_request\_delete 84](#_Toc187234395)

[prevent\_invalid\_request\_insert 85](#_Toc187234396)

[prevent\_invalid\_rental\_costume\_item\_request\_insert 86](#_Toc187234397)

[prevent\_invalid\_return\_costume\_item\_request\_insert 87](#_Toc187234398)

[prevent\_invalid\_borrow\_costume\_item\_request\_insert 88](#_Toc187234399)

[prevent\_invalid\_notification\_insert 89](#_Toc187234400)

[prevent\_invalid\_rental\_insert 90](#_Toc187234401)

[Widoki 91](#_Toc187234402)

[Locations\_with\_settlements\_regions\_countries 91](#_Toc187234403)

[User\_count\_by\_settlement 92](#_Toc187234404)

[User\_function\_counts 93](#_Toc187234405)

[Detailed\_users 94](#_Toc187234406)

[Detailed\_singers 96](#_Toc187234407)

[Detailed\_musicians 97](#_Toc187234408)

[Detailed\_dancers 98](#_Toc187234409)

[Singer\_count\_by\_voice\_type 99](#_Toc187234410)

[Musician\_count\_by\_instrument\_type 100](#_Toc187234411)

[Dancer\_count\_by\_dance\_type 101](#_Toc187234412)

[Costume\_item\_count\_by\_collection\_and\_class 102](#_Toc187234413)

[Costume\_item\_count\_by\_class 106](#_Toc187234414)

[Detailed\_aprons 107](#_Toc187234415)

[Detailed\_boots 108](#_Toc187234416)

[Detailed\_petticoats 109](#_Toc187234417)

[Detailed\_skirts 110](#_Toc187234418)

[Detailed\_caftans 111](#_Toc187234419)

[Detailed\_corsets 112](#_Toc187234420)

[Detailed\_neck\_accessories 113](#_Toc187234421)

[Detailed\_head\_accessories 114](#_Toc187234422)

[Detailed\_belts 115](#_Toc187234423)

[Detailed\_pants 116](#_Toc187234424)

[Detailed\_shirts 117](#_Toc187234425)

[Costume\_with\_costume\_items\_name 118](#_Toc187234426)

[Not\_read\_notifications 120](#_Toc187234427)

[Detailed\_rental\_costume\_item\_requests 121](#_Toc187234428)

[Detailed\_return\_costume\_item\_requests 122](#_Toc187234429)

[Detailed\_borrow\_costume\_item\_requests 123](#_Toc187234430)

[Detailed\_costume\_item\_requests 124](#_Toc187234431)

[Current\_rentals\_count\_by\_costume\_item\_class 125](#_Toc187234432)

[Current\_rentals\_count\_by\_user\_function 127](#_Toc187234433)

[Detailed\_rentals 128](#_Toc187234434)

[Detailed\_current\_rentals 129](#_Toc187234435)

[Funkcje 130](#_Toc187234436)

[check\_costume\_inconsistency 130](#_Toc187234437)

[check\_rental\_inconsistency 137](#_Toc187234438)

[get\_costume\_item\_rental\_history 139](#_Toc187234439)

[check\_if\_error\_in\_costume\_item\_common\_part 140](#_Toc187234440)

[get\_user\_rental\_history 142](#_Toc187234441)

[get\_costumier\_unresolved\_requests 143](#_Toc187234442)

[get\_user\_unresolved\_borrow\_requests 145](#_Toc187234443)

[get\_user\_current\_rentals 147](#_Toc187234444)

[get\_user\_function\_percentage 148](#_Toc187234445)

[get\_fits\_aprons 149](#_Toc187234446)

[get\_fits\_boots 150](#_Toc187234447)

[get\_fits\_petticoats 151](#_Toc187234448)

[get\_fits\_skirts 153](#_Toc187234449)

[get\_fits\_caftans 154](#_Toc187234450)

[get\_fits\_corsets 156](#_Toc187234451)

[get\_fits\_neck\_accessories 158](#_Toc187234452)

[get\_fits\_head\_accessories 159](#_Toc187234453)

[get\_fits\_belts 161](#_Toc187234454)

[get\_fits\_pants 162](#_Toc187234455)

[get\_fits\_shirts 164](#_Toc187234456)

[get\_user\_unread\_notifications 166](#_Toc187234457)

[get\_user\_unclosed\_costume\_item\_requests 167](#_Toc187234458)

[Procedury 168](#_Toc187234459)

[add\_country 168](#_Toc187234460)

[add\_region 169](#_Toc187234461)

[add\_settlement 170](#_Toc187234462)

[add\_location 172](#_Toc187234463)

[add\_gender 174](#_Toc187234464)

[add\_role 175](#_Toc187234465)

[add\_type\_of\_voice 176](#_Toc187234466)

[add\_type\_of\_instrument 177](#_Toc187234467)

[add\_dance 178](#_Toc187234468)

[add\_color 179](#_Toc187234469)

[add\_collection 180](#_Toc187234470)

[add\_pattern 181](#_Toc187234471)

[add\_head\_accessory\_category 182](#_Toc187234472)

[add\_state\_of\_request 183](#_Toc187234473)

[add\_user 184](#_Toc187234474)

[make\_user\_costumier 188](#_Toc187234475)

[make\_user\_singer 190](#_Toc187234476)

[make\_user\_musician 192](#_Toc187234477)

[make\_user\_dancer 194](#_Toc187234478)

[add\_voice\_to\_singer 196](#_Toc187234479)

[add\_instrument\_to\_musician 198](#_Toc187234480)

[add\_dance\_to\_dancer 200](#_Toc187234481)

[add\_apron 202](#_Toc187234482)

[add\_head\_accessory 204](#_Toc187234483)

[add\_caftan 206](#_Toc187234484)

[add\_petticoat 208](#_Toc187234485)

[add\_corset 210](#_Toc187234486)

[add\_skirt 213](#_Toc187234487)

[add\_belt 215](#_Toc187234488)

[add\_shirt 217](#_Toc187234489)

[add\_pants 220](#_Toc187234490)

[add\_boots 222](#_Toc187234491)

[add\_neck\_accessory 224](#_Toc187234492)

[update\_apron 226](#_Toc187234493)

[update\_head\_accessory 228](#_Toc187234494)

[update\_caftan 230](#_Toc187234495)

[update\_petticoat 233](#_Toc187234496)

[update\_corset 235](#_Toc187234497)

[update\_skirt 238](#_Toc187234498)

[update\_belt 240](#_Toc187234499)

[update\_shirt 242](#_Toc187234500)

[update\_pants 245](#_Toc187234501)

[update\_boots 247](#_Toc187234502)

[update\_neck\_accessory 249](#_Toc187234503)

[add\_costume 251](#_Toc187234504)

[add\_rental\_costume\_item\_request 256](#_Toc187234505)

[add\_return\_costume\_item\_request 258](#_Toc187234506)

[add\_borrow\_costume\_item\_request 260](#_Toc187234507)

[add\_notification 262](#_Toc187234508)

[add\_rental 264](#_Toc187234509)

[update\_costume\_item\_location 266](#_Toc187234510)

[delete\_request 268](#_Toc187234511)

[accept\_rental\_costume\_item\_request 269](#_Toc187234512)

[deny\_rental\_costume\_item\_request 271](#_Toc187234513)

[accept\_return\_costume\_item\_request 273](#_Toc187234514)

[deny\_return\_costume\_item\_request 275](#_Toc187234515)

[accept\_borrow\_costume\_item\_request 277](#_Toc187234516)

[deny\_borrow\_costume\_item\_request 279](#_Toc187234517)

[mark\_notification\_as\_read 281](#_Toc187234518)

[rent\_costume\_item 282](#_Toc187234519)

[borrow\_costume\_item 284](#_Toc187234520)

[Indeksy 286](#_Toc187234521)

[I1 286](#_Toc187234522)

[Użytkownicy 287](#_Toc187234523)

[Normal\_user 287](#_Toc187234524)

[Costumier 288](#_Toc187234525)

[Manager 290](#_Toc187234526)

[Para\_admin 292](#_Toc187234527)

[Przykłady użycia 296](#_Toc187234528)

[Akceptacja request’u wypożycz 296](#_Toc187234529)

[Finalizacja wypożyczenia 296](#_Toc187234530)

[Odmowa request’u pożycz 296](#_Toc187234531)

[Użytkownik wypożycza koszulę 297](#_Toc187234532)

[Dodanie i poprawa informacji o fartuszku 297](#_Toc187234533)

[Sprawdzenie wypożyczenia elementu stroju. 297](#_Toc187234534)

# Spis pojęć

Klient – Kierownik Zespół Pieśni i Tańca Mietniowiacy.

User – Członek zespołu.

Strój – Pojedynczy element kostiumu regionalnego lub zbiór elementów stanowiący jedną całość.

Kostiumolog – Członek zespołu odpowiedzialny za utrzymanie Strój.

Wypożyczyć – Udostępnienie Strój przez Kostiumolog dla User do jego przechowywania w domu.

Oddać – Zwrócenie Strój do Kostiumolog przez User w konkretnej lokalizacji.

Pożyczyć - User swój wypożyczony Strój przekazuje innemu User. Transakcja inicjowana jest przez User, a wymagana jest jedynie zgoda drugiego User. Nie jest wymagana zgoda Kostiumolog.

# Opis projektu

Głównym celem projektu jest dostarczenie kompleksowego systemu bazodanowego umożliwiającego zarządzanie zespołem. Część podlegająca ocenie jako zaliczenie z przedmiotu Bazy Danych skupia się na części umożliwiającej User przeprowadzanie wypożyczenia Strój. Przedstawicielem Klient od spraw ww. projektu jest członek zespołu – Jakub Kowalski.

## Wymagania Klient

* Wypożyczyć i Oddać Strój odbywa się wyłącznie za zgodą Kostiumolog.
* Możliwe jest pożyczenie Strój innemu User. Odbywa się to bez zgody Kostiumolog, natomiast wymagana jest zgoda od User obecnie posiadającego Strój, do wykonania takiej transakcji.
* User, Kostiumolog zostają poinformowani o zgłoszeniach o Wypożyczyć, Oddać lub Pożyczyć.
* User może pełnić funkcje: tancerz, śpiewaka, muzyka, kostiumologa.

### Wymagania Strój

* Strój posiada informacje o swoich właściwościach fizycznych (przekazanych przez Przedstawiciela).
* Dodatkowo Strój powinien posiadać informacje o jego obecnej lokalizacji.
* Jeśli został Wypożyczyć to lokalizacja to adres User, jeśli Oddać to miejsce ośrodka do którego został zwrócony Strój.

### Wymagania User

* User może zobaczyć Strój, swoje Strój, Strój który pasuje wymiarowo.
* User może Wypożyczyć Strój lub go Oddać.
* User może zainicjować Pożyczyć Strój.
* User może zobaczyć stan Wypożyczyć, Oddać lub Pożyczyć.
* User może anulować Wypożyczyć, Oddać lub Pożyczyć Strój.
* User może wyrazić zgodę na Pożyczyć lub jej nie wyrazić.
* User może zobaczyć i oznaczyć jako przeczytaną informację o decyzji związanej z Wypożyczyć, Oddać i Pożyczyć Strój.
* User może zobaczyć historię swoich Wypożyczyć, Oddać i Pożyczyć Strój.

### Wymagania Kostiumolog

* Kostiumolog dziedziczy po User.
* Posiada dodatkowe informacje o lokalizacji miejsca pracy i pełniącej roli.
* Kostiumolog może zobaczyć Strój należący do innego User
* Kostiumolog może dodawać i edytować informacje o Strój.
* Kostiumolog może zobaczyć prośby o Wypożyczyć i Oddać.
* Kostiumolog może wyrazić zgodę na Wypożyczyć lub Oddać Strój lub jej nie wyrazić.
* Kostiumolog może zobaczyć historię Wypożyczyć, Oddać i Pożyczyć Strój innych User.

### Wymagania Członka Chóru

* Członek chóru dziedziczy po User
* Posiada informację o pełniącej roli oraz jakimi głosami potrafi śpiewać.

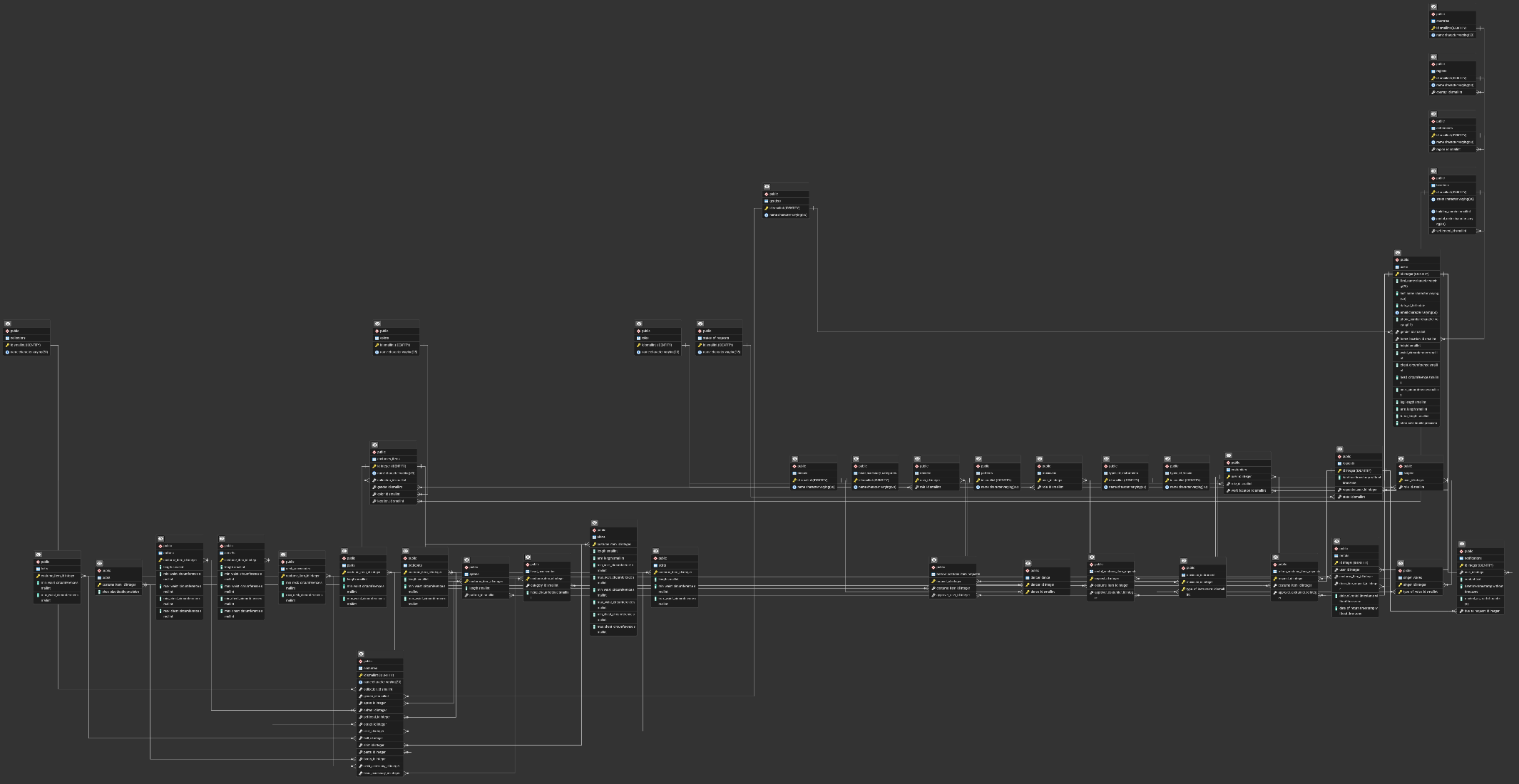
### Wymagania Członka Kapeli

* Członek chóru dziedziczy po User
* Posiada informację o pełniącej roli oraz na jakich instrumentach potrafi grać.

### Wymagania Członka Baletu

* Członek chóru dziedziczy po User
* Posiada informację o pełniącej roli oraz jakie tańce potrafi tańczyć.

# Schemat bazy danych



# Tabele

## Countries

**Opis:**

* Tabela zawiera informacje o krajach.

**Warunki integralności:**

* Nazwa danego kraju może wystąpić tylko raz.

**Implementacja:**

CREATE TABLE Countries

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR(30) NOT NULL

)

;

ALTER TABLE Countries

ADD CONSTRAINT Countries\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Countries

ADD CONSTRAINT Countries\_UN UNIQUE ( name ) ;

## Regions

**Opis:**

* Tabela zawiera informacje o regionach danego kraju.
* Kraj – klucz obcy do tabeli Countries.

**Warunki integralności:**

* Nazwa danego regionu może wystąpić tylko raz dla danego kraju.

**Implementacja**:

CREATE TABLE Regions

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (30) NOT NULL ,

country\_id SMALLINT NOT NULL

)

;

ALTER TABLE Regions

ADD CONSTRAINT Regions\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Regions

ADD CONSTRAINT Regions\_UN UNIQUE ( name , country\_id ) ;

ALTER TABLE Regions

ADD CONSTRAINT Regions\_Countries\_FK FOREIGN KEY

(

country\_id

)

REFERENCES Countries

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Settlements

**Opis:**

* Tabela zawiera informacje o miejscowościach danego regionu.
* Region – klucz obcy do tabeli Regions

**Warunki integralności:**

* Nazwa danej miejscowości może wystąpić tylko raz dla danego regionu.

**Implementacja:**

CREATE TABLE Settlements

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL,

name VARCHAR(30) NOT NULL ,

region\_id SMALLINT NOT NULL

)

;

ALTER TABLE Settlements

ADD CONSTRAINT Settlements\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Settlements

ADD CONSTRAINT Settlements\_UN UNIQUE ( name , region\_id ) ;

ALTER TABLE Settlements

ADD CONSTRAINT Settlements\_Regions\_FK FOREIGN KEY

(

region\_id

)

REFERENCES Regions

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Locations

**Opis:**

* Tabela zawiera informacje o konkretnym miejscu (lokacji).
* Lokacja składa się z ulicy, numeru budynki, kodu pocztowego i konkretnego miasta/wsi.
* Miasto/wieś – klucz obcy do tabeli Settlements

**Warunki integralności:**

* Dana lokacja (ulica, numer budynku, kod pocztowy, id miasta/wsi) musi być unikatowa.

**Implementacja:**

CREATE TABLE Locations

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

street VARCHAR (30) NOT NULL ,

building\_number SMALLINT NOT NULL ,

postal\_code VARCHAR (10) NOT NULL ,

settlement\_id SMALLINT NOT NULL

)

;

ALTER TABLE Locations

ADD CONSTRAINT Locations\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Locations

ADD CONSTRAINT Locations\_UN UNIQUE ( street , building\_number , postal\_code , settlement\_id ) ;

ALTER TABLE Locations

ADD CONSTRAINT Locations\_Settlements\_FK FOREIGN KEY

(

settlement\_id

)

REFERENCES Settlements

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Genders

**Opis:**

* Tabela zawiera informacje o płciach.

**Warunki integralności:**

* Dana płeć może wystąpić tylko raz.
* Id:
  + 1: male
  + 2: female
  + 3: bigender

**Implementacja:**

CREATE TABLE Genders

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (25) NOT NULL

)

;

ALTER TABLE Genders

ADD CONSTRAINT Genders\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Genders

ADD CONSTRAINT Genders\_UN UNIQUE ( name ) ;

INSERT INTO Genders (name) VALUES ('male'), ('female'), ('bigender');

## Users

**Opis:**

* Tabela zawiera ogólne informacje o członkach zespołu (użytkownikach).
* Informacje przechowywane to: pierwsze imię, nazwisko, data urodzenia, adres email, numer telefonu (kierunkowy), płeć, lokalizacje zamieszkania, wysokość, obwód w pasie, obwód w klatce piersiowej, obwód głowy, obwód szyi, długość nogi (od pasa do kostki), długość ręki (od ramienia do nadgarstku), długość torsu (od szyi do pasa) i rozmiar buta (miara EU). Wszystkie miary są podawane w cm.
* Płeć – klucz obcy do tabeli Genders.
* Lokalizacja zamieszkania – klucz obcy do tabeli Locations.

**Warunki integralności:**

* Użytkownik musi posiadać unikatowy adres email.
* Email musi posiadać odpowiednią formę.
* Numer telefonu musi posiadać odpowiednią formę.
* Parametry fizyczne (np. wzrost) nie mogą być ujemne lub 0.

**Implementacja:**

CREATE TABLE Users

(

id INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL ,

first\_name VARCHAR (25) NOT NULL ,

last\_name VARCHAR (30) NOT NULL ,

date\_of\_birth DATE NOT NULL ,

email VARCHAR (50) NOT NULL ,

phone\_number VARCHAR (12) NOT NULL ,

gender\_id SMALLINT NOT NULL ,

home\_location\_id SMALLINT NOT NULL ,

height SMALLINT NOT NULL ,

waist\_circumference SMALLINT NOT NULL ,

chest\_circumference SMALLINT NOT NULL ,

head\_circumference SMALLINT NOT NULL ,

neck\_circumference SMALLINT NOT NULL ,

leg\_length SMALLINT NOT NULL ,

arm\_length SMALLINT NOT NULL ,

torso\_length SMALLINT NOT NULL ,

shoe\_size FLOAT NOT NULL

)

;

ALTER TABLE Users

ADD CONSTRAINT Users\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Users

ADD CONSTRAINT Users\_UN UNIQUE ( email ) ;

ALTER TABLE Users

ADD CONSTRAINT chk\_email\_format CHECK (email ~\* '^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$');

ALTER TABLE Users

ADD CONSTRAINT chk\_phone\_format CHECK (phone\_number ~\* '^\+\d{2}\d{9}$');

ALTER TABLE Users

ADD CONSTRAINT chk\_height\_value CHECK (height > 0);

ALTER TABLE Users

ADD CONSTRAINT chk\_waist\_circumference\_value CHECK (waist\_circumference > 0);

ALTER TABLE Users

ADD CONSTRAINT chk\_chest\_circumference\_value CHECK (chest\_circumference > 0);

ALTER TABLE Users

ADD CONSTRAINT chk\_head\_circumference\_value CHECK (head\_circumference > 0);

ALTER TABLE Users

ADD CONSTRAINT chk\_neck\_circumference\_value CHECK (neck\_circumference > 0);

ALTER TABLE Users

ADD CONSTRAINT chk\_leg\_length\_value CHECK (leg\_length > 0 AND leg\_length < height);

ALTER TABLE Users

ADD CONSTRAINT chk\_arm\_length\_value CHECK (arm\_length > 0 AND arm\_length < height);

ALTER TABLE Users

ADD CONSTRAINT chk\_torso\_length\_value CHECK (torso\_length > 0 AND torso\_length < height);

ALTER TABLE Users

ADD CONSTRAINT chk\_shoe\_size\_value CHECK (shoe\_size > 0);

ALTER TABLE Users

ADD CONSTRAINT Users\_Genders\_FK FOREIGN KEY

(

gender\_id

)

REFERENCES Genders

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Users

ADD CONSTRAINT Users\_Locations\_FK FOREIGN KEY

(

home\_location\_id

)

REFERENCES Locations

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Roles

**Opis:**

* Tabela zawiera informacje o roli danego członka zespołu.

**Warunki integralności:**

* Dana rola może wystąpić tylko raz.

**Implementacja:**

CREATE TABLE Roles

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (20) NOT NULL

)

;

ALTER TABLE Roles

ADD CONSTRAINT Roles\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Roles

ADD CONSTRAINT Roles\_UN UNIQUE ( name ) ;

## Types\_of\_voices

**Opis:**

* Tabela zawiera informacje o rodzajach głosów w śpiewie.

**Warunki integralności:**

* Dany rodzaj głosu może wystąpić tylko raz.

**Implementacja:**

CREATE TABLE Types\_of\_voices

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (10) NOT NULL

)

;

ALTER TABLE Types\_of\_voices

ADD CONSTRAINT Types\_of\_voices\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Types\_of\_voices

ADD CONSTRAINT Types\_of\_voices\_UN UNIQUE ( name ) ;

## Types\_of\_instruments

**Opis:**

* Tabela zawiera informacje o rodzajach instumentów muzycznych.

**Warunki integralności:**

* Dany rodzaj instrumentu może wystąpić tylko raz.

**Implementacja:**

CREATE TABLE Types\_of\_instruments

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (20) NOT NULL

)

;

ALTER TABLE Types\_of\_instruments

ADD CONSTRAINT Types\_of\_instruments\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Types\_of\_instruments

ADD CONSTRAINT Types\_of\_instruments\_UN UNIQUE ( name ) ;

## Dances

**Opis:**

* Tabela zawiera informacje o tańcach.

**Warunki integralności:**

* Dany taniec może wystąpić tylko raz.

**Implementacja:**

CREATE TABLE Dances

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (20) NOT NULL

)

;

ALTER TABLE Dances

ADD CONSTRAINT Dances\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Dances

ADD CONSTRAINT Dances\_UN UNIQUE ( name ) ;

## Costumiers

**Opis:**

* Tabela rozszerza informacje o członku zespołu.
* Zawiera dodatkowe informacje specyficzne dla członka będącego kostiumologiem: rolę jaką posiada na tym stanowisku i miejsce (lokację) w której pracuje.
* User\_id – klucz obcy do tabeli Users.
* Rola – klucz obcy do tabeli Roles.
* Lokację pracy – klucz obcy do tabeli Locations.

**Warunki integralności:**

* Dany członek może posiadać tylko jeden zestaw dodatkowych atrybutów odpowiadających kostiumologowi.
* W przypadku usunięcia członka zespołu, dodatkowe dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Costumiers

(

user\_id INTEGER NOT NULL ,

role\_id SMALLINT NOT NULL ,

work\_location\_id SMALLINT NOT NULL

)

;

ALTER TABLE Costumiers

ADD CONSTRAINT Costumiers\_PK PRIMARY KEY ( user\_id ) ;

ALTER TABLE Costumiers

ADD CONSTRAINT Costumiers\_Locations\_FK FOREIGN KEY

(

work\_location\_id

)

REFERENCES Locations

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumiers

ADD CONSTRAINT Costumiers\_Roles\_FK FOREIGN KEY

(

role\_id

)

REFERENCES Roles

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumiers

ADD CONSTRAINT Costumiers\_Users\_FK FOREIGN KEY

(

user\_id

)

REFERENCES Users

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Singers

**Opis:**

* Tabela rozszerza informacje o członku zespołu.
* Zawiera dodatkowe informacje specyficzne dla członka będącego członkiem chóru: rolę jaką posiada na tym stanowisku.
* User\_id – klucz obcy do tabeli Users.
* Rola – klucz obcy do tabeli Roles.

**Warunki integralności:**

* Dany członek może posiadać tylko jeden zestaw dodatkowych atrybutów odpowiadających członkowi chóru.
* W przypadku usunięcia członka zespołu, dodatkowe dane z tej tabeli też powinny zostać usunięte.

Implementacja:

CREATE TABLE Singers

(

user\_id INTEGER NOT NULL ,

role\_id SMALLINT NOT NULL

)

;

ALTER TABLE Singers

ADD CONSTRAINT Singers\_PK PRIMARY KEY ( user\_id ) ;

ALTER TABLE Singers

ADD CONSTRAINT Singers\_Roles\_FK FOREIGN KEY

(

role\_id

)

REFERENCES Roles

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Singers

ADD CONSTRAINT Singers\_Users\_FK FOREIGN KEY

(

user\_id

)

REFERENCES Users

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Singer\_voices

**Opis:**

* Tabela zawiera informacje o tym jakim głosem umie śpiewać dany śpiewak.
* Singer\_id – klucz obcy do tabeli Singers.
* Głos – klucz obcy do tabeli Types\_of\_voices.

**Warunki integralności:**

* Członek chóru może umieć śpiewać tylko raz tym samym głosem.
* W przypadku usunięcia rekordu z tabeli Singers, dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Singer\_voices

(

singer\_id INTEGER NOT NULL ,

type\_of\_voice\_id SMALLINT NOT NULL

)

;

ALTER TABLE Singer\_voices

ADD CONSTRAINT Singer\_voices\_PK PRIMARY KEY ( singer\_id, type\_of\_voice\_id ) ;

ALTER TABLE Singer\_voices

ADD CONSTRAINT Singer\_voices\_Singers\_FK FOREIGN KEY

(

singer\_id

)

REFERENCES Singers

(

user\_id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Singer\_voices

ADD CONSTRAINT Singer\_voices\_Types\_of\_voices\_FK FOREIGN KEY

(

type\_of\_voice\_id

)

REFERENCES Types\_of\_voices

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Musicians

**Opis:**

* Tabela rozszerza informacje o członku zespołu.
* Zawiera dodatkowe informacje specyficzne dla członka będącego członkiem kapeli: rolę jaką posiada na tym stanowisku.
* User\_id – klucz obcy do tabeli Users.
* Rola – klucz obcy do tabeli Roles.

**Warunki integralności:**

* Dany członek może posiadać tylko jeden zestaw dodatkowych atrybutów odpowiadających członkowi kapeli.
* W przypadku usunięcia członka zespołu, dodatkowe dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Musicians

(

user\_id INTEGER NOT NULL ,

role\_id SMALLINT NOT NULL

)

;

ALTER TABLE Musicians

ADD CONSTRAINT Musicians\_PK PRIMARY KEY ( user\_id ) ;

ALTER TABLE Musicians

ADD CONSTRAINT Musicians\_Roles\_FK FOREIGN KEY

(

role\_id

)

REFERENCES Roles

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Musicians

ADD CONSTRAINT Musicians\_Users\_FK FOREIGN KEY

(

user\_id

)

REFERENCES Users

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Musician\_instrument

**Opis:**

* Tabela zawiera informacje o tym na jakim instrumencie umie grać dany członek kapeli.
* Musician\_id – klucz obcy do tabeli Musicians.
* Instrument – klucz obcy do tabeli Types\_of\_instruments.

**Warunki integralności:**

* Członek kapeli może umieć grać tylko raz tym samym instrumentem.
* W przypadku usunięcia rekordu z tabli Musicians, dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Musician\_instrument

(

musician\_id INTEGER NOT NULL ,

type\_of\_instrument\_id SMALLINT NOT NULL

)

;

ALTER TABLE Musician\_instrument

ADD CONSTRAINT Musician\_instrument\_PK PRIMARY KEY ( musician\_id, type\_of\_instrument\_id ) ;

ALTER TABLE Musician\_instrument

ADD CONSTRAINT Musician\_instrument\_Musicians\_FK FOREIGN KEY

(

musician\_id

)

REFERENCES Musicians

(

user\_id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Musician\_instrument

ADD CONSTRAINT Musician\_instrument\_Types\_of\_instruments\_FK FOREIGN KEY

(

type\_of\_instrument\_id

)

REFERENCES Types\_of\_instruments

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Dancers

**Opis:**

* Tabela rozszerza informacje o członku zespołu.
* Zawiera dodatkowe informacje specyficzne dla członka będącego członkiem baletu: rolę jaką posiada na tym stanowisku.
* User\_id – klucz obcy do tabeli Users.
* Rola – klucz obcy do tabeli Roles.

**Warunki integralności:**

* Dany członek może posiadać tylko jeden zestaw dodatkowych atrybutów odpowiadających członkowi baletu.
* W przypadku usunięcia członka zespołu, dodatkowe dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Dancers

(

user\_id INTEGER NOT NULL ,

role\_id SMALLINT NOT NULL

)

;

ALTER TABLE Dancers

ADD CONSTRAINT Dancers\_PK PRIMARY KEY ( user\_id ) ;

ALTER TABLE Dancers

ADD CONSTRAINT Dancers\_Roles\_FK FOREIGN KEY

(

role\_id

)

REFERENCES Roles

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Dancers

ADD CONSTRAINT Dancers\_Users\_FK FOREIGN KEY

(

user\_id

)

REFERENCES Users

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Dancer\_dance

**Opis:**

* Tabela zawiera informacje o tym na jakie tańce umie tancerz.
* Dancer\_id – klucz obcy do tabeli Dancers.
* Taniec – klucz obcy do tabeli Types\_of\_dances.

**Warunki integralności:**

* Tancerz może umieć tańczyć tylko raz ten sam taniec.
* W przypadku usunięcia rekordu z tabeli Dancers, dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Dancer\_dance

(

dancer\_id INTEGER NOT NULL ,

dance\_id SMALLINT NOT NULL

)

;

ALTER TABLE Dancer\_dance

ADD CONSTRAINT Dancer\_dance\_PK PRIMARY KEY ( dancer\_id, dance\_id ) ;

ALTER TABLE Dancer\_dance

ADD CONSTRAINT Dancer\_dance\_Dancers\_FK FOREIGN KEY

(

dancer\_id

)

REFERENCES Dancers

(

user\_id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Dancer\_dance

ADD CONSTRAINT Dancer\_dance\_Dances\_FK FOREIGN KEY

(

Dance\_id

)

REFERENCES Dances

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Colors

**Opis:**

* Tabela zawiera informacje o kolorach.

**Warunki integralności:**

* Dany kolor może wystąpić tylko raz.

**Implementacja:**

CREATE TABLE Colors

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (25) NOT NULL

)

;

ALTER TABLE Colors

ADD CONSTRAINT Colors\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Colors

ADD CONSTRAINT Colors\_UN UNIQUE ( name ) ;

## Collections

**Opis:**

* Tabela zawiera informacje o rodzajach kolekcji, do której mogą należeć dane elementu strojów lub stroje.

**Warunki integralności:**

* Dany rodzaj może wystąpić tylko raz.
* id
  + 1: Universal

**Implementacja:**

CREATE TABLE Collections

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (20) NOT NULL

)

;

ALTER TABLE Collections

ADD CONSTRAINT Collections\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Collections

ADD CONSTRAINT Collections\_UN UNIQUE ( name ) ;

INSERT INTO collections (name) VALUES ('universal');

## Costumes\_items

**Opis:**

* Tabela zawiera informacje element stroju, jego nazwę, kolekcję do jakiej należy, płeć dla jakiej element jest przeznaczony, dominujący kolor i w jakiej lokacji się znajduje.
* Kolekcja – klucz obcy do tabeli Collections.
* Płeć – klucz obcy do tabeli Genders.
* Kolor – klucz obcy do tabeli Colors.
* Lokacja – klucz obcy do tabeli Locations.

**Warunki integralności:**

* Nazwa elementu musi być unikatowa.
* Gender\_id
  + 1: male
  + 2: female
  + 3: bigender

**Implementacja:**

CREATE TABLE Costumes\_items

(

id INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (30) NOT NULL ,

collection\_id SMALLINT NOT NULL ,

gender\_id SMALLINT NOT NULL ,

color\_id SMALLINT NOT NULL ,

location\_id SMALLINT NOT NULL

)

;

ALTER TABLE Costumes\_items

ADD CONSTRAINT Costumes\_items\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Costumes\_items

ADD CONSTRAINT Costumes\_items\_UN UNIQUE ( name ) ;

ALTER TABLE Costumes\_items ADD CONSTRAINT chk\_gender\_id\_value CHECK (gender\_id in (1, 2, 3));

ALTER TABLE Costumes\_items

ADD CONSTRAINT Costumes\_items\_Collections\_FK FOREIGN KEY

(

collection\_id

)

REFERENCES Collections

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes\_items

ADD CONSTRAINT Costumes\_items\_Colors\_FK FOREIGN KEY

(

color\_id

)

REFERENCES Colors

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes\_items

ADD CONSTRAINT Costumes\_items\_Genders\_FK FOREIGN KEY

(

gender\_id

)

REFERENCES Genders

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes\_items

ADD CONSTRAINT Costumes\_items\_Locations\_FK FOREIGN KEY

(

location\_id

)

REFERENCES Locations

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Patterns

**Opis:**

* Tabela zawiera informacje o wzorach.

**Warunki integralności:**

* Dany wzór może być tylko jeden raz.

**Implementacja:**

CREATE TABLE Patterns

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (20) NOT NULL

)

;

ALTER TABLE Patterns

ADD CONSTRAINT Patterns\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Patterns

ADD CONSTRAINT Patterns\_UN UNIQUE ( name ) ;

## Head\_accessory\_categories

**Opis:**

* Tabela zawiera informacje o rodzajach akcesoriów głowy.

**Warunki integralności:**

* Dany rodzaj akcesoria występuje tylko raz

**Implementacja:**

CREATE TABLE Head\_accessory\_categories

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (20) NOT NULL

)

;

ALTER TABLE Head\_accessory\_categories

ADD CONSTRAINT Head\_accessory\_categories\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Head\_accessory\_categories

ADD CONSTRAINT Head\_accessory\_categories\_UN UNIQUE ( name ) ;

## Head\_accessories

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla akcesoria głowy: kategorię akcesoria i ewentualnie rekomendowany obwód głowy (wartość NULL oznacza, że obwód głowy nie ma znaczenia).
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.
* Kategoria – klucz obcy do tabeli Head\_accessory\_categories.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* head\_circumference może przyjąć wartości > 0 lub NULL

**Implementacja:**

CREATE TABLE Head\_accessories

(

costume\_item\_id INTEGER NOT NULL ,

category\_id SMALLINT NOT NULL ,

head\_circumference SMALLINT

)

;

ALTER TABLE Head\_accessories

ADD CONSTRAINT Head\_accessories\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Head\_accessories

ADD CONSTRAINT chk\_head\_circumference\_value CHECK (head\_circumference > 0 OR head\_circumference IS NULL);

ALTER TABLE Head\_accessories

ADD CONSTRAINT Head\_accessories\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Head\_accessories

ADD CONSTRAINT Head\_accessories\_Head\_accessory\_categories\_FK FOREIGN KEY

(

category\_id

)

REFERENCES Head\_accessory\_categories

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Aprons

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla fartuszka: wzór jaki jest na nim i jego długość.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.
* Wzór – klucz obcy do tabeli Patterns.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0.

**Implementacja:**

CREATE TABLE Aprons

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

pattern\_id SMALLINT NOT NULL

)

;

ALTER TABLE Aprons

ADD CONSTRAINT Aprons\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Aprons

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Aprons

ADD CONSTRAINT Aprons\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Aprons

ADD CONSTRAINT Aprons\_Patterns\_FK FOREIGN KEY

(

pattern\_id

)

REFERENCES Patterns

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Caftans

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla kaftanów: jego długość, rekomendowany zakres obwodu w pasie i klatce piersiowe.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Caftans

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL ,

min\_chest\_circumference SMALLINT NOT NULL ,

max\_chest\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Caftans

ADD CONSTRAINT Caftans\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Caftans

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Caftans

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Caftans

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Caftans

ADD CONSTRAINT chk\_min\_chest\_circumference\_value CHECK (min\_chest\_circumference > 0);

ALTER TABLE Caftans

ADD CONSTRAINT chk\_max\_chest\_circumference\_value CHECK (max\_chest\_circumference > 0);

ALTER TABLE Caftans

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Caftans

ADD CONSTRAINT chk\_min\_max\_chest\_circumference\_value CHECK (min\_chest\_circumference <= max\_chest\_circumference);

ALTER TABLE Caftans

ADD CONSTRAINT Caftans\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Petticoats

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla halek: jej długość i rekomendowany zakres obwodu w pasie.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0, wartość max ma być większ lub równa wartości min.

**Implementacja:**

CREATE TABLE Petticoats

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Petticoats

ADD CONSTRAINT Petticoats\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Petticoats

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Petticoats

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Petticoats

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Petticoats

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Petticoats

ADD CONSTRAINT Petticoats\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Corsets

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla gorsetu: jego długość, rekomendowany obwód w pasie i klatce piersiowej.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Corsets

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL ,

min\_chest\_circumference SMALLINT NOT NULL ,

max\_chest\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Corsets

ADD CONSTRAINT Corsets\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Corsets

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Corsets

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Corsets

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Corsets

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Corsets

ADD CONSTRAINT chk\_min\_chest\_circumference\_value CHECK (min\_chest\_circumference > 0);

ALTER TABLE Corsets

ADD CONSTRAINT chk\_max\_chest\_circumference\_value CHECK (max\_chest\_circumference > 0);

ALTER TABLE Corsets

ADD CONSTRAINT chk\_min\_max\_chest\_circumference\_value CHECK (min\_chest\_circumference <= max\_chest\_circumference);

ALTER TABLE Corsets

ADD CONSTRAINT Corsets\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Skirts

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla spódnicy: jej długość i rekomendowany zakres obwodu w pasie.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Skirts

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Skirts

ADD CONSTRAINT Skirts\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Skirts

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Skirts

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Skirts

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Skirts

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Skirts

ADD CONSTRAINT Skirts\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Belts

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla pasa: rekomendowany zakres obwodu w pasie.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. obwód) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Belts

(

costume\_item\_id INTEGER NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Belts

ADD CONSTRAINT Belts\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Belts

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Belts

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Belts

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Belts

ADD CONSTRAINT Belts\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Shirts

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla koszuli: jej długość, długość rękawa oraz rekomendowane obwody pasa, szyi i klatki piersiowej.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Shirts

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

arm\_length SMALLINT NOT NULL ,

min\_neck\_circumference SMALLINT NOT NULL ,

max\_neck\_circumference SMALLINT NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL ,

min\_chest\_circumference SMALLINT NOT NULL ,

max\_chest\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Shirts

ADD CONSTRAINT Shirts\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Shirts

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_arm\_length\_value CHECK (arm\_length > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_min\_chest\_circumference\_value CHECK (min\_chest\_circumference > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_max\_chest\_circumference\_value CHECK (max\_chest\_circumference > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_min\_max\_chest\_circumference\_value CHECK (min\_chest\_circumference <= max\_chest\_circumference);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_min\_neck\_circumference\_value CHECK (min\_neck\_circumference > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_max\_neck\_circumference\_value CHECK (max\_neck\_circumference > 0);

ALTER TABLE Shirts

ADD CONSTRAINT chk\_min\_max\_neck\_circumference\_value CHECK (min\_neck\_circumference <= max\_neck\_circumference);

ALTER TABLE Shirts

ADD CONSTRAINT Shirts\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Pants

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla spodni: ich długość i rekomendowany zakres obwodu w pasie.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. długość) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Pants

(

costume\_item\_id INTEGER NOT NULL ,

length SMALLINT NOT NULL ,

min\_waist\_circumference SMALLINT NOT NULL ,

max\_waist\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Pants

ADD CONSTRAINT Pants\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Pants

ADD CONSTRAINT chk\_length\_value CHECK (length > 0);

ALTER TABLE Pants

ADD CONSTRAINT chk\_min\_waist\_circumference\_value CHECK (min\_waist\_circumference > 0);

ALTER TABLE Pants

ADD CONSTRAINT chk\_max\_waist\_circumference\_value CHECK (max\_waist\_circumference > 0);

ALTER TABLE Pants

ADD CONSTRAINT chk\_min\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

ALTER TABLE Pants

ADD CONSTRAINT Pants\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Boots

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla butów: rozmiar.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Numer buta musi być większy od 0.

**Implementacja:**

CREATE TABLE Boots

(

costume\_item\_id INTEGER NOT NULL ,

shoe\_size FLOAT NOT NULL

)

;

ALTER TABLE Boots

ADD CONSTRAINT Boots\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Boots

ADD CONSTRAINT chk\_shoe\_size\_value CHECK (shoe\_size > 0);

ALTER TABLE Boots

ADD CONSTRAINT Boots\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Neck\_accessories

**Opis:**

* Tabela rozszerzająca informacje o elemencie stroju.
* Zawiera dodatkowe informacje specyficzne dla ozdób na szyję: rekomendowany zakres obwodu szyi.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.

**Warunki integralności:**

* Element stroju nie może posiadać kilku rozszerzających informacji należących do różnych klas – TRIGGER - prevent\_invalid\_costume\_item\_insert.
* Element stroju może posiadać tylko jedno rozszerzenie informacji należących danej klasy.
* W przypadku usunięcia rekordu z tabeli Costume\_items, dane z tej tabeli też powinny zostać usunięte.
* Parametry fizyczne (np. obwód) powinny być większe od 0, wartość max ma być większa lub równa wartości min.

**Implementacja:**

CREATE TABLE Neck\_accessories

(

costume\_item\_id INTEGER NOT NULL ,

min\_neck\_circumference SMALLINT NOT NULL ,

max\_neck\_circumference SMALLINT NOT NULL

)

;

ALTER TABLE Neck\_accessories

ADD CONSTRAINT Neck\_accessories\_PK PRIMARY KEY ( costume\_item\_id ) ;

ALTER TABLE Neck\_accessories

ADD CONSTRAINT chk\_min\_neck\_circumference\_value CHECK (min\_neck\_circumference > 0);

ALTER TABLE Neck\_accessories

ADD CONSTRAINT chk\_max\_neck\_circumference\_value CHECK (max\_neck\_circumference > 0);

ALTER TABLE Neck\_accessories

ADD CONSTRAINT chk\_min\_max\_neck\_circumference\_value CHECK (min\_neck\_circumference <= max\_neck\_circumference);

ALTER TABLE Neck\_accessories

ADD CONSTRAINT Neck\_accessories\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Costumes

**Opis:**

* Zawiera zestawienie elementów stroju tworzący kompletny strój.
* Kilka strojów może zawierać ten sam element.

**Warunki integralności:**

* Nazwa stroju musi być unikalna.
* Gender\_id
  + 1: male
  + 2: female
  + 3: bigender
* Strój musi być zgodny pod względem rekomendowanej płci i kolekcji, do której należą elementy.
* Kolekcja universal (id = 1) może wystąpić w zestawieniu.
* Płeć bigender (id = 3) może wystąpić w zestawieniu – TRIGGER prevent\_invalid\_costume\_insert\_update

**Implementacja:**

CREATE TABLE Costumes

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (30) NOT NULL ,

collection\_id SMALLINT NOT NULL ,

gender\_id SMALLINT NOT NULL ,

apron\_id INTEGER ,

caftan\_id INTEGER ,

petticoat\_id INTEGER ,

corset\_id INTEGER ,

skirt\_id INTEGER ,

belt\_id INTEGER ,

shirt\_id INTEGER ,

pants\_id INTEGER ,

boots\_id INTEGER ,

neck\_accessory\_id INTEGER ,

head\_accessory\_id INTEGER

)

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_UN UNIQUE ( name ) ;

ALTER TABLE Costumes ADD CONSTRAINT chk\_gender\_id\_value CHECK (gender\_id in (1, 2, 3));

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Aprons\_FK FOREIGN KEY

(

apron\_id

)

REFERENCES Aprons

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Belts\_FK FOREIGN KEY

(

belt\_id

)

REFERENCES Belts

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Boots\_FK FOREIGN KEY

(

boots\_id

)

REFERENCES Boots

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Caftans\_FK FOREIGN KEY

(

caftan\_id

)

REFERENCES Caftans

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Collections\_FK FOREIGN KEY

(

collection\_id

)

REFERENCES Collections

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Corsets\_FK FOREIGN KEY

(

corset\_id

)

REFERENCES Corsets

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Genders\_FK FOREIGN KEY

(

gender\_id

)

REFERENCES Genders

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Head\_accessories\_FK FOREIGN KEY

(

head\_accessory\_id

)

REFERENCES Head\_accessories

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Neck\_accessories\_FK FOREIGN KEY

(

neck\_accessory\_id

)

REFERENCES Neck\_accessories

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Pants\_FK FOREIGN KEY

(

pants\_id

)

REFERENCES Pants

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Petticoats\_FK FOREIGN KEY

(

petticoat\_id

)

REFERENCES Petticoats

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Shirts\_FK FOREIGN KEY

(

shirt\_id

)

REFERENCES Shirts

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Costumes

ADD CONSTRAINT Costumes\_Skirts\_FK FOREIGN KEY

(

skirt\_id

)

REFERENCES Skirts

(

costume\_item\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## States\_of\_requests

**Opis:**

* Tabela zawiera informacje o stanie żądania.

**Warunki integralności:**

* Dany stan może być tylko raz.
* Id statusów
  + 1: pending
  + 2: accept
  + 3: deny

**Implementacja:**

CREATE TABLE States\_of\_requests

(

id SMALLINT GENERATED ALWAYS AS IDENTITY NOT NULL ,

name VARCHAR (15) NOT NULL

)

;

ALTER TABLE States\_of\_requests

ADD CONSTRAINT States\_of\_requests\_PK PRIMARY KEY ( id ) ;

ALTER TABLE States\_of\_requests

ADD CONSTRAINT States\_of\_requests\_UN UNIQUE ( name ) ;

INSERT INTO States\_of\_requests (name) VALUES ('PENDING'), ('ACCEPT'), ('DENY');

## Requests

**Opis:**

* Tabela zawiera informacje o requestach: kto go złożył (user), kiedy, i jaki jest jego stan.
* Kto – klucz obcy do tabeli Users.
* Stan – klucz obcy do tabeli States\_of\_requests.

**Warunki integralności:**

* Nie można usunąć request jeśli zapadła decyzja (state różny od PENDING) – TRIGGER prevent\_invalid\_request\_delete

**Implementacja:**

CREATE TABLE Requests

(

id INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL ,

datetime TIMESTAMP NOT NULL ,

requester\_user\_id INTEGER NOT NULL ,

state\_id SMALLINT NOT NULL

)

;

ALTER TABLE Requests

ADD CONSTRAINT Requests\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Requests

ADD CONSTRAINT Requests\_States\_of\_requests\_FK FOREIGN KEY

(

state\_id

)

REFERENCES States\_of\_requests

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Requests

ADD CONSTRAINT Requests\_Users\_FK FOREIGN KEY

(

requester\_user\_id

)

REFERENCES Users

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Rental\_costume\_item\_requests

**Opis:**

* Tabela rozszerza informacjie o request (z tablicy Requests): jaki element stroju będzie wypożyczony i jaki Kostiumolog dane żądzanie obsłużył.
* NULL w approver\_costumier\_id oznacza to, że dowolny Kostiumolog może zaakceptować request.
* approver\_costumier\_id - klucz obcy do tabeli Costumiers.
* element stroju - klucz obcy do tabeli Costume\_elements.

**Warunki integralności:**

* Request nie może posiadać kilku rozszerzających informacji należących do różnych typów requestu – TRIGGER - prevent\_invalid\_request\_insert.
* Request można stworzyć jedynie, jeśli element stroju jest nie wypożyczony – TRIGGER prevent\_invalid\_rental\_costume\_item\_request\_insert
* Jeśli został usunięty request to dane z tej tabeli też powinny zostać usunięte.

**Implementacja:**

CREATE TABLE Rental\_costume\_item\_requests

(

request\_id INTEGER NOT NULL ,

costume\_item\_id INTEGER NOT NULL ,

approver\_costumier\_id INTEGER

)

;

ALTER TABLE Rental\_costume\_item\_requests

ADD CONSTRAINT Rental\_costume\_item\_requests\_PK PRIMARY KEY ( request\_id ) ;

ALTER TABLE Rental\_costume\_item\_requests

ADD CONSTRAINT Rental\_costume\_item\_requests\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Rental\_costume\_item\_requests

ADD CONSTRAINT Rental\_costume\_item\_requests\_Costumiers\_FK FOREIGN KEY

(

approver\_costumier\_id

)

REFERENCES Costumiers

(

user\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Rental\_costume\_item\_requests

ADD CONSTRAINT Rental\_costume\_item\_requests\_Requests\_FK FOREIGN KEY

(

request\_id

)

REFERENCES Requests

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Return\_costume\_item\_requests

**Opis:**

* Tabela rozszerza informacje o request (z tablicy Requests): jaki element stroju będzie oddany i jaki Kostiumolog dane żądanie obsłużył.
* NULL w approver\_costumier\_id oznacza to, że dowolny Kostiumolog może zaakceptować request.
* approver\_costumier\_id - klucz obcy do tabeli Costumiers.
* Element stroju - klucz obcy do tabeli Costume\_elements.

**Warunki integralności:**

* Request nie może posiadać kilku rozszerzających informacji należących do różnych typów requestu – TRIGGER - prevent\_invalid\_request\_insert.
* Request można stworzyć tylko jak się posiada element stroju – TRIGGER prevent\_invalid\_return\_costume\_item\_request\_insert.
* Jeśli został usunięty request to dane z tej tabeli też powinny zostać usunięte.

**Implementacja:** CREATE TABLE Return\_costume\_item\_requests

(

request\_id INTEGER NOT NULL ,

costume\_item\_id INTEGER NOT NULL ,

approver\_costumier\_id INTEGER

)

;

ALTER TABLE Return\_costume\_item\_requests

ADD CONSTRAINT Return\_costume\_item\_requests\_PK PRIMARY KEY ( request\_id ) ;

ALTER TABLE Return\_costume\_item\_requests

ADD CONSTRAINT Return\_costume\_item\_requests\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Return\_costume\_item\_requests

ADD CONSTRAINT Return\_costume\_item\_requests\_Costumiers\_FK FOREIGN KEY

(

approver\_costumier\_id

)

REFERENCES Costumiers

(

user\_id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Return\_costume\_item\_requests

ADD CONSTRAINT Return\_costume\_item\_requests\_Requests\_FK FOREIGN KEY

(

request\_id

)

REFERENCES Requests

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

## Borrow\_costume\_item\_requests

**Opis:**

* Tabela rozszerza informacjie o request (z tablicy Requests): jaki element stroju będzie pożyczony i jaki członek zespołu dane żądzanie ma zaakceptować.
* approver\_user\_id - klucz obcy do tabeli Users.
* element stroju - klucz obcy do tabeli Costume\_elements.

**Warunki integralności:**

* Request nie może posiadać kilku rozszerzających informacji należących do różnych typów requestu – TRIGGER - prevent\_invalid\_request\_insert.
* Jeśli został usunięty request to dane z tej tabeli też powinny zostać usunięte.
* approver\_user\_id musi posiadać strój przed jego pożyczeniem, nie można pożyczyć sam sobie elementu – TRIGGER prevent\_invalid\_borrow\_costume\_item\_request\_insert

**Implementacja:**

CREATE TABLE Borrow\_costume\_item\_requests

(

request\_id INTEGER NOT NULL ,

costume\_item\_id INTEGER NOT NULL ,

approver\_user\_id INTEGER NOT NULL

)

;

ALTER TABLE Borrow\_costume\_item\_requests

ADD CONSTRAINT Borrow\_costume\_item\_requests\_PK PRIMARY KEY ( request\_id ) ;

ALTER TABLE Borrow\_costume\_item\_requests

ADD CONSTRAINT Borrow\_costume\_item\_requests\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Borrow\_costume\_item\_requests

ADD CONSTRAINT Borrow\_costume\_item\_requests\_Requests\_FK FOREIGN KEY

(

request\_id

)

REFERENCES Requests

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Borrow\_costume\_item\_requests

ADD CONSTRAINT Borrow\_costume\_item\_requests\_Users\_FK FOREIGN KEY

(

approver\_user\_id

)

REFERENCES Users

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Notifications

**Opis:**

* Tabela zawiera informacje o powiadomieniach dostępnych dla użytkownika: do kogo jest skierowane powiadomieni, treść, czas powstania, czy został już odczytany oraz czy jest związany z jakim reqestem.
* User\_id - klucz obcy do tabeli Users.
* Request - klucz obcy do tabeli Requests.

**Warunki integralności:**

* Jeśli request którego dotyczy wiadomość zostanie usunięty należy usunąć powiadomienia.  
  marked\_as\_read może przyjąć tylko dwie wartości.
* Jeśli request jest związany z requestem user\_id i requester z requestu powinny być takie same – TRIGGER prevent\_invalid\_notification\_insert.

**Implementacja:**

CREATE TABLE Notifications

(

id INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL ,

user\_id INTEGER NOT NULL ,

content TEXT NOT NULL ,

datetime TIMESTAMP NOT NULL ,

marked\_as\_read CHAR (1) DEFAULT 'F' NOT NULL,

due\_to\_request\_id INTEGER

)

;

ALTER TABLE Notifications

ADD CONSTRAINT Notifications\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Notifications

ADD CONSTRAINT chk\_marked\_as\_read CHECK (marked\_as\_read in ('F', 'T'));

ALTER TABLE Notifications

ADD CONSTRAINT Notifications\_Requests\_FK FOREIGN KEY

(

due\_to\_request\_id

)

REFERENCES Requests

(

id

)

ON DELETE CASCADE

ON UPDATE CASCADE

;

ALTER TABLE Notifications

ADD CONSTRAINT Notifications\_Users\_FK FOREIGN KEY

(

user\_id

)

REFERENCES Users

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

## Rentals

**Opis:**

* Tabela zawiera informacje o wypożyczeniach: kto, co, na podstawie jakiego requestu, czas wypożyczenia i oddania (jeśli został oddany).
* User\_id - klucz obcy do tabeli Users.
* Costume\_item\_id – klucz obcy do tabeli Costume\_items.
* Request - klucz obcy do tabeli Costume\_item\_rental\_requests.

**Warunki integralności:**

* Czas oddanie musi być późniejszy od wypożyczenia.
* User\_id i costume\_item\_id są zgodne z danymi w request – TRIGGER prevent\_invalid\_rental\_insert.

**Implementacja:**

CREATE TABLE Rentals

(

id INTEGER GENERATED ALWAYS AS IDENTITY NOT NULL ,

user\_id INTEGER NOT NULL ,

costume\_item\_id INTEGER NOT NULL ,

done\_due\_request\_id INTEGER NOT NULL ,

date\_of\_rental TIMESTAMP NOT NULL ,

date\_of\_return TIMESTAMP

)

;

ALTER TABLE Rentals

ADD CONSTRAINT Rentals\_PK PRIMARY KEY ( id ) ;

ALTER TABLE Rentals

ADD CONSTRAINT chk\_date\_of\_rental\_and\_return\_value CHECK (date\_of\_return IS NULL OR date\_of\_return > date\_of\_rental);

ALTER TABLE Rentals

ADD CONSTRAINT Rentals\_Requests\_FK FOREIGN KEY

(

done\_due\_request\_id

)

REFERENCES Requests

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Rentals

ADD CONSTRAINT Rentals\_Costumes\_items\_FK FOREIGN KEY

(

costume\_item\_id

)

REFERENCES Costumes\_items

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

ALTER TABLE Rentals

ADD CONSTRAINT Rentals\_Users\_FK FOREIGN KEY

(

user\_id

)

REFERENCES Users

(

id

)

ON DELETE RESTRICT

ON UPDATE CASCADE

;

# Wyzwalacze

TODO? Dodac dla update??? Chyba nie

## prevent\_invalid\_costume\_item\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenie elementu stroju, który posiadałby dodatkowe informacje zawarte w więcej niż jednej klasie.
* Element stroju nie może byś jednocześnie np. fartuszkiem i butami.

**Implementacja:**

CREATE FUNCTION check\_costume\_item\_has\_class\_extenction() RETURNS TRIGGER AS $$

BEGIN

IF NOT EXISTS (

SELECT 1

FROM ((SELECT 1 AS "is\_in" FROM Neck\_accessories WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Boots WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Pants WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Shirts WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Belts WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Skirts WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Corsets WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Petticoats WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Caftans WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Aprons WHERE costume\_item\_id = NEW.costume\_item\_id)

UNION

(SELECT 1 AS "is\_in" FROM Head\_accessories WHERE costume\_item\_id = NEW.costume\_item\_id)) t

GROUP BY

t.is\_in

HAVING

SUM(t.is\_in) > 0

) THEN

RETURN NEW;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Head\_accessories

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Aprons

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Caftans

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Petticoats

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Corsets

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Skirts

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Belts

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Shirts

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Pants

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Boots

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

CREATE TRIGGER prevent\_invalid\_costume\_item\_insert BEFORE INSERT ON Neck\_accessories

FOR EACH ROW EXECUTE FUNCTION check\_costume\_item\_has\_class\_extenction();

## prevent\_invalid\_costume\_insert\_update

**Opis:**

* Zadaniem wyzwalacza jest sprawdzenie (i poinformowanie) czy elementy wchodzące w skład stroju są zgodne pod względem płci (gender) i kolekcji (collection) z opisem całego stroju.
* Zgodna kolekcja to taka sama co kolekcja całego stroju lub uniwersalna o id=1.
* Zgodna płeć to taka sama co płeć całego stroju lub bigender o id=3.

**Implementacja:**

CREATE FUNCTION check\_costume\_consistency()

RETURNS TRIGGER AS $$

BEGIN

IF check\_costume\_inconsistency(

NEW.collection\_id, NEW.gender\_id, NEW.apron\_id, NEW.caftan\_id, NEW.petticoat\_id, NEW.corset\_id,

NEW.skirt\_id, NEW.belt\_id, NEW.shirt\_id, NEW.pants\_id, NEW.boots\_id, NEW.neck\_accessory\_id,

NEW.head\_accessory\_id) THEN

RAISE EXCEPTION 'Costume is inconsistancy';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_costume\_insert\_update BEFORE INSERT OR UPDATE ON Costumes

FOR EACH ROW EXECUTE FUNCTION check\_costume\_consistency();

## prevent\_invalid\_request\_delete

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie usunięcia zamkniętych requestów o stanie różnym od PENDING (id = 1).

**Implementacja:**

CREATE FUNCTION check\_request\_state()

RETURNS TRIGGER AS $$

BEGIN

IF OLD.state\_id = 1 THEN

RETURN OLD;

END IF;

RAISE NOTICE 'Cannot delete closed request';

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_request\_delete BEFORE DELETE ON Requests

FOR EACH ROW EXECUTE FUNCTION check\_request\_state();

## prevent\_invalid\_request\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenie requestu który posiadałby dodatkowe informacje specyficzne dla więcej niż jednego danego typy requestu. Request nie może być jednocześnie np. borrow\_costume\_item\_requests i return\_costume\_item\_requests.

**Implementacja:**

CREATE FUNCTION check\_request\_has\_type\_extenction()

RETURNS TRIGGER AS $$

BEGIN

IF NOT EXISTS (

SELECT 1

FROM ((SELECT 1 AS "is\_in" FROM Return\_costume\_item\_requests WHERE request\_id = NEW.request\_id)

UNION

(SELECT 1 AS "is\_in" FROM Rental\_costume\_item\_requests WHERE request\_id = NEW.request\_id)

UNION

(SELECT 1 AS "is\_in" FROM Borrow\_costume\_item\_requests WHERE request\_id = NEW.request\_id)) t

GROUP BY

t.is\_in

HAVING

SUM(t.is\_in) > 0

) THEN

RETURN NEW;

END IF;

RAISE NOTICE 'Request has already extenction';

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_request\_insert BEFORE INSERT ON Rental\_costume\_item\_requests

FOR EACH ROW EXECUTE FUNCTION check\_request\_has\_type\_extenction();

CREATE TRIGGER prevent\_invalid\_request\_insert BEFORE INSERT ON Return\_costume\_item\_requests

FOR EACH ROW EXECUTE FUNCTION check\_request\_has\_type\_extenction();

CREATE TRIGGER prevent\_invalid\_request\_insert BEFORE INSERT ON Borrow\_costume\_item\_requests

FOR EACH ROW EXECUTE FUNCTION check\_request\_has\_type\_extenction();

## prevent\_invalid\_rental\_costume\_item\_request\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenia requestu rental\_costume\_item dla elementu stroju, który już jest wypożyczony.

**Implementacja:**

CREATE FUNCTION check\_rental\_costume\_item\_request\_costume\_item()

RETURNS TRIGGER AS $$

BEGIN

PERFORM 1

FROM Rentals

WHERE costume\_item\_id = NEW.costume\_item\_id AND date\_of\_return IS NULL;

IF FOUND THEN

RAISE EXCEPTION 'Cannot create request to rent costume item with id % because it is already rented', NEW.costume\_item\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_rental\_costume\_item\_request\_insert BEFORE INSERT ON Rental\_costume\_item\_requests

FOR EACH ROW EXECUTE FUNCTION check\_rental\_costume\_item\_request\_costume\_item();

## prevent\_invalid\_return\_costume\_item\_request\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenia requestu return\_costume\_item, jeśli członek zespołu nie posiada danego elementu stroju.

**Implementacja:**

CREATE FUNCTION check\_return\_costume\_item\_request\_costume\_item()

RETURNS TRIGGER AS $$

DECLARE

r\_user\_id INT;

BEGIN

SELECT requester\_user\_id INTO r\_user\_id

FROM Requests

WHERE id = NEW.request\_id;

PERFORM 1

FROM Rentals

WHERE user\_id = r\_user\_id AND costume\_item\_id = NEW.costume\_item\_id AND date\_of\_return IS NULL;

IF NOT FOUND THEN

RAISE EXCEPTION 'Cannot create request to return costume item which you do not rent';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_return\_costume\_item\_request\_insert BEFORE INSERT ON Return\_costume\_item\_requests

FOR EACH ROW EXECUTE FUNCTION check\_return\_costume\_item\_request\_costume\_item();

## prevent\_invalid\_borrow\_costume\_item\_request\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenia requestu borrow\_costume\_item, który będzie skierowany do osoby, która nie posiada żądanego elementu stroju oraz uniemożliwienie pożyczenia elementu od samego siebie.

**Implementacja:**

CREATE FUNCTION check\_borrow\_costume\_item\_request\_approver\_and\_costume\_item()

RETURNS TRIGGER AS $$

DECLARE

r\_user\_id INT;

BEGIN

PERFORM 1

FROM Rentals

WHERE costume\_item\_id = NEW.costume\_item\_id AND user\_id = NEW.approver\_user\_id AND date\_of\_return IS NULL;

IF NOT FOUND THEN

RAISE EXCEPTION 'User % does not have requested costume item', NEW.approver\_user\_id;

END IF;

SELECT requester\_user\_id INTO r\_user\_id

FROM Requests

WHERE id = NEW.request\_id;

IF NEW.approver\_user\_id = r\_user\_id THEN

RAISE EXCEPTION 'You cannot borrow costume item to yourself';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_borrow\_costume\_item\_request\_insert BEFORE INSERT ON Borrow\_costume\_item\_requests

FOR EACH ROW EXECUTE FUNCTION check\_borrow\_costume\_item\_request\_approver\_and\_costume\_item();

## prevent\_invalid\_notification\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenia powiadomienia, który jest związany z requestem jeśli user z requestu i powiadomienia nie są takie same.

**Implementacja:**

CREATE FUNCTION check\_notification\_due\_to\_request()

RETURNS TRIGGER AS $$

DECLARE

r\_user\_id INT;

BEGIN

IF NEW.due\_to\_request\_id IS NOT NULL THEN

SELECT requester\_user\_id INTO r\_user\_id

FROM Requests

WHERE id = NEW.due\_to\_request\_id;

IF NEW.user\_id <> r\_user\_id THEN

RAISE EXCEPTION 'User id and requester id from request are not the same';

END IF;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_notification\_insert BEFORE INSERT ON Notifications

FOR EACH ROW EXECUTE FUNCTION check\_notification\_due\_to\_request();

## prevent\_invalid\_rental\_insert

**Opis:**

* Zadaniem wyzwalacza jest uniemożliwienie stworzenia i zaktualizowania wypożyczenia jeśli user\_id i costume\_item\_id nie są zgodne z tymi zawartymi w request zezwalającym na wypożyczenie.

**Implementacja:**

CREATE FUNCTION check\_rental\_consistency()

RETURNS TRIGGER AS $$

BEGIN

IF check\_rental\_inconsistency(NEW.user\_id, NEW.costume\_item\_id, NEW.done\_due\_request\_id) THEN

RAISE EXCEPTION 'Rental is inconsistancy';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_invalid\_rental\_insert BEFORE INSERT ON Rentals

FOR EACH ROW EXECUTE FUNCTION check\_rental\_consistency();

# Widoki

## Locations\_with\_settlements\_regions\_countries

**Opis:**

* Umożliwia zobaczenie pełnych danych lokalizacji.

**Implementacja:**

CREATE OR REPLACE VIEW Locations\_with\_settlements\_regions\_countries ( id, street, building\_number, postal\_code, settlement, region, country ) AS

SELECT l.id, l.street, l.building\_number, l.postal\_code, s.name AS "settlement", r.name AS "region", c.name AS "country"

FROM Locations l

INNER JOIN Settlements s

ON l.settlement\_id=s.id

INNER JOIN Regions r

ON s.region\_id=r.id

INNER JOIN Countries c

ON r.country\_id=c.id

;

## User\_count\_by\_settlement

**Opis:**

* Pokazuje, ile członków mieszka w danym wsi/mieście.

**Implementacja:**

CREATE OR REPLACE VIEW User\_count\_by\_settlement ( settlement, number\_of\_users ) AS

SELECT s.name AS "settlement", COUNT(\*) AS "number\_of\_users"

FROM Users u

INNER JOIN Locations l

ON u.home\_location\_id=l.id

INNER JOIN Settlements s

ON l.settlement\_id=s.id

GROUP BY

s.name

ORDER BY

s.name

ASC

;

## User\_function\_counts

**Opis:**

* Pokazuje, ile członków należy do chóru, baletu, kapeli lub jest odpowiedzialnych za stroje.

**Implementacja:**

CREATE OR REPLACE VIEW User\_function\_counts ( user\_function, number\_of\_users\_with\_this\_function ) AS

SELECT 'Costumiers' AS "function", COUNT(\*) AS "number\_of\_users\_with\_this\_function"

FROM Costumiers

UNION

SELECT 'Singers' AS "function", COUNT(\*) AS "number\_of\_users\_with\_this\_function"

FROM Singers

UNION

SELECT 'Musicians' AS "function", COUNT(\*) AS "number\_of\_users\_with\_this\_function"

FROM Musicians

UNION

SELECT 'Dancers' AS "function", COUNT(\*) AS "number\_of\_users\_with\_this\_function"

FROM Dancers

;

## Detailed\_users

**Opis:**

* Pokazuje dokładne informacje (wszystkie wraz z szczegółowymi) o członku zespołu.

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_users ( id, first\_name, last\_name, date\_of\_birth, email, phone\_number, gender, home\_address\_street, home\_address\_building\_number, home\_address\_postal\_code, home\_address\_settlement, home\_address\_region, home\_address\_country, height, waist\_circumference, chest\_circumference, head\_circumference, neck\_circumference, leg\_length, arm\_length, torso\_length, shoe\_size, singer\_role, musician\_role, dancer\_role, costumier\_role, costumier\_work\_address\_street, costumier\_work\_address\_building\_number, costumier\_work\_address\_postal\_code, costumier\_work\_address\_settlement, costumier\_work\_address\_region, costumier\_work\_address\_country ) AS

SELECT u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name AS "gender", l.street AS "home\_address\_street", l.building\_number AS "home\_address\_building\_number", l.postal\_code AS "home\_address\_postal\_code", l.settlement AS "home\_address\_settlement", l.region AS "home\_address\_region", l.country AS "home\_address\_country", u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, sr.name AS "singer\_role", mr.name AS "musician\_role", dr.name AS "dancer\_role", cr.name AS "costumier\_role", w.street AS "costumier\_work\_address\_street", w.building\_number AS "costumier\_work\_address\_building\_number", w.postal\_code AS "costumier\_work\_address\_postal\_code", w.settlement AS "costumier\_work\_address\_settlement", w.region AS "costumier\_work\_address\_region", w.country AS "costumier\_work\_address\_country"

FROM Users u

INNER JOIN Genders g

ON u.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON u.home\_location\_id=l.id

LEFT JOIN Singers s

ON u.id=s.user\_id

LEFT JOIN Roles sr

ON s.role\_id=sr.id

LEFT JOIN Musicians m

ON u.id=m.user\_id

LEFT JOIN Roles mr

ON m.role\_id=mr.id

LEFT JOIN Dancers d

ON u.id=d.user\_id

LEFT JOIN Roles dr

ON d.role\_id=dr.id

LEFT JOIN Costumiers c

ON u.id=c.user\_id

LEFT JOIN Roles cr

ON c.role\_id=cr.id

LEFT JOIN Locations\_with\_settlements\_regions\_countries w

ON c.work\_location\_id=w.id

;

## Detailed\_singers

**Opis:**

* Pokazuje dokładne informacje o członkach chóru (z informacją jakimi głosami potrafi śpiewać).

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_singers ( id, first\_name, last\_name, date\_of\_birth, email, phone\_number, gender, home\_address\_street, home\_address\_building\_number, home\_address\_postal\_code, home\_address\_settlement, home\_address\_region, home\_address\_country, height, waist\_circumference, chest\_circumference, head\_circumference, neck\_circumference, leg\_length, arm\_length, torso\_length, shoe\_size, role, voices ) AS

SELECT u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name AS "gender", l.street AS "home\_address\_street", l.building\_number AS "home\_address\_building\_number", l.postal\_code AS "home\_address\_postal\_code", l.settlement AS "home\_address\_settlement", l.region AS "home\_address\_region", l.country AS "home\_address\_country", u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, sr.name AS "role", STRING\_AGG(tov.name, ', ') AS "voices"

FROM Users u

INNER JOIN Genders g

ON u.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON u.home\_location\_id=l.id

INNER JOIN Singers s

ON u.id=s.user\_id

INNER JOIN Roles sr

ON s.role\_id=sr.id

INNER JOIN Singer\_voices sv

ON s.user\_id=sv.singer\_id

INNER JOIN Types\_of\_voices tov

ON sv.type\_of\_voice\_id=tov.id

GROUP BY

u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name, l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, sr.name

ORDER BY

u.last\_name, u.first\_name

ASC

;

## Detailed\_musicians

**Opis:**

* Pokazuje dokładne informacje o członkach kapeli (z informacją na jakim instrumencie potrafi grać).

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_musicians ( id, first\_name, last\_name, date\_of\_birth, email, phone\_number, gender, home\_address\_street, home\_address\_building\_number, home\_address\_postal\_code, home\_address\_settlement, home\_address\_region, home\_address\_country, height, waist\_circumference, chest\_circumference, head\_circumference, neck\_circumference, leg\_length, arm\_length, torso\_length, shoe\_size, role, instruments ) AS

SELECT u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name AS "gender", l.street AS "home\_address\_street", l.building\_number AS "home\_address\_building\_number", l.postal\_code AS "home\_address\_postal\_code", l.settlement AS "home\_address\_settlement", l.region AS "home\_address\_region", l.country AS "home\_address\_country", u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, mr.name AS "role", STRING\_AGG(toi.name, ', ') AS "instruments"

FROM Users u

INNER JOIN Genders g

ON u.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON u.home\_location\_id=l.id

INNER JOIN Musicians m

ON u.id=m.user\_id

INNER JOIN Roles mr

ON m.role\_id=mr.id

INNER JOIN Musician\_instrument mi

ON m.user\_id=mi.musician\_id

INNER JOIN Types\_of\_instruments toi

ON mi.type\_of\_instrument\_id=toi.id

GROUP BY

u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name, l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, mr.name

ORDER BY

u.last\_name, u.first\_name

ASC

;

## Detailed\_dancers

**Opis:**

* Pokazuje dokładne informacje o członkach baletu (z informacją jakie tańce umie tańczyć).

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_dancers ( id, first\_name, last\_name, date\_of\_birth, email, phone\_number, gender, home\_address\_street, home\_address\_building\_number, home\_address\_postal\_code, home\_address\_settlement, home\_address\_region, home\_address\_country, height, waist\_circumference, chest\_circumference, head\_circumference, neck\_circumference, leg\_length, arm\_length, torso\_length, shoe\_size, role, dances ) AS

SELECT u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name AS "gender", l.street AS "home\_address\_street", l.building\_number AS "home\_address\_building\_number", l.postal\_code AS "home\_address\_postal\_code", l.settlement AS "home\_address\_settlement", l.region AS "home\_address\_region", l.country AS "home\_address\_country", u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, dr.name AS "role", STRING\_AGG(dan.name, ', ') AS "dances"

FROM Users u

INNER JOIN Genders g

ON u.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON u.home\_location\_id=l.id

INNER JOIN Dancers d

ON u.id=d.user\_id

INNER JOIN Roles dr

ON d.role\_id=dr.id

INNER JOIN Dancer\_dance dd

ON d.user\_id=dd.dancer\_id

INNER JOIN Dances dan

ON dd.dance\_id=dan.id

GROUP BY

u.id, u.first\_name, u.last\_name, u.date\_of\_birth, u.email, u.phone\_number, g.name, l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, u.height, u.waist\_circumference, u.chest\_circumference, u.head\_circumference, u.neck\_circumference, u.leg\_length, u.arm\_length, u.torso\_length, u.shoe\_size, dr.name

ORDER BY

u.last\_name, u.first\_name

ASC

;

## Singer\_count\_by\_voice\_type

**Opis:**

* Pokazuje, ile członków chóru umie śpiewać danym głosem.

**Implementacja:**

CREATE OR REPLACE VIEW Singer\_count\_by\_voice\_type ( type\_of\_voice, number\_of\_singers ) AS

SELECT tov.name AS "type\_of\_voice", COUNT(\*) AS "number\_of\_singers"

FROM Singer\_voices sv

INNER JOIN Types\_of\_voices tov

ON sv.type\_of\_voice\_id=tov.id

GROUP BY

tov.name

ORDER BY

tov.name

ASC

;

## Musician\_count\_by\_instrument\_type

**Opis:**

* Pokazuje, ile członków kapeli umie grać na danym instrumencie.

**Implementacja:**

CREATE OR REPLACE VIEW Musician\_count\_by\_instrument\_type ( type\_of\_instrument, number\_of\_musicians ) AS

SELECT toi.name AS "type\_of\_instrument", COUNT(\*) AS "number\_of\_musicians"

FROM Musician\_instrument mi

INNER JOIN Types\_of\_instruments toi

ON mi.type\_of\_instrument\_id=toi.id

GROUP BY

toi.name

ORDER BY

toi.name

ASC

;

## Dancer\_count\_by\_dance\_type

**Opis:**

* Pokazuje, ile członków baletu umie tańczyć dane tańce.

**Implementacja:**

CREATE OR REPLACE VIEW Dancer\_count\_by\_dance\_type ( type\_of\_dance, number\_of\_dancers ) AS

SELECT d.name AS "type\_of\_dance", COUNT(\*) AS "number\_of\_dancers"

FROM Dancer\_dance dd

INNER JOIN Dances d

ON dd.dance\_id=d.id

GROUP BY

d.name

ORDER BY

d.name

ASC

;

## Costume\_item\_count\_by\_collection\_and\_class

**Opis:**

* Pokazuje, ile elementów stroju (z podziałem na klasę [fartuszki, buty, ...]) należy do danej kolekcji.

**Implementacja:**

CREATE OR REPLACE VIEW Costume\_item\_count\_by\_collection\_and\_class ( costume\_item\_class, collection, number\_of\_items) AS

(SELECT 'apron' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Aprons type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'caftan' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Caftans type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'petticoat' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Petticoats type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'corset' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Corsets type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'skirt' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Skirts type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'belt' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Belts type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'shirt' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Shirts type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'pants' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Pants type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'boots' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Boots type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'neck\_accessory' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Neck\_accessories type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

UNION

(SELECT 'head\_accessory' AS "costume\_item\_class", c.name AS "collection", COUNT(\*) AS "number\_of\_items"

FROM Head\_accessories type

INNER JOIN Costumes\_items ci

ON type.costume\_item\_id=ci.id

INNER JOIN Collections c

ON ci.collection\_id=c.id

GROUP BY

c.name

ORDER BY

c.name

ASC)

;

## Costume\_item\_count\_by\_class

**Opis:**

* Pokazuje, ile elementów należy do danej klasy (fartuszki, buty ...)

**Implementacja:**

CREATE OR REPLACE VIEW Costume\_item\_count\_by\_class ( costume\_item\_class, number\_of\_items) AS

(SELECT 'apron' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Aprons)

UNION

(SELECT 'caftan' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Caftans)

UNION

(SELECT 'petticoat' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Petticoats)

UNION

(SELECT 'corset' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Corsets)

UNION

(SELECT 'skirt' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Skirts)

UNION

(SELECT 'belt' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Belts)

UNION

(SELECT 'shirt' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Shirts)

UNION

(SELECT 'pants' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Pants)

UNION

(SELECT 'boots' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Boots)

UNION

(SELECT 'neck\_accessory' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Neck\_accessories)

UNION

(SELECT 'head\_accessory' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Head\_accessories)

;

## Detailed\_aprons

**Opis:**

* Pokazuje dokładne informacje o fartuszkach.

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_aprons ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, pattern ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, a.length, p.name AS "pattern"

FROM Aprons a

INNER JOIN Costumes\_items ci

ON a.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

INNER JOIN Patterns p

ON a.pattern\_id=p.id

;

## Detailed\_boots

**Opis:**

* Pokazuje dokładne informacje o butach.

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_boots ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, shoe\_size ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, b.shoe\_size

FROM Boots b

INNER JOIN Costumes\_items ci

ON b.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_petticoats

**Opis**:

* Pokazuje dokłądne informacje o halkach.

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_petticoats ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, min\_waist\_circumference, max\_waist\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, p.length, p.min\_waist\_circumference, p.max\_waist\_circumference

FROM Petticoats p

INNER JOIN Costumes\_items ci

ON p.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_skirts

**Opis:**

* Pokazuje dokładne informacje o spódnicach.

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_skirts ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, min\_waist\_circumference, max\_waist\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, s.length, s.min\_waist\_circumference, s.max\_waist\_circumference

FROM Skirts s

INNER JOIN Costumes\_items ci

ON s.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_caftans

**Opis:**

* Pokazuje dokładne informacje o kaftanach.

**Implementacja:**

CREATE OR REPLACE VIEW Detailed\_caftans ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, min\_waist\_circumference, max\_waist\_circumference, min\_chest\_circumference, max\_chest\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, c.length, c.min\_waist\_circumference, c.max\_waist\_circumference, c.min\_chest\_circumference, c.max\_chest\_circumference

FROM Caftans c

INNER JOIN Costumes\_items ci

ON c.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_corsets

**Opis**:

* Pokazuje dokładne informacje o gorsetach.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_corsets ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, min\_waist\_circumference, max\_waist\_circumference, min\_chest\_circumference, max\_chest\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, c.length, c.min\_waist\_circumference, c.max\_waist\_circumference, c.min\_chest\_circumference, c.max\_chest\_circumference

FROM Corsets c

INNER JOIN Costumes\_items ci

ON c.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_neck\_accessories

**Opis**:

* Pokazuje dokładne informacje o akcesoriach na szyję.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_neck\_accessories ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, min\_neck\_circumference, max\_neck\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, na.min\_neck\_circumference, na.max\_neck\_circumference

FROM Neck\_accessories na

INNER JOIN Costumes\_items ci

ON na.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_head\_accessories

**Opis**:

* Pokazuje informacje o akcesoriach na głowę.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_head\_accessories ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, head\_circumference, category ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, ha.head\_circumference, hac.name AS "category"

FROM Head\_accessories ha

INNER JOIN Costumes\_items ci

ON ha.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

INNER JOIN Head\_accessory\_categories hac

ON ha.category\_id=hac.id

;

## Detailed\_belts

**Opis**:

* Pokazuje dokładne informacje o pasach.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_belts ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, min\_waist\_circumference, max\_waist\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, b.min\_waist\_circumference, b.max\_waist\_circumference

FROM Belts b

INNER JOIN Costumes\_items ci

ON b.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_pants

**Opis**:

* Pokazuje dokładne informacje o spodniach.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_pants ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, min\_waist\_circumference, max\_waist\_circumference ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, p.length, p.min\_waist\_circumference, p.max\_waist\_circumference

FROM Pants p

INNER JOIN Costumes\_items ci

ON p.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Detailed\_shirts

**Opis**:

* Pokazuje dokładne informacje o koszulach.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_shirts ( id, name, collection, color, gender, street, building\_number, postal\_code, settlement, region, country, length, min\_waist\_circumference, max\_waist\_circumference, min\_chest\_circumference, max\_chest\_circumference, min\_neck\_circumference, max\_neck\_circumference, arm\_length ) AS

SELECT ci.id, ci.name, collec.name AS "collection", color.name AS "color", g.name AS "gender", l.street, l.building\_number, l.postal\_code, l.settlement, l.region, l.country, s.length, s.min\_waist\_circumference, s.max\_waist\_circumference, s.min\_chest\_circumference, s.max\_chest\_circumference, s.min\_neck\_circumference, s.max\_neck\_circumference, s.arm\_length

FROM Shirts s

INNER JOIN Costumes\_items ci

ON s.costume\_item\_id=ci.id

INNER JOIN Collections collec

ON ci.collection\_id=collec.id

INNER JOIN Colors color

ON ci.color\_id=color.id

INNER JOIN Genders g

ON ci.gender\_id=g.id

INNER JOIN Locations\_with\_settlements\_regions\_countries l

ON ci.location\_id=l.id

;

## Costume\_with\_costume\_items\_name

**Opis**:

* Pokazuje dokładne informacje strojach (zbiorze elementów stroju – tylko nazwy).

**Implementacja**:

CREATE OR REPLACE VIEW Costume\_with\_costume\_items\_name ( id, name, collection, gender, apron, caftan, petticoate, corset, skirt, belt, shirt, pants, boots, neck\_accessory, head\_accessory ) AS

SELECT c.id, c.name, col.name AS "collection", g.name AS "gender", COALESCE(a.name, 'N/A') AS "apron", COALESCE(ca.name, 'N/A') AS "caftan", COALESCE(p.name, 'N/A') AS "petticoate", COALESCE(co.name, 'N/A') AS "corset", COALESCE(sk.name, 'N/A') AS "skirt", COALESCE(b.name, 'N/A') AS "belt", COALESCE(sh.name, 'N/A') AS "shirt", COALESCE(pa.name, 'N/A') AS "pants", COALESCE(bo.name, 'N/A') AS "boots", COALESCE(ne.name, 'N/A') AS "neck\_accessory", COALESCE(h.name, 'N/A') AS "head\_accessory"

FROM Costumes c

INNER JOIN Collections col

ON c.collection\_id=col.id

INNER JOIN Genders g

ON c.gender\_id=g.id

LEFT JOIN Aprons ia

ON c.apron\_id=ia.costume\_item\_id

LEFT JOIN Costumes\_items a

ON ia.costume\_item\_id=a.id

LEFT JOIN Caftans ica

ON c.caftan\_id=ica.costume\_item\_id

LEFT JOIN Costumes\_items ca

ON ica.costume\_item\_id=ca.id

LEFT JOIN Petticoats ip

ON c.petticoat\_id=ip.costume\_item\_id

LEFT JOIN Costumes\_items p

ON ip.costume\_item\_id=p.id

LEFT JOIN Corsets ico

ON c.corset\_id=ico.costume\_item\_id

LEFT JOIN Costumes\_items co

ON ico.costume\_item\_id=co.id

LEFT JOIN Skirts isk

ON c.skirt\_id=isk.costume\_item\_id

LEFT JOIN Costumes\_items sk

ON isk.costume\_item\_id=sk.id

LEFT JOIN Belts ib

ON c.belt\_id=ib.costume\_item\_id

LEFT JOIN Costumes\_items b

ON ib.costume\_item\_id=b.id

LEFT JOIN Shirts ish

ON c.shirt\_id=ish.costume\_item\_id

LEFT JOIN Costumes\_items sh

ON ish.costume\_item\_id=sh.id

LEFT JOIN Pants ipa

ON c.pants\_id=ipa.costume\_item\_id

LEFT JOIN Costumes\_items pa

ON ipa.costume\_item\_id=pa.id

LEFT JOIN Boots ibo

ON c.boots\_id=ibo.costume\_item\_id

LEFT JOIN Costumes\_items bo

ON ibo.costume\_item\_id=bo.id

LEFT JOIN Neck\_accessories ine

ON c.neck\_accessory\_id=ine.costume\_item\_id

LEFT JOIN Costumes\_items ne

ON ine.costume\_item\_id=ne.id

LEFT JOIN Head\_accessories ih

ON c.head\_accessory\_id=ih.costume\_item\_id

LEFT JOIN Costumes\_items h

ON ih.costume\_item\_id=h.id

;

## Not\_read\_notifications

**Opis**:

* Pokazuje informacje nieprzeczytanych powiadomieniach.

**Implementacja**:

CREATE OR REPLACE VIEW Not\_read\_notifications ( id, user\_id, content, datetime, due\_to\_request\_id ) AS

SELECT id, user\_id, content, datetime, due\_to\_request\_id

FROM Notifications

WHERE marked\_as\_read = 'F'

;

## Detailed\_rental\_costume\_item\_requests

**Opis**:

* Pokazuje dokładne informacje o requestach - wypożyczenie.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_rental\_costume\_item\_requests ( id, datetime, requester\_user\_id, state, costume\_item\_id, approver\_costumier\_id ) AS

SELECT r.id, r.datetime, r.requester\_user\_id, s.name AS "state", rr.costume\_item\_id, rr.approver\_costumier\_id

FROM Rental\_costume\_item\_requests rr

INNER JOIN Requests r

ON rr.request\_id=r.id

INNER JOIN States\_of\_requests s

ON r.state\_id=s.id

;

## Detailed\_return\_costume\_item\_requests

**Opis**:

Pokazuje dokładne informacje o requestach - oddanie.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_return\_costume\_item\_requests ( id, datetime, requester\_user\_id, state, costume\_item\_id, approver\_costumier\_id ) AS

SELECT r.id, r.datetime, r.requester\_user\_id, s.name AS "state", rr.costume\_item\_id, rr.approver\_costumier\_id

FROM Return\_costume\_item\_requests rr

INNER JOIN Requests r

ON rr.request\_id=r.id

INNER JOIN States\_of\_requests s

ON r.state\_id=s.id

;

## Detailed\_borrow\_costume\_item\_requests

**Opis**:

* Pokazuje dokładne informacje o requestach - pożyczenie.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_borrow\_costume\_item\_requests ( id, datetime, requester\_user\_id, state, costume\_item\_id, approver\_user\_id ) AS

SELECT r.id, r.datetime, r.requester\_user\_id, s.name AS "state", rr.costume\_item\_id, rr.approver\_user\_id

FROM Borrow\_costume\_item\_requests rr

INNER JOIN Requests r

ON rr.request\_id=r.id

INNER JOIN States\_of\_requests s

ON r.state\_id=s.id

;

## Detailed\_costume\_item\_requests

**Opis**:

* Pokazuje dokładne informacje o requestach dotyczących elementu stroju.

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_costume\_item\_requests ( id, datetime, type, requester\_user\_id, state, costume\_item\_id, approver\_id ) AS

(SELECT d.id, d.datetime, 'RENTAL' AS "type", d.requester\_user\_id, d.state, d.costume\_item\_id, d.approver\_costumier\_id

FROM Detailed\_rental\_costume\_item\_requests d)

UNION

(SELECT d.id, d.datetime, 'RETURN' AS "type", d.requester\_user\_id, d.state, d.costume\_item\_id, d.approver\_costumier\_id

FROM Detailed\_return\_costume\_item\_requests d)

UNION

(SELECT d.id, d.datetime, 'BORROW' AS "type", d.requester\_user\_id, d.state, d.costume\_item\_id, d.approver\_user\_id

FROM Detailed\_borrow\_costume\_item\_requests d)

;

## Current\_rentals\_count\_by\_costume\_item\_class

**Opis**:

* Pokazuje, ile elementów stroju jest wypożyczonych z podziałem na klasę elementu.

**Implementacja**:

CREATE OR REPLACE VIEW Current\_rentals\_count\_by\_costume\_item\_class ( costume\_item\_class, number\_of\_rent\_items ) AS

SELECT \*

FROM ((SELECT 'apron' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_rent\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Aprons))

UNION

(SELECT 'caftan' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Caftans))

UNION

(SELECT 'petticoat' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Petticoats))

UNION

(SELECT 'corset' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Corsets))

UNION

(SELECT 'skirt' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Skirts))

UNION

(SELECT 'belt' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Belts))

UNION

(SELECT 'shirt' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Shirts))

UNION

(SELECT 'pants' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Pants))

UNION

(SELECT 'boots' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Boots))

UNION

(SELECT 'neck\_accessory' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Neck\_accessories))

UNION

(SELECT 'head\_accessory' AS "costume\_item\_class", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND costume\_item\_id IN (SELECT costume\_item\_id FROM Head\_accessories))) t

ORDER BY

t.costume\_item\_class

ASC

;

## Current\_rentals\_count\_by\_user\_function

**Opis**:

* Pokazuje, ile elementów strojów jest wypożyczonych przez członka o danej funkcji [członek chóru, ...]. Co jeśli członek pełni kilka funkcji?

**Implementacja**:

CREATE OR REPLACE VIEW Current\_rentals\_count\_by\_user\_function ( costume\_item\_class, number\_of\_rent\_items ) AS

SELECT \*

FROM ((SELECT 'costumier' AS "user\_function", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND user\_id IN (SELECT user\_id FROM Costumiers))

UNION

(SELECT 'singer' AS "user\_function", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND user\_id IN (SELECT user\_id FROM Singers))

UNION

(SELECT 'musician' AS "user\_function", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND user\_id IN (SELECT user\_id FROM Musicians))

UNION

(SELECT 'dancer' AS "user\_function", COUNT(\*) AS "number\_of\_items"

FROM Rentals

WHERE date\_of\_return IS NULL AND user\_id IN (SELECT user\_id FROM Dancers))) t

ORDER BY

t.user\_function

ASC

;

## Detailed\_rentals

**Opis**:

* Pokazuje, informacje o wypożyczeniach zawierających imię i nazwisko użytkownika oraz nazwę elementu stroju

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_rentals ( id, user\_id, user\_first\_name, user\_last\_name, costume\_item\_id, costume\_item\_name, done\_due\_request\_id, date\_of\_rental, date\_of\_return ) AS

SELECT r.id, r.user\_id, u.first\_name AS "user\_first\_name", u.last\_name AS "user\_last\_name", r.costume\_item\_id, ci.name AS "costume\_item\_name", r.done\_due\_request\_id, r.date\_of\_rental, r.date\_of\_return

FROM Rentals r

INNER JOIN Users u

ON r.user\_id = u.id

INNER JOIN Costumes\_items ci

ON r.costume\_item\_id = ci.id

;

## Detailed\_current\_rentals

**Opis**:

* Pokazuje, informacje o obecnych wypożyczeniach zawierających imię i nazwisko użytkownika oraz nazwę elementu stroju

**Implementacja**:

CREATE OR REPLACE VIEW Detailed\_current\_rentals ( id, user\_id, user\_first\_name, user\_last\_name, costume\_item\_id, costume\_item\_name, done\_due\_request\_id, date\_of\_rental, date\_of\_return ) AS

SELECT d.id, d.user\_id, d.user\_first\_name, d.user\_last\_name, d.costume\_item\_id, d.costume\_item\_name, d.done\_due\_request\_id, d.date\_of\_rental, d.date\_of\_return

FROM Detailed\_rentals d

WHERE d.date\_of\_return IS NULL

ORDER BY

d.date\_of\_rental

ASC

;

# Funkcje

## check\_costume\_inconsistency

**Opis:**

* Sprawdza czy elementy stroju mają niezgodną płeć i kolekcję. Zwraca TRUE kiedy występuje niekonsekwencja.

**Implementacja:**

CREATE FUNCTION check\_costume\_inconsistency(

f\_collection\_id SMALLINT,

f\_gender\_id SMALLINT,

f\_apron\_id INTEGER,

f\_caftan\_id INTEGER,

f\_petticoat\_id INTEGER,

f\_corset\_id INTEGER,

f\_skirt\_id INTEGER,

f\_belt\_id INTEGER,

f\_shirt\_id INTEGER,

f\_pants\_id INTEGER,

f\_boots\_id INTEGER,

f\_neck\_accessory\_id INTEGER,

f\_head\_accessory\_id INTEGER

)

RETURNS BOOLEAN AS $$

DECLARE

inconsistency\_found BOOLEAN := FALSE;

BEGIN

IF f\_apron\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_apron\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Apron does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_apron\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Apron does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_belt\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_belt\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Belt does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_belt\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Belt does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_boots\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_boots\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Boots do not match collection % or are not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_boots\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Boots do not match gender % or are not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_caftan\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_caftan\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Caftan does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_caftan\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Caftan does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_corset\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_corset\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Corset does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_corset\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Corset does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_petticoat\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_petticoat\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Petticoat does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_petticoat\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Petticoat does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_skirt\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_skirt\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Skirt does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_skirt\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Skirt does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_shirt\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_shirt\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Shirt does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_shirt\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Shirts does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_pants\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_pants\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Pants does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_pants\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Pants does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_neck\_accessory\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_neck\_accessory\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Neck accessory does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_neck\_accessory\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Neck accessory does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF f\_head\_accessory\_id IS NOT NULL THEN

PERFORM 1

FROM Costumes\_items

WHERE id = f\_head\_accessory\_id AND (collection\_id = f\_collection\_id OR collection\_id = 1);

IF NOT FOUND THEN

RAISE NOTICE 'Head accessory does not match collection % or is not universal', f\_collection\_id;

inconsistency\_found := TRUE;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE id = f\_head\_accessory\_id AND (gender\_id = f\_gender\_id OR gender\_id = 3);

IF NOT FOUND THEN

RAISE NOTICE 'Head accessory does not match gender % or is not bigender', f\_gender\_id;

inconsistency\_found := TRUE;

END IF;

END IF;

IF inconsistency\_found THEN

RETURN TRUE;

END IF;

RETURN FALSE;

END;

$$ LANGUAGE plpgsql;

## check\_rental\_inconsistency

**Opis:**

* Sprawdza czy dane w wypożyczeniu zgadzają się z danymi z requesta. Zwraca TRUE kiedy występuje nie konsekwencja.

**Implementacja:**

CREATE FUNCTION check\_rental\_inconsistency(

f\_user\_id INTEGER,

f\_costume\_item\_id INTEGER,

f\_done\_due\_request\_id INTEGER

)

RETURNS BOOLEAN AS $$

DECLARE

r\_user\_id INT;

r\_costume\_item\_id INT;

inconsistency\_found BOOLEAN := FALSE;

BEGIN

SELECT r.requester\_user\_id, COALESCE(ren\_r.costume\_item\_id, ret\_r.costume\_item\_id, b\_r.costume\_item\_id) INTO r\_user\_id, r\_costume\_item\_id

FROM Requests r

LEFT JOIN Rental\_costume\_item\_requests ren\_r

ON r.id=ren\_r.request\_id

LEFT JOIN Return\_costume\_item\_requests ret\_r

ON r.id=ret\_r.request\_id

LEFT JOIN Borrow\_costume\_item\_requests b\_r

ON r.id=b\_r.request\_id

WHERE r.id = f\_done\_due\_request\_id;

IF f\_user\_id <> r\_user\_id THEN

RAISE NOTICE 'user\_id are not consistency with request %', f\_done\_due\_request\_id;

inconsistency\_found := TRUE;

END IF;

IF f\_costume\_item\_id <> r\_costume\_item\_id THEN

RAISE NOTICE 'costume\_item\_id are not consistency with request %', f\_done\_due\_request\_id;

inconsistency\_found := TRUE;

END IF;

IF inconsistency\_found THEN

RETURN TRUE;

END IF;

RETURN FALSE;

END;

$$ LANGUAGE plpgsql;

## get\_costume\_item\_rental\_history

**Opis:**

* Funkcja zwraca chronologicznie (od najstarszej) historię wypożyczeń danego elementu stroju: nazwę, imię, nazwisko, daty wypożyczeń

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_costume\_item\_rental\_history(

f\_costume\_item\_id INT

)

RETURNS TABLE (

costume\_item\_name VARCHAR,

user\_id INT,

user\_first\_name VARCHAR,

user\_last\_name VARCHAR,

date\_of\_rental TIMESTAMP,

date\_of\_return TIMESTAMP

) AS $$

BEGIN

RETURN QUERY

SELECT

d.costume\_item\_name,

d.user\_id,

d.user\_first\_name,

d.user\_last\_name,

d.date\_of\_rental,

d.date\_of\_return

FROM

Detailed\_rentals d

WHERE

d.costume\_item\_id = f\_costume\_item\_id

ORDER BY

d.date\_of\_rental

ASC;

END;

$$ LANGUAGE plpgsql;

## check\_if\_error\_in\_costume\_item\_common\_part

**Opis:**

* Sprawdza czy element stroju ma poprawną cześć współną id kolekcji, płci, koloru i lokalizacji. Zwraca TRUE kiedy występuje błąd.

**Implementacja:**

CREATE FUNCTION check\_if\_error\_in\_costume\_item\_common\_part(

f\_collection\_id SMALLINT,

f\_gender\_id SMALLINT,

f\_color\_id SMALLINT,

f\_location\_id SMALLINT

)

RETURNS BOOLEAN AS $$

DECLARE

error\_found BOOLEAN := FALSE;

BEGIN

PERFORM 1

FROM Collections

WHERE

id = f\_collection\_id;

IF NOT FOUND THEN

RAISE NOTICE 'Collection with id % does not exist', f\_collection\_id;

error\_found := TRUE;

END IF;

IF f\_gender\_id NOT IN (1, 2, 3) THEN

RAISE NOTICE 'Gender with id 1 (male) or 2 (female) or 3 (bigender) can be selected';

error\_found := TRUE;

END IF;

PERFORM 1

FROM Genders

WHERE

id = f\_gender\_id;

IF NOT FOUND THEN

RAISE NOTICE 'Gender with id % does not exist', f\_gender\_id;

error\_found := TRUE;

END IF;

PERFORM 1

FROM Colors

WHERE

id = f\_color\_id;

IF NOT FOUND THEN

RAISE NOTICE 'Color with id % does not exist', f\_color\_id;

error\_found := TRUE;

END IF;

PERFORM 1

FROM Locations

WHERE

id = f\_location\_id;

IF NOT FOUND THEN

RAISE NOTICE 'Location with id % does not exist', f\_location\_id;

error\_found := TRUE;

END IF;

IF error\_found THEN

RETURN TRUE;

END IF;

RETURN FALSE;

END;

$$ LANGUAGE plpgsql;

## get\_user\_rental\_history

**Opis:**

* Funkcja zwraca chronologicznie (od najstarszej) historię wypożyczeń danego użytkownika: imię, nazwisko, id i nazwę elementu stroju, daty wypożyczeń

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_user\_rental\_history(

f\_user\_id INT

)

RETURNS TABLE (

user\_first\_name VARCHAR,

user\_last\_name VARCHAR,

costume\_item\_id INT,

costume\_item\_name VARCHAR,

date\_of\_rental TIMESTAMP,

date\_of\_return TIMESTAMP

) AS $$

BEGIN

RETURN QUERY

SELECT

d.user\_first\_name,

d.user\_last\_name,

d.costume\_item\_id,

d.costume\_item\_name,

d.date\_of\_rental,

d.date\_of\_return

FROM

Detailed\_rentals d

WHERE

d.user\_id = f\_user\_id

ORDER BY

d.date\_of\_rental

ASC;

END;

$$ LANGUAGE plpgsql;

## get\_costumier\_unresolved\_requests

**Opis:**

Pokazuje informacje o requestach, które mają zostać obsłużone przez Kostiumologa: id status i czas requestu, id imie i nazwisko osoby, id i nazwę elementu stroju.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_costumier\_unresolved\_requests()

RETURNS TABLE (

request\_id INT,

request\_state VARCHAR,

request\_datetie TIMESTAMP,

user\_id INT,

user\_first\_name VARCHAR,

user\_last\_name VARCHAR,

costume\_item\_id INT,

costume\_item\_name VARCHAR

) AS $$

BEGIN

RETURN QUERY

SELECT \* FROM ((SELECT r.id, r.state, r.datetime, r.requester\_user\_id, u.first\_name, u.last\_name, r.costume\_item\_id, ci.name

FROM

Detailed\_rental\_costume\_item\_requests r

INNER JOIN

Users u

ON r.requester\_user\_id = u.id

INNER JOIN

Costumes\_items ci

ON r.costume\_item\_id = ci.id

WHERE

r.approver\_costumier\_id IS NULL)

UNION

(SELECT r.id, r.state, r.datetime, r.requester\_user\_id, u.first\_name, u.last\_name, r.costume\_item\_id, ci.name

FROM

Detailed\_return\_costume\_item\_requests r

INNER JOIN

Users u

ON r.requester\_user\_id = u.id

INNER JOIN

Costumes\_items ci

ON r.costume\_item\_id = ci.id

WHERE

r.approver\_costumier\_id IS NULL)) t

ORDER BY

t.datetime

ASC;

END;

$$ LANGUAGE plpgsql;

## get\_user\_unresolved\_borrow\_requests

**Opis:**

Pokazuje informacje o requestach, które mają zostać obsłużone przez User: id status i czas requestu, id imie i nazwisko osoby, id i nazwę elementu stroju.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_user\_unresolved\_borrow\_requests(

f\_user\_id INT

)

RETURNS TABLE (

request\_id INT,

request\_state TEXT,

request\_datetie TIMESTAMP,

user\_id INT,

user\_first\_name VARCHAR,

user\_last\_name VARCHAR,

costume\_item\_id INT,

costume\_item\_name VARCHAR

) AS $$

BEGIN

RETURN QUERY

SELECT

r.id,

r.state,

r.datetime,

r.requester\_user\_id,

u.first\_name,

u.last\_name,

r.costume\_item\_id,

ci.name

FROM

Detailed\_borrow\_costume\_item\_requests r

INNER JOIN

Users u

ON r.requester\_user\_id = u.id

INNER JOIN

Costumes\_items ci

ON r.costume\_item\_id = ci.id

WHERE

r.approver\_costumier\_id = f\_user\_id

AND

(r.state <> 'ACCEPT' AND r.state <> 'DENY')

ORDER BY

r.datetime

ASC;

END;

$$ LANGUAGE plpgsql;

## get\_user\_current\_rentals

**Opis:**

Pokazuje informacje obecnie wypożyczone elementy stroju użytkownika: id i date wypożyczenia, imie i nazwisko usera, id i nazwę elementu stroju

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_user\_current\_rentals(

f\_user\_id INT

)

RETURNS TABLE (

rental\_id INT,

date\_of\_rental TIMESTAMP,

user\_first\_name VARCHAR,

user\_last\_name VARCHAR,

costume\_item\_id INT,

costume\_item\_name VARCHAR,

) AS $$

BEGIN

RETURN QUERY

SELECT

d.id,

d.date\_of\_rental,

d.user\_first\_name,

d.user\_last\_name,

d.costume\_item\_id,

d.costume\_item\_name

FROM

Detailed\_current\_rentals d

WHERE

d.user\_id = f\_user\_id;

END;

$$ LANGUAGE plpgsql;

## get\_user\_function\_percentage

**Opis:**

Pokazuje w procentach jaką cześć zespołu stanowią człokowie o funkcjach.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_user\_function\_percentage()

RETURNS TABLE (

user\_function TEXT,

percentage\_of\_users\_with\_this\_function TEXT

) AS $$

BEGIN

RETURN QUERY

SELECT

ufc.user\_function,

CONCAT(((ufc.number\_of\_users\_with\_this\_function/(SELECT COUNT(\*) FROM Users))\*100), '%')

FROM

User\_function\_counts ufc;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_aprons

**Opis:**

Pokazuje informacje o fartuszkach, które wymiarowo pasują do użytkownika.  
Długość = 76% długości nogi +/- 5cm

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_aprons(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

pattern VARCHAR

) AS $$

DECLARE

user\_leg\_length SMALLINT := (SELECT u.leg\_length FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.pattern

FROM

Detailed\_aprons d

WHERE

d.length >= 0.76\*user\_leg\_length - 5

AND

d.length <= 0.76\*user\_leg\_length + 5;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_boots

**Opis:**

Pokazuje informacje o butach, które wymiarowo pasują do użytkownika.  
Numer buta pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_boots(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

shoe\_size FLOAT

) AS $$

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.shoe\_size

FROM

Detailed\_boots d

WHERE

d.shoe\_size = (SELECT u.shoe\_size FROM Users u WHERE u.id = f\_user\_id);

END;

$$ LANGUAGE plpgsql;

## get\_fits\_petticoats

**Opis:**

Pokazuje informacje o halkach które wymiarowo pasują do użytkownika.  
Długość = 76% długości nogi +/- 5cm. Obwód w pasie pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_petticoats(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT

) AS $$

DECLARE

user\_leg\_length SMALLINT := (SELECT u.leg\_length FROM Users u WHERE u.id = f\_user\_id);

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.min\_waist\_circumference,

d.max\_waist\_circumference

FROM

Detailed\_petticoats d

WHERE

d.length >= 0.76\*user\_leg\_length - 5

AND

d.length <= 0.76\*user\_leg\_length + 5

AND

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_skirts

**Opis:**

Pokazuje informacjie o spudnicach które wymiarowo pasują do użytkownika.  
Długość = 76% długości nogi +/- 5cm. Obwód w pasie pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_skirts(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT

) AS $$

DECLARE

user\_leg\_length SMALLINT := (SELECT u.leg\_length FROM Users u WHERE u.id = f\_user\_id);

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.min\_waist\_circumference,

d.max\_waist\_circumference

FROM

Detailed\_skirts d

WHERE

d.length >= 0.76\*user\_leg\_length - 5

AND

d.length <= 0.76\*user\_leg\_length + 5

AND

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_caftans

**Opis:**

Pokazuje informacjie o X które wymiarowo pasują do użytkownika.  
Długość >= 60% długości nogi. Obwód w pasie i klatce piersiowej pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_caftans(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT,

min\_chest\_circumference SMALLINT,

max\_chest\_circumference SMALLINT

) AS $$

DECLARE

user\_leg\_length SMALLINT := (SELECT u.leg\_length FROM Users u WHERE u.id = f\_user\_id);

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

user\_chest\_circumference SMALLINT := (SELECT u.chest\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.min\_waist\_circumference,

d.max\_waist\_circumference,

d.min\_chest\_circumference,

d.max\_chest\_circumference

FROM

Detailed\_caftans d

WHERE

d.length >= 0.60\*user\_leg\_length

AND

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference

AND

user\_chest\_circumference >= d.min\_chest\_circumference

AND

user\_chest\_circumference <= d.max\_chest\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_corsets

**Opis:**

Pokazuje informacjie o X które wymiarowo pasują do użytkownika.  
Długość >= torsu. Obwód w klatce piersiowej i pasie pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_corsets(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT,

min\_chest\_circumference SMALLINT,

max\_chest\_circumference SMALLINT

) AS $$

DECLARE

user\_torso\_length SMALLINT := (SELECT u.torso\_length FROM Users u WHERE u.id = f\_user\_id);

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

user\_chest\_circumference SMALLINT := (SELECT u.chest\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.min\_waist\_circumference,

d.max\_waist\_circumference,

d.min\_chest\_circumference,

d.max\_chest\_circumference

FROM

Detailed\_corsets d

WHERE

d.length >= user\_torso\_length

AND

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference

AND

user\_chest\_circumference >= d.min\_chest\_circumference

AND

user\_chest\_circumference <= d.max\_chest\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_neck\_accessories

**Opis:**

Pokazuje informacjie o akcesoriach na szyje, które wymiarowo pasują do użytkownika.  
Obdód szyii pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_neck\_accessories(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

min\_neck\_circumference SMALLINT,

max\_neck\_circumference SMALLINT

) AS $$

DECLARE

user\_neck\_circumference SMALLINT := (SELECT u.neck\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.min\_neck\_circumference,

d.max\_neck\_circumference

FROM

Detailed\_neck\_accessories d

WHERE

user\_neck\_circumference >= d.min\_neck\_circumference

AND

user\_neck\_circumference <= d.max\_neck\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_head\_accessories

**Opis:**

Pokazuje informacjie o akcesoriach na głowę, które wymiarowo pasują do użytkownika.  
Obwód głowy +/- 2cm.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_head\_accessories(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

head\_circumference SMALLINT,

category VARCHAR

) AS $$

DECLARE

user\_head\_circumference SMALLINT := (SELECT u.head\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.head\_circumference,

d.category

FROM

Detailed\_head\_accessories d

WHERE

d.head\_circumference IS NULL

OR

(user\_head\_circumference - 2 >= d.head\_circumference

AND

user\_head\_circumference + 2 <= d.head\_circumference);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get\_fits\_head\_accessories(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

head\_circumference SMALLINT,

category VARCHAR

) AS $$

DECLARE

user\_head\_circumference SMALLINT := (SELECT u.head\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.head\_circumference,

d.category

FROM

Detailed\_head\_accessories d

WHERE

d.head\_circumference IS NULL

OR

(user\_head\_circumference - 2 >= d.head\_circumference

AND

user\_head\_circumference + 2 <= d.head\_circumference);

END;

$$ LANGUAGE plpgsql;

## get\_fits\_belts

**Opis:**

Pokazuje informacjie o pasach, które wymiarowo pasują do użytkownika.  
Obwód w pasie pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_belts(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT

) AS $$

DECLARE

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.min\_waist\_circumference,

d.max\_waist\_circumference

FROM

Detailed\_belts d

WHERE

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_pants

**Opis:**

Pokazuje informacjie o spodniach, które wymiarowo pasują do użytkownika.  
Długość +/- 3cm długości nogi. Obwód w pasie pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_pants(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT

) AS $$

DECLARE

user\_leg\_length SMALLINT := (SELECT u.leg\_length FROM Users u WHERE u.id = f\_user\_id);

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.min\_waist\_circumference,

d.max\_waist\_circumference

FROM

Detailed\_pants d

WHERE

d.length >= user\_leg\_length - 3

AND

d.length <= user\_leg\_length + 3

AND

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_fits\_shirts

**Opis:**

Pokazuje informacjie o koszule, które wymiarowo pasują do użytkownika.  
Długość >= torsu. Długość rękawa >= ręki. Obwód w klatce piersiowej, pasie i szyi pasuje.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_fits\_shirts(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

name VARCHAR,

collection VARCHAR,

color VARCHAR,

gender VARCHAR,

length SMALLINT,

arm\_length SMALLINT,

min\_waist\_circumference SMALLINT,

max\_waist\_circumference SMALLINT,

min\_chest\_circumference SMALLINT,

max\_chest\_circumference SMALLINT,

min\_neck\_circumference SMALLINT,

max\_neck\_circumference SMALLINT

) AS $$

DECLARE

user\_torso\_length SMALLINT := (SELECT u.torso\_length FROM Users u WHERE u.id = f\_user\_id);

user\_arm\_length SMALLINT := (SELECT u.arm\_length FROM Users u WHERE u.id = f\_user\_id);

user\_waist\_circumference SMALLINT := (SELECT u.waist\_circumference FROM Users u WHERE u.id = f\_user\_id);

user\_chest\_circumference SMALLINT := (SELECT u.chest\_circumference FROM Users u WHERE u.id = f\_user\_id);

user\_neck\_circumference SMALLINT := (SELECT u.neck\_circumference FROM Users u WHERE u.id = f\_user\_id);

BEGIN

RETURN QUERY

SELECT

d.id,

d.name,

d.collection,

d.color,

d.gender,

d.length,

d.arm\_length,

d.min\_waist\_circumference,

d.max\_waist\_circumference,

d.min\_chest\_circumference,

d.max\_chest\_circumference,

d.min\_neck\_circumference,

d.max\_neck\_circumference

FROM

Detailed\_shirts d

WHERE

d.length >= user\_torso\_length

AND

d.arm\_length >= user\_arm\_length

AND

user\_waist\_circumference >= d.min\_waist\_circumference

AND

user\_waist\_circumference <= d.max\_waist\_circumference

AND

user\_chest\_circumference >= d.min\_chest\_circumference

AND

user\_chest\_circumference <= d.max\_chest\_circumference

AND

user\_neck\_circumference >= d.min\_neck\_circumference

AND

user\_neck\_circumference <= d.max\_neck\_circumference;

END;

$$ LANGUAGE plpgsql;

## get\_user\_unread\_notifications

**Opis:**

Pokazuje nieprzeczytane powiadomienia danego użytkownika.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_user\_unread\_notifications(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

content TEXT,

datetime TIMESTAMP,

due\_to\_request\_id INT

) AS $$

BEGIN

RETURN QUERY

SELECT

d.id,

d.content,

d.datetime,

d.due\_to\_request\_id

FROM

Not\_read\_notifications d

WHERE

d.user\_id = f\_user\_id;

END;

$$ LANGUAGE plpgsql;

## get\_user\_unclosed\_costume\_item\_requests

**Opis:**

Pokazuje requesty danego użytkownika, w których decyzja jeszcze nie zapadła.

**Implementacja:**

CREATE OR REPLACE FUNCTION get\_user\_unclosed\_costume\_item\_requests(

f\_user\_id INT

)

RETURNS TABLE (

id INT,

datetime TIMESTAMP,

type TEXT,

state VARCHAR,

costume\_item\_id INT,

approver\_id INT

) AS $$

BEGIN

RETURN QUERY

SELECT

d.id,

d.datetime,

d.type,

d.state,

d.costume\_item\_id,

d.approver\_id

FROM

Detailed\_costume\_item\_requests d

WHERE

d.requester\_user\_id = f\_user\_id;

END;

$$ LANGUAGE plpgsql;

# Procedury

## add\_country

**Opis:**

* Pozwala dodać kraj, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_country(

p\_country\_name VARCHAR(30)

) AS $$

BEGIN

IF p\_country\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_country\_name) > 30 OR LENGTH(p\_country\_name) < 1 THEN

RAISE EXCEPTION 'Country name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Countries

WHERE

name = p\_country\_name;

IF FOUND THEN

RAISE EXCEPTION 'Country already exist';

END IF;

BEGIN

INSERT INTO Countries (name)

VALUES (p\_country\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_region

**Opis**:

* Pozwala dodać region, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_region(

p\_region\_name VARCHAR(30),

p\_country\_id SMALLINT

) AS $$

BEGIN

IF p\_region\_name IS NULL OR p\_country\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_region\_name) > 30 OR LENGTH(p\_region\_name) < 1 THEN

RAISE EXCEPTION 'Region name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Countries

WHERE

id = p\_country\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Country with id % does not exist', p\_country\_id;

END IF;

PERFORM 1

FROM Regions

WHERE

name = p\_region\_name AND country\_id = p\_country\_id;

IF FOUND THEN

RAISE EXCEPTION 'Region % in country with id % already exist', p\_region\_name, p\_country\_id;

END IF;

BEGIN

INSERT INTO Regions (name, country\_id)

VALUES (p\_region\_name, p\_country\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_settlement

**Opis**:

* Pozwala dodać miejscowość, która spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_settlement(

p\_settlement\_name VARCHAR(30),

p\_region\_id SMALLINT

) AS $$

BEGIN

IF p\_settlement\_name IS NULL OR p\_region\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_settlement\_name) > 30 OR LENGTH(p\_settlement\_name) < 1 THEN

RAISE EXCEPTION 'Settlement name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Regions

WHERE

id = p\_region\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Region with id % does not exist', p\_region\_id;

END IF;

PERFORM 1

FROM Settlements

WHERE

name = p\_settlement\_name AND region\_id = p\_region\_id;

IF FOUND THEN

RAISE EXCEPTION 'Settlement % in region with id % already exist', p\_settlement\_name, p\_region\_id;

END IF;

BEGIN

INSERT INTO Settlements (name, region\_id)

VALUES (p\_settlement\_name, p\_region\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_location

**Opis:**

* Pozwala dodać lokalizację, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_location(

p\_location\_street VARCHAR(30),

p\_location\_building\_number SMALLINT,

p\_location\_postal\_code VARCHAR(10),

p\_settlement\_id SMALLINT

) AS $$

BEGIN

IF p\_location\_street IS NULL OR

p\_location\_building\_number IS NULL OR

p\_location\_postal\_code IS NULL OR

p\_settlement\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_location\_street) > 30 OR LENGTH(p\_location\_street) < 1 THEN

RAISE EXCEPTION 'Street name can have between 1 and 30 characters';

END IF;

IF LENGTH(p\_location\_postal\_code) > 10 OR LENGTH(p\_location\_postal\_code) < 1 THEN

RAISE EXCEPTION 'Postal code can have between 1 and 10 characters';

END IF;

PERFORM 1

FROM Settlements

WHERE

id = p\_settlement\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Settlement with id % does not exist', p\_settlement\_id;

END IF;

PERFORM 1

FROM Locations

WHERE

street = p\_location\_street AND building\_number = p\_location\_building\_number AND postal\_code = p\_location\_postal\_code AND settlement\_id = p\_settlement\_id;

IF FOUND THEN

RAISE EXCEPTION 'Location: %, %, %, in sattlement with id % already exist', p\_location\_street, p\_location\_building\_number, p\_location\_postal\_code, p\_settlement\_id;

END IF;

BEGIN

INSERT INTO Locations (street, building\_number, postal\_code, settlement\_id)

VALUES (p\_location\_street, p\_location\_building\_number, p\_location\_postal\_code, p\_settlement\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_gender

**Opis:**

* Pozwala dodać płeć, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_gender(

p\_gender\_name VARCHAR(25)

) AS $$

BEGIN

IF p\_gender\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_gender\_name) > 25 OR LENGTH(p\_gender\_name) < 1 THEN

RAISE EXCEPTION 'Gender name can have between 1 and 25 characters';

END IF;

PERFORM 1

FROM Genders

WHERE

name = p\_gender\_name;

IF FOUND THEN

RAISE EXCEPTION 'Gender already exist';

END IF;

BEGIN

INSERT INTO Genders (name)

VALUES (p\_gender\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_role

**Opis:**

* Pozwala dodać role, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_role(

p\_role\_name VARCHAR(20)

) AS $$

BEGIN

IF p\_role\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_role\_name) > 20 OR LENGTH(p\_role\_name) < 1 THEN

RAISE EXCEPTION 'Role name can have between 1 and 20 characters';

END IF;

PERFORM 1

FROM Roles

WHERE

name = p\_role\_name;

IF FOUND THEN

RAISE EXCEPTION 'Role already exist';

END IF;

BEGIN

INSERT INTO Roles (name)

VALUES (p\_role\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_type\_of\_voice

**Opis:**

* Pozwala dodać typ głosu, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_type\_of\_voice(

p\_type\_of\_voice\_name VARCHAR(10)

) AS $$

BEGIN

IF p\_type\_of\_voice\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_type\_of\_voice\_name) > 10 OR LENGTH(p\_type\_of\_voice\_name) < 1 THEN

RAISE EXCEPTION 'Type of voice name can have between 1 and 10 characters';

END IF;

PERFORM 1

FROM Types\_of\_voices

WHERE

name = p\_type\_of\_voice\_name;

IF FOUND THEN

RAISE EXCEPTION 'Type of voice already exist';

END IF;

BEGIN

INSERT INTO Types\_of\_voices (name)

VALUES (p\_type\_of\_voice\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_type\_of\_instrument

**Opis:**

* Pozwala dodać typ instrumentu, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_type\_of\_instrument(

p\_type\_of\_instrument\_name VARCHAR(20)

) AS $$

BEGIN

IF p\_type\_of\_instrument\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_type\_of\_instrument\_name) > 20 OR LENGTH(p\_type\_of\_instrument\_name) < 1 THEN

RAISE EXCEPTION 'Type of instrument name can have between 1 and 20 characters';

END IF;

PERFORM 1

FROM Types\_of\_instruments

WHERE

name = p\_type\_of\_instrument\_name;

IF FOUND THEN

RAISE EXCEPTION 'Type of instrument already exist';

END IF;

BEGIN

INSERT INTO Types\_of\_instruments (name)

VALUES (p\_type\_of\_instrument\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_dance

**Opis:**

* Pozwala dodać taniec, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_dance(

p\_dance\_name VARCHAR(20)

) AS $$

BEGIN

IF p\_dance\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_dance\_name) > 20 OR LENGTH(p\_dance\_name) < 1 THEN

RAISE EXCEPTION 'Dance name can have between 1 and 20 characters';

END IF;

PERFORM 1

FROM Dances

WHERE

name = p\_dance\_name;

IF FOUND THEN

RAISE EXCEPTION 'Dance already exist';

END IF;

BEGIN

INSERT INTO Dances (name)

VALUES (p\_dance\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_color

**Opis:**

* Pozwala dodać kolor, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_color(

p\_color\_name VARCHAR(25)

) AS $$

BEGIN

IF p\_color\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_color\_name) > 25 OR LENGTH(p\_color\_name) < 1 THEN

RAISE EXCEPTION 'Color name can have between 1 and 25 characters';

END IF;

PERFORM 1

FROM Colors

WHERE

name = p\_color\_name;

IF FOUND THEN

RAISE EXCEPTION 'Color already exist';

END IF;

BEGIN

INSERT INTO Colors (name)

VALUES (p\_color\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_collection

**Opis:**

* Pozwala dodać kolekcje, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_collection(

p\_collection\_name VARCHAR(20)

) AS $$

BEGIN

IF p\_collection\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_collection\_name) > 20 OR LENGTH(p\_collection\_name) < 1 THEN

RAISE EXCEPTION 'Collection name can have between 1 and 20 characters';

END IF;

PERFORM 1

FROM Collections

WHERE

name = p\_collection\_name;

IF FOUND THEN

RAISE EXCEPTION 'Collection already exist';

END IF;

BEGIN

INSERT INTO Collections (name)

VALUES (p\_collection\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_pattern

**Opis:**

* Pozwala dodać wzór, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_pattern(

p\_pattern\_name VARCHAR(20)

) AS $$

BEGIN

IF p\_pattern\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_pattern\_name) > 20 OR LENGTH(p\_pattern\_name) < 1 THEN

RAISE EXCEPTION 'Pattern name can have between 1 and 20characters';

END IF;

PERFORM 1

FROM Patterns

WHERE

name = p\_pattern\_name;

IF FOUND THEN

RAISE EXCEPTION 'Pattern already exist';

END IF;

BEGIN

INSERT INTO Patterns (name)

VALUES (p\_pattern\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_head\_accessory\_category

**Opis:**

* Pozwala dodać kategorię akcesoria głowy, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_head\_accessory\_category(

p\_head\_accessory\_category\_name VARCHAR(20)

) AS $$

BEGIN

IF p\_head\_accessory\_category\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_head\_accessory\_category\_name) > 20 OR LENGTH(p\_head\_accessory\_category\_name) > 20 THEN

RAISE EXCEPTION 'Head accessory category name can have between 1 and 20 characters';

END IF;

PERFORM 1

FROM Head\_accessory\_categories

WHERE

name = p\_head\_accessory\_category\_name;

IF FOUND THEN

RAISE EXCEPTION 'Head accessory category already exist';

END IF;

BEGIN

INSERT INTO Head\_accessory\_categories (name)

VALUES (p\_head\_accessory\_category\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_state\_of\_request

**Opis:**

* Pozwala dodać stan żądania, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_state\_of\_request(

p\_state\_of\_request\_name VARCHAR(15)

) AS $$

BEGIN

IF p\_state\_of\_request\_name IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_state\_of\_request\_name) > 15 OR LENGTH(p\_state\_of\_request\_name) < 1 THEN

RAISE EXCEPTION 'State of request name can have between 1 and 15 characters';

END IF;

PERFORM 1

FROM States\_of\_requests

WHERE

name = p\_state\_of\_request\_name;

IF FOUND THEN

RAISE EXCEPTION 'State of request already exist';

END IF;

BEGIN

INSERT INTO States\_of\_requests (name)

VALUES (p\_state\_of\_request\_name);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_user

**Opis:**

* Pozwala dodać użytkownika, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_user(

p\_user\_first\_name VARCHAR (25),

p\_user\_last\_name VARCHAR (30),

p\_user\_date\_of\_birth DATE,

p\_user\_email VARCHAR (100),

p\_user\_phone\_number VARCHAR (12),

p\_gender\_id SMALLINT,

p\_home\_location\_id SMALLINT,

p\_user\_height SMALLINT,

p\_user\_waist\_circumference SMALLINT,

p\_user\_chest\_circumference SMALLINT,

p\_user\_head\_circumference SMALLINT,

p\_user\_neck\_circumference SMALLINT,

p\_user\_leg\_length SMALLINT,

p\_user\_arm\_length SMALLINT,

p\_user\_torso\_length SMALLINT,

p\_user\_shoe\_size FLOAT

) AS $$

BEGIN

IF p\_user\_first\_name IS NULL OR

p\_user\_last\_name IS NULL OR

p\_user\_date\_of\_birth IS NULL OR

p\_user\_email IS NULL OR

p\_user\_phone\_number IS NULL OR

p\_gender\_id IS NULL OR

p\_home\_location\_id IS NULL OR

p\_user\_height IS NULL OR

p\_user\_waist\_circumference IS NULL OR

p\_user\_chest\_circumference IS NULL OR

p\_user\_head\_circumference IS NULL OR

p\_user\_neck\_circumference IS NULL OR

p\_user\_leg\_length IS NULL OR

p\_user\_arm\_length IS NULL OR

p\_user\_torso\_length IS NULL OR

p\_user\_shoe\_size IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF LENGTH(p\_user\_first\_name) > 25 OR LENGTH(p\_user\_first\_name) < 1 THEN

RAISE EXCEPTION 'First name can have between 1 and 25 characters';

END IF;

IF LENGTH(p\_user\_last\_name) > 30 OR LENGTH(p\_user\_last\_name) < 1 THEN

RAISE EXCEPTION 'Last name can have between 1 and 30 characters';

END IF;

IF LENGTH(p\_user\_email) > 50 OR NOT (p\_user\_email ~\* '^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$') THEN

RAISE EXCEPTION 'Wrong email format. min characters - 5, max characters - 50';

END IF;

IF NOT (p\_user\_phone\_number ~\* '^\+\d{2}\d{9}$') THEN

RAISE EXCEPTION 'Wrong phone number format';

END IF;

IF p\_user\_height <= 0 THEN

RAISE EXCEPTION 'Height must be greater than 0';

END IF;

IF p\_user\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Waist circumference must be greater than 0';

END IF;

IF p\_user\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Chest circumference must be greater than 0';

END IF;

IF p\_user\_head\_circumference <= 0 THEN

RAISE EXCEPTION 'Head circumference must be greater than 0';

END IF;

IF p\_user\_neck\_circumference <= 0 THEN

RAISE EXCEPTION 'Neck circumference must be greater than 0';

END IF;

IF p\_user\_leg\_length <= 0 THEN

RAISE EXCEPTION 'Leg length must be greater than 0';

END IF;

IF p\_user\_arm\_length <= 0 THEN

RAISE EXCEPTION 'Arm length must be greater than 0';

END IF;

IF p\_user\_torso\_length <= 0 THEN

RAISE EXCEPTION 'Torso length must be greater than 0';

END IF;

IF p\_user\_leg\_length >= p\_user\_height THEN

RAISE EXCEPTION 'Leg length cannot be greater than height';

END IF;

IF p\_user\_arm\_length >= p\_user\_height THEN

RAISE EXCEPTION 'Arm length cannot be greater than height';

END IF;

IF p\_user\_torso\_length >= p\_user\_height THEN

RAISE EXCEPTION 'Torso length cannot be greater than height';

END IF;

IF p\_user\_shoe\_size <= 0 THEN

RAISE EXCEPTION 'Shoe size must be greater than 0';

END IF;

PERFORM 1

FROM Genders

WHERE

id = p\_gender\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Gender with id % does not exist', p\_gender\_id;

END IF;

PERFORM 1

FROM Locations

WHERE

id = p\_home\_location\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Location with id % does not exist', p\_home\_location\_id;

END IF;

PERFORM 1

FROM Users

WHERE

email = p\_user\_email;

IF FOUND THEN

RAISE EXCEPTION 'User with email % already exist', p\_user\_email;

END IF;

BEGIN

INSERT INTO Users (first\_name, last\_name, date\_of\_birth, email, phone\_number, gender\_id, home\_location\_id,

height, waist\_circumference, chest\_circumference, head\_circumference, neck\_circumference, leg\_length, arm\_length,

torso\_length, shoe\_size)

VALUES (p\_user\_first\_name, p\_user\_last\_name, p\_user\_date\_of\_birth, p\_user\_email, p\_user\_phone\_number,

p\_gender\_id, p\_home\_location\_id, p\_user\_height, p\_user\_waist\_circumference, p\_user\_chest\_circumference,

p\_user\_head\_circumference, p\_user\_neck\_circumference, p\_user\_leg\_length, p\_user\_arm\_length,

p\_user\_torso\_length, p\_user\_shoe\_size);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_costumier

**Opis:**

* Pozwala dodać członkowi zespołu funkcję Kostiumologa, którą będzie pełnił oraz jej parametry, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE make\_user\_costumier(

p\_user\_id INT,

p\_role\_id SMALLINT,

p\_work\_location\_id SMALLINT

) AS $$

BEGIN

IF p\_user\_id IS NULL OR

p\_role\_id IS NULL OR

p\_work\_location\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Roles

WHERE

id = p\_role\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Role with id % does not exist', p\_role\_id;

END IF;

PERFORM 1

FROM Locations

WHERE

id = p\_work\_location\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Location with id % does not exist', p\_work\_location\_id;

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_user\_id;

END IF;

PERFORM 1

FROM Costumiers

WHERE

user\_id = p\_user\_id;

IF FOUND THEN

RAISE EXCEPTION 'User with id % is costumier', p\_user\_id;

END IF;

BEGIN

PERFORM 1 FROM Users WHERE id = p\_user\_id FOR UPDATE;

INSERT INTO Costumiers (user\_id, role\_id, work\_location\_id)

VALUES (p\_user\_id, p\_role\_id, p\_work\_location\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_singer

**Opis:**

* Pozwala dodać członkowi zespołu funkcję członka chóru, którą będzie pełnił oraz jej parametry, która spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE make\_user\_singer(

p\_user\_id INT,

p\_role\_id SMALLINT

) AS $$

BEGIN

IF p\_user\_id IS NULL OR

p\_role\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Roles

WHERE

id = p\_role\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Role with id % does not exist', p\_role\_id;

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_user\_id;

END IF;

PERFORM 1

FROM Singers

WHERE

user\_id = p\_user\_id;

IF FOUND THEN

RAISE EXCEPTION 'User with id % is singer', p\_user\_id;

END IF;

BEGIN

PERFORM 1 FROM Users WHERE id = p\_user\_id FOR UPDATE;

INSERT INTO Singers (user\_id, role\_id)

VALUES (p\_user\_id, p\_role\_id); EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_musician

**Opis**:

* Pozwala dodać członkowi zespołu funkcję członka kapeli, którą będzie pełnił oraz jej parametry, która spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE make\_user\_musician(

p\_user\_id INT,

p\_role\_id SMALLINT

) AS $$

BEGIN

IF p\_user\_id IS NULL OR

p\_role\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Roles

WHERE

id = p\_role\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Role with id % does not exist', p\_role\_id;

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_user\_id;

END IF;

PERFORM 1

FROM Musicians

WHERE

user\_id = p\_user\_id;

IF FOUND THEN

RAISE EXCEPTION 'User with id % is musician', p\_user\_id;

END IF;

BEGIN

PERFORM 1 FROM Users WHERE id = p\_user\_id FOR UPDATE;

INSERT INTO Musicians (user\_id, role\_id)

VALUES (p\_user\_id, p\_role\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_dancer

**Opis:**

* Pozwala dodać członkowi zespołu funkcję członka baletu, którą będzie pełnił oraz jej parametry, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE make\_user\_dancer(

p\_user\_id INT,

p\_role\_id SMALLINT

) AS $$

BEGIN

IF p\_user\_id IS NULL OR

p\_role\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Roles

WHERE

id = p\_role\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Role with id % does not exist', p\_role\_id;

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_user\_id;

END IF;

PERFORM 1

FROM Dancers

WHERE

user\_id = p\_user\_id;

IF FOUND THEN

RAISE EXCEPTION 'User with id % is dancer', p\_user\_id;

END IF;

BEGIN

PERFORM 1 FROM Users WHERE id = p\_user\_id FOR UPDATE;

INSERT INTO Dancers (user\_id, role\_id)

VALUES (p\_user\_id, p\_role\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_voice\_to\_singer

**Opis:**

* Pozwala dodać typ głosu, którym umie śpiewać członek chóru, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_voice\_to\_singer(

p\_singer\_id INT,

p\_type\_of\_voice\_id SMALLINT

) AS $$

BEGIN

IF p\_singer\_id IS NULL OR

p\_type\_of\_voice\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Types\_of\_voices

WHERE

id = p\_type\_of\_voice\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Type of voice with id % does not exist', p\_type\_of\_voice\_id;

END IF;

PERFORM 1

FROM Singers

WHERE

user\_id = p\_singer\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Singer with id % does not exist', p\_singer\_id;

END IF;

PERFORM 1

FROM Singer\_voices

WHERE

singer\_id = p\_singer\_id AND type\_of\_voice\_id = p\_type\_of\_voice\_id;

IF FOUND THEN

RAISE EXCEPTION 'Singer with id % can sing with voice of id %', p\_singer\_id, p\_type\_of\_voice\_id;

END IF;

BEGIN

PERFORM 1 FROM Singers WHERE user\_id = p\_singer\_id FOR UPDATE;

INSERT INTO Singer\_voices (singer\_id, type\_of\_voice\_id)

VALUES (p\_singer\_id, p\_type\_of\_voice\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_instrument\_to\_musician

**Opis**:

* Pozwala dodać typ instrumentu, na którym umie grać członek kapeli, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_instrument\_to\_musician(

p\_musician\_id INT,

p\_type\_of\_instrument\_id SMALLINT

) AS $$

BEGIN

IF p\_musician\_id IS NULL OR

p\_type\_of\_instrument\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Types\_of\_instruments

WHERE

id = p\_type\_of\_instrument\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Type of instrument with id % does not exist', p\_type\_of\_instrument\_id;

END IF;

PERFORM 1

FROM Musicians

WHERE

user\_id = p\_musician\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Musician with id % does not exist', p\_musician\_id;

END IF;

PERFORM 1

FROM Musician\_instrument

WHERE

musician\_id = p\_musician\_id AND type\_of\_instrument\_id = p\_type\_of\_instrument\_id;

IF FOUND THEN

RAISE EXCEPTION 'Musician with id % can paly on instrument with id %', p\_musician\_id, p\_type\_of\_instrument\_id;

END IF;

BEGIN

PERFORM 1 FROM Musicians WHERE user\_id = p\_musician\_id FOR UPDATE;

INSERT INTO Musician\_instrument (musician\_id, type\_of\_instrument\_id)

VALUES (p\_musician\_id, p\_type\_of\_instrument\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_dance\_to\_dancer

**Opis:**

* Pozwala dodać taniec, który umie tańczyć członek baletu, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_dance\_to\_dancer(

p\_dancer\_id INT,

p\_dance\_id SMALLINT

) AS $$

BEGIN

IF p\_dancer\_id IS NULL OR

p\_dance\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Dances

WHERE

id = p\_dance\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Dance with id % does not exist', p\_dance\_id;

END IF;

PERFORM 1

FROM Dancers

WHERE

user\_id = p\_dancer\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Dancer with id % does not exist', p\_dancer\_id;

END IF;

PERFORM 1

FROM Dancer\_dance

WHERE

dancer\_id = p\_dancer\_id AND dance\_id = p\_dance\_id;

IF FOUND THEN

RAISE EXCEPTION 'Dancer with id % can dance dance with id %', p\_dancer\_id, p\_dance\_id;

END IF;

BEGIN

PERFORM 1 FROM Dancers WHERE user\_id = p\_dancer\_id FOR UPDATE;

INSERT INTO Dancer\_dance (dancer\_id, dance\_id)

VALUES (p\_dancer\_id, p\_dance\_id);

--COMMIT;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_apron

**Opis**:

* Pozwala dodać fartuszek, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_apron(

p\_apron\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_apron\_length SMALLINT,

p\_pattern\_id SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_apron\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_apron\_length IS NULL OR

p\_pattern\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_apron\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF LENGTH(p\_apron\_name) > 30 OR LENGTH(p\_apron\_name) < 1 THEN

RAISE EXCEPTION 'Apron name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_apron\_name;

IF FOUND THEN

RAISE EXCEPTION 'Apron with name % already exist', p\_apron\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_apron\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Aprons (costume\_item\_id, length, pattern\_id)

VALUES (i\_id, p\_apron\_length, p\_pattern\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_head\_accessory

**Opis**:

* Pozwala dodać akcesorium na głowę, które spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_head\_accessory(

p\_head\_accessory\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_head\_accessory\_head\_circumference SMALLINT,

p\_category\_id SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_head\_accessory\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_category\_id IS NULL THEN

RAISE EXCEPTION 'Only head circumference parameter can be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

PERFORM 1

FROM Head\_accessory\_categories

WHERE

id = p\_category\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Head accessory category with id % does not exist', p\_category\_id;

END IF;

IF p\_head\_accessory\_head\_circumference IS NOT NULL AND p\_head\_accessory\_head\_circumference <= 0 THEN

RAISE EXCEPTION 'Head circumference must be greater than 0';

END IF;

IF LENGTH(p\_head\_accessory\_name) > 30 OR LENGTH(p\_head\_accessory\_name) < 1 THEN

RAISE EXCEPTION 'Head accessory name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_head\_accessory\_name;

IF FOUND THEN

RAISE EXCEPTION 'Head accessory with name % already exist', p\_head\_accessory\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_head\_accessory\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Head\_accessories (costume\_item\_id, category\_id, head\_circumference)

VALUES (i\_id, p\_category\_id, p\_head\_accessory\_head\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_caftan

**Opis**:

* Pozwala dodać kaftan, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_caftan(

p\_caftan\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_caftan\_length SMALLINT,

p\_caftan\_min\_waist\_circumference SMALLINT,

p\_caftan\_max\_waist\_circumference SMALLINT,

p\_caftan\_min\_chest\_circumference SMALLINT,

p\_caftan\_max\_chest\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_caftan\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_caftan\_length IS NULL OR

p\_caftan\_min\_waist\_circumference IS NULL OR

p\_caftan\_max\_waist\_circumference IS NULL OR

p\_caftan\_min\_chest\_circumference IS NULL OR

p\_caftan\_max\_chest\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_caftan\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_caftan\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_caftan\_max\_waist\_circumference < p\_caftan\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF p\_caftan\_min\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Min chest circumference must be greater than 0';

END IF;

IF p\_caftan\_max\_chest\_circumference < p\_caftan\_min\_chest\_circumference THEN

RAISE EXCEPTION 'Max chest circumference must be greater or equal than min chest circumference';

END IF;

IF LENGTH(p\_caftan\_name) > 30 OR LENGTH(p\_caftan\_name) < 1 THEN

RAISE EXCEPTION 'Caftan name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_caftan\_name;

IF FOUND THEN

RAISE EXCEPTION 'Caftan with name % already exist', p\_caftan\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_caftan\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Caftans (costume\_item\_id, length, min\_waist\_circumference, max\_waist\_circumference,

min\_chest\_circumference, max\_chest\_circumference)

VALUES (i\_id, p\_caftan\_length, p\_caftan\_min\_waist\_circumference, p\_caftan\_max\_waist\_circumference,

p\_caftan\_min\_chest\_circumference, p\_caftan\_max\_chest\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_petticoat

**Opis**:

* Pozwala dodać halkę, która spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_petticoat(

p\_petticoat\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_petticoat\_length SMALLINT,

p\_petticoat\_min\_waist\_circumference SMALLINT,

p\_petticoat\_max\_waist\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_petticoat\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_petticoat\_length IS NULL OR

p\_petticoat\_min\_waist\_circumference IS NULL OR

p\_petticoat\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_petticoat\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_petticoat\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_petticoat\_max\_waist\_circumference < p\_petticoat\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_petticoat\_name) > 30 OR LENGTH(p\_petticoat\_name) < 1 THEN

RAISE EXCEPTION 'Petticoat name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_petticoat\_name;

IF FOUND THEN

RAISE EXCEPTION 'Petticoat with name % already exist', p\_petticoat\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_petticoat\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Petticoats (costume\_item\_id, length, min\_waist\_circumference, max\_waist\_circumference)

VALUES (i\_id, p\_petticoat\_length, p\_petticoat\_min\_waist\_circumference, p\_petticoat\_max\_waist\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_corset

**Opis**:

* Pozwala dodać gorset, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_corset(

p\_corset\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_corset\_length SMALLINT,

p\_corset\_min\_waist\_circumference SMALLINT,

p\_corset\_max\_waist\_circumference SMALLINT,

p\_corset\_min\_chest\_circumference SMALLINT,

p\_corset\_max\_chest\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_corset\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_corset\_length IS NULL OR

p\_corset\_min\_waist\_circumference IS NULL OR

p\_corset\_max\_waist\_circumference IS NULL OR

p\_corset\_min\_chest\_circumference IS NULL OR

p\_corset\_max\_chest\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF NOT FOUND THEN

RAISE EXCEPTION 'Location with id % does not exist', p\_location\_id;

END IF;

IF p\_corset\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_corset\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_corset\_max\_waist\_circumference < p\_corset\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF p\_corset\_min\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Min chest circumference must be greater than 0';

END IF;

IF p\_corset\_max\_chest\_circumference < p\_corset\_min\_chest\_circumference THEN

RAISE EXCEPTION 'Max chest circumference must be greater or equal than min chest circumference';

END IF;

IF LENGTH(p\_corset\_name) > 30 OR LENGTH(p\_corset\_name) < 1 THEN

RAISE EXCEPTION 'Corset name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_corset\_name;

IF FOUND THEN

RAISE EXCEPTION 'Corset with name % already exist', p\_corset\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_corset\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Corsets (costume\_item\_id, length, min\_waist\_circumference, max\_waist\_circumference,

min\_chest\_circumference, max\_chest\_circumference)

VALUES (i\_id, p\_corset\_length, p\_corset\_min\_waist\_circumference, p\_corset\_max\_waist\_circumference,

p\_corset\_min\_chest\_circumference, p\_corset\_max\_chest\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_skirt

**Opis**:

* Pozwala dodać spódnicę, która spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_skirt(

p\_skirt\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_skirt\_length SMALLINT,

p\_skirt\_min\_waist\_circumference SMALLINT,

p\_skirt\_max\_waist\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_skirt\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_skirt\_length IS NULL OR

p\_skirt\_min\_waist\_circumference IS NULL OR

p\_skirt\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_skirt\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_skirt\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_skirt\_max\_waist\_circumference < p\_skirt\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_skirt\_name) > 30 OR LENGTH(p\_skirt\_name) < 1 THEN

RAISE EXCEPTION 'Skirt name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_skirt\_name;

IF FOUND THEN

RAISE EXCEPTION 'Skirt with name % already exist', p\_skirt\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_skirt\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Skirts (costume\_item\_id, length, min\_waist\_circumference, max\_waist\_circumference)

VALUES (i\_id, p\_skirt\_length, p\_skirt\_min\_waist\_circumference, p\_skirt\_max\_waist\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_belt

**Opis**:

* Pozwala dodać pas, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_belt(

p\_belt\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_belt\_min\_waist\_circumference SMALLINT,

p\_belt\_max\_waist\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_belt\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_belt\_min\_waist\_circumference IS NULL OR

p\_belt\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_belt\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_belt\_max\_waist\_circumference < p\_belt\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_belt\_name) > 30 OR LENGTH(p\_belt\_name) < 1 THEN

RAISE EXCEPTION 'Belt name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_belt\_name;

IF FOUND THEN

RAISE EXCEPTION 'Belt with name % already exist', p\_belt\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_belt\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Belts (costume\_item\_id, min\_waist\_circumference, max\_waist\_circumference)

VALUES (i\_id, p\_belt\_min\_waist\_circumference, p\_belt\_max\_waist\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_shirt

**Opis**:

* Pozwala dodać koszulę, która spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_shirt(

p\_shirt\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_shirt\_length SMALLINT,

p\_shirt\_arm\_length SMALLINT,

p\_shirt\_min\_waist\_circumference SMALLINT,

p\_shirt\_max\_waist\_circumference SMALLINT,

p\_shirt\_min\_chest\_circumference SMALLINT,

p\_shirt\_max\_chest\_circumference SMALLINT,

p\_shirt\_min\_neck\_circumference SMALLINT,

p\_shirt\_max\_neck\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_shirt\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_shirt\_length IS NULL OR

p\_shirt\_arm\_length IS NULL OR

p\_shirt\_min\_waist\_circumference IS NULL OR

p\_shirt\_max\_waist\_circumference IS NULL OR

p\_shirt\_min\_chest\_circumference IS NULL OR

p\_shirt\_max\_chest\_circumference IS NULL OR

p\_shirt\_min\_neck\_circumference IS NULL OR

p\_shirt\_max\_neck\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_shirt\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_shirt\_arm\_length <= 0 THEN

RAISE EXCEPTION 'Arm length must be greater than 0';

END IF;

IF p\_shirt\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_shirt\_max\_waist\_circumference < p\_shirt\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF p\_shirt\_min\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Min chest circumference must be greater than 0';

END IF;

IF p\_shirt\_max\_chest\_circumference < p\_shirt\_min\_chest\_circumference THEN

RAISE EXCEPTION 'Max chest circumference must be greater or equal than min chest circumference';

END IF;

IF p\_shirt\_min\_neck\_circumference <= 0 THEN

RAISE EXCEPTION 'Min neck circumference must be greater than 0';

END IF;

IF p\_shirt\_max\_neck\_circumference < p\_shirt\_min\_neck\_circumference THEN

RAISE EXCEPTION 'Max neck circumference must be greater or equal than min neck circumference';

END IF;

IF LENGTH(p\_shirt\_name) > 30 OR LENGTH(p\_shirt\_name) < 1 THEN

RAISE EXCEPTION 'Shirt name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_shirt\_name;

IF FOUND THEN

RAISE EXCEPTION 'Shirt with name % already exist', p\_shirt\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_shirt\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Shirts (costume\_item\_id, length, arm\_length, min\_waist\_circumference, max\_waist\_circumference,

min\_chest\_circumference, max\_chest\_circumference, min\_neck\_circumference, max\_neck\_circumference)

VALUES (i\_id, p\_shirt\_length, p\_shirt\_arm\_length, p\_shirt\_min\_waist\_circumference, p\_shirt\_max\_waist\_circumference,

p\_shirt\_min\_chest\_circumference, p\_shirt\_max\_chest\_circumference, p\_shirt\_min\_neck\_circumference, p\_shirt\_max\_neck\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_pants

**Opis**:

* Pozwala dodać spodnie, które spełniają warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_pants(

p\_pants\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_pants\_length SMALLINT,

p\_pants\_min\_waist\_circumference SMALLINT,

p\_pants\_max\_waist\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_pants\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_pants\_length IS NULL OR

p\_pants\_min\_waist\_circumference IS NULL OR

p\_pants\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_pants\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_pants\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_pants\_max\_waist\_circumference < p\_pants\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_pants\_name) > 30 OR LENGTH(p\_pants\_name) < 1 THEN

RAISE EXCEPTION 'Pants name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_pants\_name;

IF FOUND THEN

RAISE EXCEPTION 'Pants with name % already exist', p\_pants\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_pants\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Pants (costume\_item\_id, length, min\_waist\_circumference, max\_waist\_circumference)

VALUES (i\_id, p\_pants\_length, p\_pants\_min\_waist\_circumference, p\_pants\_max\_waist\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_boots

**Opis**:

* Pozwala dodać buty, które spełniają warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_boots(

p\_boots\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_boots\_shoe\_size FLOAT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_boots\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_boots\_shoe\_size IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_boots\_shoe\_size <= 0 THEN

RAISE EXCEPTION 'Shoe size must be greater than 0';

END IF;

IF LENGTH(p\_boots\_name) > 30 OR LENGTH(p\_boots\_name) < 1 THEN

RAISE EXCEPTION 'Boots name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_boots\_name;

IF FOUND THEN

RAISE EXCEPTION 'Boots with name % already exist', p\_boots\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_boots\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Boots (costume\_item\_id, shoe\_size)

VALUES (i\_id, p\_boots\_shoe\_size);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_neck\_accessory

**Opis:**

* Pozwala dodać akcesorium na szyję, które spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE add\_neck\_accessory(

p\_neck\_accessory\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_neck\_accessory\_min\_neck\_circumference SMALLINT,

p\_neck\_accessory\_max\_neck\_circumference SMALLINT

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_neck\_accessory\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_neck\_accessory\_min\_neck\_circumference IS NULL OR

p\_neck\_accessory\_max\_neck\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_neck\_accessory\_min\_neck\_circumference <= 0 THEN

RAISE EXCEPTION 'Min neck circumference must be greater than 0';

END IF;

IF p\_neck\_accessory\_max\_neck\_circumference < p\_neck\_accessory\_min\_neck\_circumference THEN

RAISE EXCEPTION 'Max neck circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_neck\_accessory\_name) > 30 OR LENGTH(p\_neck\_accessory\_name) < 1 THEN

RAISE EXCEPTION 'Neck accessory name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

name = p\_neck\_accessory\_name;

IF FOUND THEN

RAISE EXCEPTION 'Neck accessory with name % already exist', p\_neck\_accessory\_name;

END IF;

BEGIN

INSERT INTO Costumes\_items (name, collection\_id, gender\_id, color\_id, location\_id)

VALUES (p\_neck\_accessory\_name, p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) RETURNING id INTO i\_id;

INSERT INTO Neck\_accessories (costume\_item\_id, min\_neck\_circumference, max\_neck\_circumference)

VALUES (i\_id, p\_neck\_accessory\_min\_neck\_circumference, p\_neck\_accessory\_max\_neck\_circumference);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_apron

**Opis:**

* Pozwala edytować fartuszek, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_apron(

p\_apron\_id INT,

p\_apron\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_apron\_length SMALLINT,

p\_pattern\_id SMALLINT

) AS $$

BEGIN

IF p\_apron\_id IS NULL OR

p\_apron\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_apron\_length IS NULL OR

p\_pattern\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_apron\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF LENGTH(p\_apron\_name) > 30 OR LENGTH(p\_apron\_name) < 1 THEN

RAISE EXCEPTION 'Apron name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_apron\_name

AND id <> p\_apron\_id;

IF FOUND THEN

RAISE EXCEPTION 'Apron with name % already exist', p\_apron\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_apron\_id

FOR UPDATE;

PERFORM 1

FROM Aprons

WHERE

costume\_item\_id = p\_apron\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_apron\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_apron\_id;

UPDATE Aprons

SET length = p\_apron\_length,

pattern\_id = p\_pattern\_id

WHERE costume\_item\_id = p\_apron\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_head\_accessory

**Opis:**

* Pozwala edytować akcesorium na głowę, które spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_head\_accessory(

p\_head\_accessory\_id INT,

p\_head\_accessory\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_head\_accessory\_head\_circumference SMALLINT,

p\_category\_id SMALLINT

) AS $$

BEGIN

IF p\_head\_accessory\_id IS NULL OR

p\_head\_accessory\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_category\_id IS NULL THEN

RAISE EXCEPTION 'Only head circumference parameter can be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

PERFORM 1

FROM Head\_accessory\_categories

WHERE id = p\_category\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Head accessory category with id % does not exist', p\_category\_id;

END IF;

IF p\_head\_accessory\_head\_circumference IS NOT NULL AND p\_head\_accessory\_head\_circumference <= 0 THEN

RAISE EXCEPTION 'Head circumference must be greater than 0';

END IF;

IF LENGTH(p\_head\_accessory\_name) > 30 OR LENGTH(p\_head\_accessory\_name) < 1 THEN

RAISE EXCEPTION 'Head accessory name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_head\_accessory\_name

AND id <> p\_head\_accessory\_id;

IF FOUND THEN

RAISE EXCEPTION 'Head accessory with name % already exists', p\_head\_accessory\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_head\_accessory\_id

FOR UPDATE;

PERFORM 1

FROM Head\_accessories

WHERE costume\_item\_id = p\_head\_accessory\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_head\_accessory\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_head\_accessory\_id;

UPDATE Head\_accessories

SET category\_id = p\_category\_id,

head\_circumference = p\_head\_accessory\_head\_circumference

WHERE costume\_item\_id = p\_head\_accessory\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_caftan

**Opis:**

* Pozwala edytować kaftan, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_caftan(

p\_caftan\_id INT,

p\_caftan\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_caftan\_length SMALLINT,

p\_caftan\_min\_waist\_circumference SMALLINT,

p\_caftan\_max\_waist\_circumference SMALLINT,

p\_caftan\_min\_chest\_circumference SMALLINT,

p\_caftan\_max\_chest\_circumference SMALLINT

) AS $$

BEGIN

IF p\_caftan\_id IS NULL OR

p\_caftan\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_caftan\_length IS NULL OR

p\_caftan\_min\_waist\_circumference IS NULL OR

p\_caftan\_max\_waist\_circumference IS NULL OR

p\_caftan\_min\_chest\_circumference IS NULL OR

p\_caftan\_max\_chest\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_caftan\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_caftan\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_caftan\_max\_waist\_circumference < p\_caftan\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF p\_caftan\_min\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Min chest circumference must be greater than 0';

END IF;

IF p\_caftan\_max\_chest\_circumference < p\_caftan\_min\_chest\_circumference THEN

RAISE EXCEPTION 'Max chest circumference must be greater or equal than min chest circumference';

END IF;

IF LENGTH(p\_caftan\_name) > 30 OR LENGTH(p\_caftan\_name) < 1 THEN

RAISE EXCEPTION 'Caftan name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_caftan\_name

AND id <> p\_caftan\_id;

IF FOUND THEN

RAISE EXCEPTION 'Caftan with name % already exists', p\_caftan\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_caftan\_id

FOR UPDATE;

PERFORM 1

FROM Caftans

WHERE costume\_item\_id = p\_caftan\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_caftan\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_caftan\_id;

UPDATE Caftans

SET length = p\_caftan\_length,

min\_waist\_circumference = p\_caftan\_min\_waist\_circumference,

max\_waist\_circumference = p\_caftan\_max\_waist\_circumference,

min\_chest\_circumference = p\_caftan\_min\_chest\_circumference,

max\_chest\_circumference = p\_caftan\_max\_chest\_circumference

WHERE costume\_item\_id = p\_caftan\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_petticoat

**Opis:**

* Pozwala edytować halkę, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_petticoat(

p\_petticoat\_id INT,

p\_petticoat\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_petticoat\_length SMALLINT,

p\_petticoat\_min\_waist\_circumference SMALLINT,

p\_petticoat\_max\_waist\_circumference SMALLINT

) AS $$

BEGIN

IF p\_petticoat\_id IS NULL OR

p\_petticoat\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_petticoat\_length IS NULL OR

p\_petticoat\_min\_waist\_circumference IS NULL OR

p\_petticoat\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_petticoat\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_petticoat\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_petticoat\_max\_waist\_circumference < p\_petticoat\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_petticoat\_name) > 30 OR LENGTH(p\_petticoat\_name) < 1 THEN

RAISE EXCEPTION 'Petticoat name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_petticoat\_name

AND id <> p\_petticoat\_id;

IF FOUND THEN

RAISE EXCEPTION 'Petticoat with name % already exists', p\_petticoat\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_petticoat\_id

FOR UPDATE;

PERFORM 1

FROM Petticoats

WHERE costume\_item\_id = p\_petticoat\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_petticoat\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_petticoat\_id;

UPDATE Petticoats

SET length = p\_petticoat\_length,

min\_waist\_circumference = p\_petticoat\_min\_waist\_circumference,

max\_waist\_circumference = p\_petticoat\_max\_waist\_circumference

WHERE costume\_item\_id = p\_petticoat\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_corset

**Opis:**

* Pozwala edytować gorset, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_corset(

p\_corset\_id INT,

p\_corset\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_corset\_length SMALLINT,

p\_corset\_min\_waist\_circumference SMALLINT,

p\_corset\_max\_waist\_circumference SMALLINT,

p\_corset\_min\_chest\_circumference SMALLINT,

p\_corset\_max\_chest\_circumference SMALLINT

) AS $$

BEGIN

IF p\_corset\_id IS NULL OR

p\_corset\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_corset\_length IS NULL OR

p\_corset\_min\_waist\_circumference IS NULL OR

p\_corset\_max\_waist\_circumference IS NULL OR

p\_corset\_min\_chest\_circumference IS NULL OR

p\_corset\_max\_chest\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_corset\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_corset\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_corset\_max\_waist\_circumference < p\_corset\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF p\_corset\_min\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Min chest circumference must be greater than 0';

END IF;

IF p\_corset\_max\_chest\_circumference < p\_corset\_min\_chest\_circumference THEN

RAISE EXCEPTION 'Max chest circumference must be greater or equal than min chest circumference';

END IF;

IF LENGTH(p\_corset\_name) > 30 OR LENGTH(p\_corset\_name) < 1 THEN

RAISE EXCEPTION 'Corset name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_corset\_name

AND id <> p\_corset\_id;

IF FOUND THEN

RAISE EXCEPTION 'Corset with name % already exists', p\_corset\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_corset\_id

FOR UPDATE;

PERFORM 1

FROM Corsets

WHERE costume\_item\_id = p\_corset\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_corset\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_corset\_id;

UPDATE Corsets

SET length = p\_corset\_length,

min\_waist\_circumference = p\_corset\_min\_waist\_circumference,

max\_waist\_circumference = p\_corset\_max\_waist\_circumference,

min\_chest\_circumference = p\_corset\_min\_chest\_circumference,

max\_chest\_circumference = p\_corset\_max\_chest\_circumference

WHERE costume\_item\_id = p\_corset\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_skirt

**Opis:**

* Pozwala edytować spódnicę, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_skirt(

p\_skirt\_id INT,

p\_skirt\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_skirt\_length SMALLINT,

p\_skirt\_min\_waist\_circumference SMALLINT,

p\_skirt\_max\_waist\_circumference SMALLINT

) AS $$

BEGIN

IF p\_skirt\_id IS NULL OR

p\_skirt\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_skirt\_length IS NULL OR

p\_skirt\_min\_waist\_circumference IS NULL OR

p\_skirt\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_skirt\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_skirt\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_skirt\_max\_waist\_circumference < p\_skirt\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_skirt\_name) > 30 OR LENGTH(p\_skirt\_name) < 1 THEN

RAISE EXCEPTION 'Skirt name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_skirt\_name

AND id <> p\_skirt\_id;

IF FOUND THEN

RAISE EXCEPTION 'Skirt with name % already exists', p\_skirt\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_skirt\_id

FOR UPDATE;

PERFORM 1

FROM Skirts

WHERE costume\_item\_id = p\_skirt\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_skirt\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_skirt\_id;

UPDATE Skirts

SET length = p\_skirt\_length,

min\_waist\_circumference = p\_skirt\_min\_waist\_circumference,

max\_waist\_circumference = p\_skirt\_max\_waist\_circumference

WHERE costume\_item\_id = p\_skirt\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_belt

**Opis:**

* Pozwala edytować pas, który spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_belt(

p\_belt\_id INT,

p\_belt\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_belt\_min\_waist\_circumference SMALLINT,

p\_belt\_max\_waist\_circumference SMALLINT

) AS $$

BEGIN

IF p\_belt\_id IS NULL OR

p\_belt\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_belt\_min\_waist\_circumference IS NULL OR

p\_belt\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_belt\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_belt\_max\_waist\_circumference < p\_belt\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_belt\_name) > 30 OR LENGTH(p\_belt\_name) < 1 THEN

RAISE EXCEPTION 'Belt name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_belt\_name

AND id <> p\_belt\_id;

IF FOUND THEN

RAISE EXCEPTION 'Belt with name % already exists', p\_belt\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_belt\_id

FOR UPDATE;

PERFORM 1

FROM Belts

WHERE costume\_item\_id = p\_belt\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_belt\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_belt\_id;

UPDATE Belts

SET min\_waist\_circumference = p\_belt\_min\_waist\_circumference,

max\_waist\_circumference = p\_belt\_max\_waist\_circumference

WHERE costume\_item\_id = p\_belt\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_shirt

**Opis:**

* Pozwala edytować koszulę, która spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_shirt(

p\_shirt\_id INT,

p\_shirt\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_shirt\_length SMALLINT,

p\_shirt\_arm\_length SMALLINT,

p\_shirt\_min\_waist\_circumference SMALLINT,

p\_shirt\_max\_waist\_circumference SMALLINT,

p\_shirt\_min\_chest\_circumference SMALLINT,

p\_shirt\_max\_chest\_circumference SMALLINT,

p\_shirt\_min\_neck\_circumference SMALLINT,

p\_shirt\_max\_neck\_circumference SMALLINT

) AS $$

BEGIN

IF p\_shirt\_id IS NULL OR

p\_shirt\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_shirt\_length IS NULL OR

p\_shirt\_arm\_length IS NULL OR

p\_shirt\_min\_waist\_circumference IS NULL OR

p\_shirt\_max\_waist\_circumference IS NULL OR

p\_shirt\_min\_chest\_circumference IS NULL OR

p\_shirt\_max\_chest\_circumference IS NULL OR

p\_shirt\_min\_neck\_circumference IS NULL OR

p\_shirt\_max\_neck\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_shirt\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_shirt\_arm\_length <= 0 THEN

RAISE EXCEPTION 'Arm length must be greater than 0';

END IF;

IF p\_shirt\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_shirt\_max\_waist\_circumference < p\_shirt\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF p\_shirt\_min\_chest\_circumference <= 0 THEN

RAISE EXCEPTION 'Min chest circumference must be greater than 0';

END IF;

IF p\_shirt\_max\_chest\_circumference < p\_shirt\_min\_chest\_circumference THEN

RAISE EXCEPTION 'Max chest circumference must be greater or equal than min chest circumference';

END IF;

IF p\_shirt\_min\_neck\_circumference <= 0 THEN

RAISE EXCEPTION 'Min neck circumference must be greater than 0';

END IF;

IF p\_shirt\_max\_neck\_circumference < p\_shirt\_min\_neck\_circumference THEN

RAISE EXCEPTION 'Max neck circumference must be greater or equal than min neck circumference';

END IF;

IF LENGTH(p\_shirt\_name) > 30 OR LENGTH(p\_shirt\_name) < 1 THEN

RAISE EXCEPTION 'Shirt name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_shirt\_name

AND id <> p\_shirt\_id;

IF FOUND THEN

RAISE EXCEPTION 'Shirt with name % already exists', p\_shirt\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_shirt\_id

FOR UPDATE;

PERFORM 1

FROM Shirts

WHERE costume\_item\_id = p\_shirt\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_shirt\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_shirt\_id;

UPDATE Shirts

SET length = p\_shirt\_length,

arm\_length = p\_shirt\_arm\_length,

min\_waist\_circumference = p\_shirt\_min\_waist\_circumference,

max\_waist\_circumference = p\_shirt\_max\_waist\_circumference,

min\_chest\_circumference = p\_shirt\_min\_chest\_circumference,

max\_chest\_circumference = p\_shirt\_max\_chest\_circumference,

min\_neck\_circumference = p\_shirt\_min\_neck\_circumference,

max\_neck\_circumference = p\_shirt\_max\_neck\_circumference

WHERE costume\_item\_id = p\_shirt\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_pants

**Opis:**

* Pozwala edytować spodnie, które spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_pants(

p\_pants\_id INT,

p\_pants\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_pants\_length SMALLINT,

p\_pants\_min\_waist\_circumference SMALLINT,

p\_pants\_max\_waist\_circumference SMALLINT

) AS $$

BEGIN

IF p\_pants\_id IS NULL OR

p\_pants\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_pants\_length IS NULL OR

p\_pants\_min\_waist\_circumference IS NULL OR

p\_pants\_max\_waist\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_pants\_length <= 0 THEN

RAISE EXCEPTION 'Length must be greater than 0';

END IF;

IF p\_pants\_min\_waist\_circumference <= 0 THEN

RAISE EXCEPTION 'Min waist circumference must be greater than 0';

END IF;

IF p\_pants\_max\_waist\_circumference < p\_pants\_min\_waist\_circumference THEN

RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

END IF;

IF LENGTH(p\_pants\_name) > 30 OR LENGTH(p\_pants\_name) < 1 THEN

RAISE EXCEPTION 'Pants name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_pants\_name

AND id <> p\_pants\_id;

IF FOUND THEN

RAISE EXCEPTION 'Pants with name % already exists', p\_pants\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_pants\_id

FOR UPDATE;

PERFORM 1

FROM Pants

WHERE costume\_item\_id = p\_pants\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_pants\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_pants\_id;

UPDATE Pants

SET length = p\_pants\_length,

min\_waist\_circumference = p\_pants\_min\_waist\_circumference,

max\_waist\_circumference = p\_pants\_max\_waist\_circumference

WHERE costume\_item\_id = p\_pants\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_boots

**Opis:**

* Pozwala edytować buty, które spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_boots(

p\_boots\_id INT,

p\_boots\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_boots\_shoe\_size FLOAT

) AS $$

BEGIN

IF p\_boots\_id IS NULL OR

p\_boots\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_boots\_shoe\_size IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_boots\_shoe\_size <= 0 THEN

RAISE EXCEPTION 'Shoe size must be greater than 0';

END IF;

IF LENGTH(p\_boots\_name) > 30 OR LENGTH(p\_boots\_name) < 1 THEN

RAISE EXCEPTION 'Boots name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_boots\_name

AND id <> p\_boots\_id;

IF FOUND THEN

RAISE EXCEPTION 'Boots with name % already exists', p\_boots\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_boots\_id

FOR UPDATE;

PERFORM 1

FROM Boots

WHERE costume\_item\_id = p\_boots\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_boots\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_boots\_id;

UPDATE Boots

SET shoe\_size = p\_boots\_shoe\_size

WHERE costume\_item\_id = p\_boots\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_neck\_accessory

**Opis:**

* Pozwala edytować akcesorium na szyję, które spełnia warunki spójności bazy danych.

**Implementacja:**

CREATE OR REPLACE PROCEDURE update\_neck\_accessory(

p\_neck\_accessory\_id INT,

p\_neck\_accessory\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_color\_id SMALLINT,

p\_location\_id SMALLINT,

p\_neck\_accessory\_min\_neck\_circumference SMALLINT,

p\_neck\_accessory\_max\_neck\_circumference SMALLINT

) AS $$

BEGIN

IF p\_neck\_accessory\_id IS NULL OR

p\_neck\_accessory\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL OR

p\_color\_id IS NULL OR

p\_location\_id IS NULL OR

p\_neck\_accessory\_min\_neck\_circumference IS NULL OR

p\_neck\_accessory\_max\_neck\_circumference IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

IF check\_if\_error\_in\_costume\_item\_common\_part(p\_collection\_id, p\_gender\_id, p\_color\_id, p\_location\_id) THEN

RAISE EXCEPTION 'Something wrong in costume item common part';

END IF;

IF p\_neck\_accessory\_min\_neck\_circumference <= 0 THEN

RAISE EXCEPTION 'Min neck circumference must be greater than 0';

END IF;

IF p\_neck\_accessory\_max\_neck\_circumference < p\_neck\_accessory\_min\_neck\_circumference THEN

RAISE EXCEPTION 'Max neck circumference must be greater or equal than min neck circumference';

END IF;

IF LENGTH(p\_neck\_accessory\_name) > 30 OR LENGTH(p\_neck\_accessory\_name) < 1 THEN

RAISE EXCEPTION 'Neck accessory name can have between 1 and 30 characters';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE name = p\_neck\_accessory\_name

AND id <> p\_neck\_accessory\_id;

IF FOUND THEN

RAISE EXCEPTION 'Neck accessory with name % already exists', p\_neck\_accessory\_name;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE id = p\_neck\_accessory\_id

FOR UPDATE;

PERFORM 1

FROM Neck\_accessories

WHERE costume\_item\_id = p\_neck\_accessory\_id

FOR UPDATE;

UPDATE Costumes\_items

SET name = p\_neck\_accessory\_name,

collection\_id = p\_collection\_id,

gender\_id = p\_gender\_id,

color\_id = p\_color\_id,

location\_id = p\_location\_id

WHERE id = p\_neck\_accessory\_id;

UPDATE Neck\_accessories

SET min\_neck\_circumference = p\_neck\_accessory\_min\_neck\_circumference,

max\_neck\_circumference = p\_neck\_accessory\_max\_neck\_circumference

WHERE costume\_item\_id = p\_neck\_accessory\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_costume

**Opis:**

* Pozwala dodać strój, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_costume(

p\_costume\_name VARCHAR(30),

p\_collection\_id SMALLINT,

p\_gender\_id SMALLINT,

p\_apron\_id INTEGER,

p\_caftan\_id INTEGER,

p\_petticoat\_id INTEGER,

p\_corset\_id INTEGER,

p\_skirt\_id INTEGER,

p\_belt\_id INTEGER,

p\_shirt\_id INTEGER,

p\_pants\_id INTEGER,

p\_boots\_id INTEGER,

p\_neck\_accessory\_id INTEGER,

p\_head\_accessory\_id INTEGER

) AS $$

BEGIN

IF p\_costume\_name IS NULL OR

p\_collection\_id IS NULL OR

p\_gender\_id IS NULL THEN

RAISE EXCEPTION 'Costume name, collection id and gender id cannot be NULL';

END IF;

PERFORM 1

FROM Collections

WHERE

id = p\_collection\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Collection with id % does not exist', p\_collection\_id;

END IF;

IF p\_gender\_id NOT IN (1, 2, 3) THEN

RAISE EXCEPTION 'Gender with id 1 (male) or 2 (female) or 3 (bigender) can be selected';

END IF;

PERFORM 1

FROM Genders

WHERE

id = p\_gender\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Gender with id % does not exist', p\_gender\_id;

END IF;

IF p\_apron\_id IS NOT NULL THEN

PERFORM 1

FROM Aprons

WHERE

costume\_item\_id = p\_apron\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Apron with id % does not exist', p\_apron\_id;

END IF;

END IF;

IF p\_caftan\_id IS NOT NULL THEN

PERFORM 1

FROM Caftans

WHERE

costume\_item\_id = p\_caftan\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Caftan with id % does not exist', p\_caftan\_id;

END IF;

END IF;

IF p\_petticoat\_id IS NOT NULL THEN

PERFORM 1

FROM Petticoats

WHERE

costume\_item\_id = p\_petticoat\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Peticoat with id % does not exist', p\_petticoat\_id;

END IF;

END IF;

IF p\_corset\_id IS NOT NULL THEN

PERFORM 1

FROM Corsets

WHERE

costume\_item\_id = p\_corset\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Corset with id % does not exist', p\_corset\_id;

END IF;

END IF;

IF p\_skirt\_id IS NOT NULL THEN

PERFORM 1

FROM Skirts

WHERE

costume\_item\_id = p\_skirt\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Skirt with id % does not exist', p\_skirt\_id;

END IF;

END IF;

IF p\_belt\_id IS NOT NULL THEN

PERFORM 1

FROM Belts

WHERE

costume\_item\_id = p\_belt\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Belt with id % does not exist', p\_belt\_id;

END IF;

END IF;

IF p\_shirt\_id IS NOT NULL THEN

PERFORM 1

FROM Shirts

WHERE

costume\_item\_id = p\_shirt\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Shirt with id % does not exist', p\_shirt\_id;

END IF;

END IF;

IF p\_pants\_id IS NOT NULL THEN

PERFORM 1

FROM Pants

WHERE

costume\_item\_id = p\_pants\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Pants with id % does not exist', p\_pants\_id;

END IF;

END IF;

IF p\_boots\_id IS NOT NULL THEN

PERFORM 1

FROM Boots

WHERE

costume\_item\_id = p\_boots\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Boots with id % does not exist', p\_boots\_id;

END IF;

END IF;

IF p\_neck\_accessory\_id IS NOT NULL THEN

PERFORM 1

FROM Neck\_accessories

WHERE

costume\_item\_id = p\_neck\_accessory\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Neck accessory with id % does not exist', p\_neck\_accessory\_id;

END IF;

END IF;

IF p\_head\_accessory\_id IS NOT NULL THEN

PERFORM 1

FROM Head\_accessories

WHERE

costume\_item\_id = p\_head\_accessory\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Head accessory with id % does not exist', p\_head\_accessory\_id;

END IF;

END IF;

IF LENGTH(p\_costume\_name) > 30 THEN

RAISE EXCEPTION 'Costume name exceeded 30 characters';

END IF;

PERFORM 1

FROM Costumes

WHERE

name = p\_costume\_name;

IF FOUND THEN

RAISE EXCEPTION 'Costume with name % already exist', p\_costume\_name;

END IF;

IF check\_costume\_inconsistency(p\_collection\_id, p\_gender\_id, p\_apron\_id, p\_caftan\_id, p\_petticoat\_id,

p\_corset\_id, p\_shirt\_id, p\_belt\_id, p\_shirt\_id, p\_pants\_id, p\_boots\_id, p\_neck\_accessory\_id,

p\_head\_accessory\_id) THEN

RAISE EXCEPTION 'Costume is inconsistency';

END IF;

BEGIN

INSERT INTO Costumes (name, collection\_id, gender\_id, apron\_id, caftan\_id, petticoat\_id, corset\_id, skirt\_id,

belt\_id, shirt\_id, pants\_id, boots\_id, neck\_accessory\_id, head\_accessory\_id)

VALUES (p\_costume\_name, p\_collection\_id, p\_gender\_id, p\_apron\_id, p\_caftan\_id, p\_petticoat\_id, p\_corset\_id,

p\_skirt\_id, p\_belt\_id, p\_shirt\_id, p\_pants\_id, p\_boots\_id, p\_neck\_accessory\_id, p\_head\_accessory\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_rental\_costume\_item\_request

**Opis**:

* Pozwala dodać żądanie o wypożyczenie elementu stroju, które spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_rental\_costume\_item\_request(

p\_requester\_user\_id INTEGER,

p\_costume\_item\_id INTEGER

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_requester\_user\_id IS NULL OR

p\_costume\_item\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_requester\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_requester\_user\_id;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_costume\_item\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item with id % does not exist', p\_costume\_item\_id;

END IF;

PERFORM 1

FROM Rentals

WHERE costume\_item\_id = p\_costume\_item\_id AND date\_of\_return IS NULL;

IF FOUND THEN

RAISE EXCEPTION 'Costume item with id % is already rented', p\_costume\_item\_id;

END IF;

BEGIN

-- 1-> PENDING

INSERT INTO Requests (datetime, requester\_user\_id, state\_id)

VALUES (NOW(), p\_requester\_user\_id, 1) RETURNING id INTO i\_id;

INSERT INTO Rental\_costume\_item\_requests (request\_id, costume\_item\_id, approver\_costumier\_id)

VALUES (i\_id, p\_costume\_item\_id, NULL);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_return\_costume\_item\_request

**Opis:**

* Pozwala dodać żądanie o oddanie stroju, które spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_return\_costume\_item\_request(

p\_requester\_user\_id INTEGER,

p\_costume\_item\_id INTEGER

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_requester\_user\_id IS NULL OR

p\_costume\_item\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_requester\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_requester\_user\_id;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_costume\_item\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item with id % does not exist', p\_costume\_item\_id;

END IF;

PERFORM 1

FROM Rentals

WHERE user\_id = p\_requester\_user\_id AND costume\_item\_id = p\_costume\_item\_id AND date\_of\_return IS NULL;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item is not rented';

END IF;

BEGIN

-- 1-> PENDING

INSERT INTO Requests (datetime, requester\_user\_id, state\_id)

VALUES (NOW(), p\_requester\_user\_id, 1) RETURNING id INTO i\_id;

INSERT INTO Return\_costume\_item\_requests (request\_id, costume\_item\_id, approver\_costumier\_id)

VALUES (i\_id, p\_costume\_item\_id, NULL);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_borrow\_costume\_item\_request

**Opis:**

* Pozwala dodać żądanie o pożyczenie stroju, które spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_borrow\_costume\_item\_request(

p\_requester\_user\_id INTEGER,

p\_costume\_item\_id INTEGER,

p\_approver\_user\_id INTEGER

) AS $$

DECLARE

i\_id INT;

BEGIN

IF p\_requester\_user\_id IS NULL OR

p\_costume\_item\_id IS NULL OR

p\_approver\_user\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_requester\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_requester\_user\_id;

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_approver\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_approver\_user\_id;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_costume\_item\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item with id % does not exist', p\_costume\_item\_id;

END IF;

PERFORM 1

FROM Rentals

WHERE user\_id = p\_approver\_user\_id AND costume\_item\_id = p\_costume\_item\_id AND date\_of\_return IS NULL;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item with id % was not rented by user with id %', p\_costume\_item\_id, p\_approver\_user\_id;

END IF;

IF p\_requester\_user\_id = p\_approver\_user\_id THEN

RAISE EXCEPTION 'Requester user id and approver user id are the same';

END IF;

BEGIN

-- 1-> PENDING

INSERT INTO Requests (datetime, requester\_user\_id, state\_id)

VALUES (NOW(), p\_requester\_user\_id, 1) RETURNING id INTO i\_id;

INSERT INTO Borrow\_costume\_item\_requests (request\_id, costume\_item\_id, approver\_user\_id)

VALUES (i\_id, p\_costume\_item\_id, p\_approver\_user\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_notification

**Opis**:

* Pozwala dodać powiadomienie, które spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_notification(

p\_user\_id INTEGER,

p\_notification\_content TEXT,

p\_due\_to\_request\_id INTEGER

) AS $$

DECLARE

r\_user\_id INTEGER;

BEGIN

IF p\_user\_id IS NULL OR

p\_notification\_content IS NULL OR

p\_due\_to\_request\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_user\_id;

END IF;

IF p\_due\_to\_request\_id IS NOT NULL THEN

PERFORM 1

FROM Requests

WHERE

id = p\_due\_to\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_due\_to\_request\_id;

END IF;

SELECT requester\_user\_id INTO r\_user\_id

FROM Requests

WHERE

id = p\_due\_to\_request\_id;

IF p\_user\_id <> r\_user\_id THEN

RAISE EXCEPTION 'User id and requester user id from request with % are not the same', p\_due\_to\_request\_id;

END IF;

END IF;

IF LENGTH(p\_notification\_content) < 1 THEN

RAISE EXCEPTION 'Notification must be at least 1 character';

END IF;

BEGIN

INSERT INTO Notifications (user\_id, content, datetime, due\_to\_request\_id)

VALUES (p\_user\_id, p\_notification\_content, NOW(), p\_due\_to\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## add\_rental

**Opis**:

* Pozwala dodać wypożyczenie, który spełnia warunki spójności bazy danych.

**Implementacja**:

CREATE OR REPLACE PROCEDURE add\_rental(

p\_user\_id INTEGER,

p\_costume\_item\_id INTEGER,

p\_done\_due\_request\_id INTEGER,

p\_rental\_date\_of\_rental TIMESTAMP

) AS $$

BEGIN

IF p\_user\_id IS NULL OR

p\_costume\_item\_id IS NULL OR

p\_done\_due\_request\_id IS NULL OR

p\_rental\_date\_of\_rental IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Users

WHERE

id = p\_user\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'User with id % does not exist', p\_user\_id;

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_costume\_item\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item with id % does not exist', p\_costume\_item\_id;

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_done\_due\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_done\_due\_request\_id;

END IF;

IF check\_rental\_inconsistency(p\_user\_id, p\_costume\_item\_id, p\_done\_due\_request\_id) THEN

RAISE EXCEPTION 'Rental is inconsistency';

END IF;

PERFORM 1

FROM Rentals

WHERE

costume\_item\_id = p\_costume\_item\_id

AND

date\_of\_return IS NULL;

IF FOUND THEN

RAISE EXCEPTION 'Costume item with id % is already rented', p\_costume\_item\_id;

END IF;

BEGIN

INSERT INTO Rentals (user\_id, costume\_item\_id, done\_due\_request\_id, date\_of\_rental)

VALUES (p\_user\_id, p\_costume\_item\_id, p\_done\_due\_request\_id, p\_rental\_date\_of\_rental);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to insert: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## update\_costume\_item\_location

**Opis**:

* Pozwala zaktualizować lokacje elementu stroju.

**Implementacja**:

CREATE OR REPLACE PROCEDURE update\_costume\_item\_location(

p\_costume\_item\_id INTEGER,

p\_location\_id INTEGER

) AS $$

BEGIN

IF p\_costume\_item\_id IS NULL OR

p\_location\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_costume\_item\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costume item with id % does not exist', p\_costume\_item\_id;

END IF;

PERFORM 1

FROM Locations

WHERE

id = p\_location\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Location with id % does not exist', p\_location\_id;

END IF;

BEGIN

PERFORM 1

FROM Costumes\_items

WHERE

id = p\_costume\_item\_id

FOR UPDATE;

UPDATE Costumes\_items

SET

location\_id = p\_location\_id

WHERE

id = p\_costume\_item\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## delete\_request

**Opis**:

* Pozwala usunąć request, który nie naruszy warunków spójności bazy danych.
* Tylko ze state o id = 1 (PENDING).

**Implementacja**:

CREATE OR REPLACE PROCEDURE delete\_request(

p\_request\_id INTEGER

) AS $$

DECLARE

r\_state\_id INT;

BEGIN

IF p\_request\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

SELECT state\_id INTO r\_state\_id

FROM Requests

WHERE

id = p\_request\_id;

IF r\_state\_id <> 1 THEN

RAISE EXCEPTION 'Request closed - cannot delete';

END IF;

BEGIN

DELETE FROM Requests WHERE id = p\_request\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to delete: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## accept\_rental\_costume\_item\_request

**Opis**:

* Pozwala zaakceptować żądanie wypożyczenia elementu stroju.
* Generuje powiadomienie rozszerzone o przekazany komentarz. (np. gdzie i kiedy można odebrać strój).

**Implementacja**:

CREATE OR REPLACE PROCEDURE accept\_rental\_costume\_item\_request(

p\_request\_id INTEGER,

p\_approver\_costumier\_id INTEGER,

p\_comment TEXT

) AS $$

DECLARE

r\_requester\_user\_id INT;

r\_costume\_item\_id INT;

notification\_content TEXT;

BEGIN

IF p\_request\_id IS NULL OR

p\_approver\_costumier\_id IS NULL OR

p\_comment IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

PERFORM 1

FROM Rental\_costume\_item\_requests

WHERE

request\_id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % is not rental\_costume\_item\_request', p\_request\_id;

END IF;

PERFORM 1

FROM Costumiers

WHERE

user\_id = p\_approver\_costumier\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costumier with id % does not exist', p\_approver\_costumier\_id;

END IF;

IF LENGTH(p\_comment) < 1 THEN

RAISE EXCEPTION 'Comment must be at least 1 character';

END IF;

BEGIN

SELECT requester\_user\_id INTO r\_requester\_user\_id

FROM Requests

WHERE

id = p\_request\_id

FOR UPDATE;

SELECT costume\_item\_id INTO r\_costume\_item\_id

FROM Rental\_costume\_item\_requests

WHERE

request\_id = p\_request\_id

FOR UPDATE;

-- 2-> ACCEPT

UPDATE Requests

SET

state\_id = 2

WHERE

id = p\_request\_id;

UPDATE Rental\_costume\_item\_requests

SET

approver\_costumier\_id = p\_approver\_costumier\_id

WHERE

request\_id = p\_request\_id;

notification\_content := 'Request with id ' || p\_request\_id || ' has been accepted. You can rent costume item with id ' || r\_costume\_item\_id || '. ' || p\_comment;

CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## deny\_rental\_costume\_item\_request

**Opis**:

* Pozwala odrzucić żądanie o wypożyczenie elementu stroju.
* Generuje powiadomienie rozszerzone o przekazany komentarz. (np. dlaczego został odrzucony: jest w naprawie).

**Implementacja**:

CREATE OR REPLACE PROCEDURE deny\_rental\_costume\_item\_request(

p\_request\_id INTEGER,

p\_approver\_costumier\_id INTEGER,

p\_comment TEXT

) AS $$

DECLARE

r\_requester\_user\_id INT;

notification\_content TEXT := 'Request with id ' || p\_request\_id || ' has been denied.' || p\_comment;

BEGIN

IF p\_request\_id IS NULL OR

p\_approver\_costumier\_id IS NULL OR

p\_comment IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

PERFORM 1

FROM Rental\_costume\_item\_requests

WHERE

request\_id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % is not rental\_costume\_item\_request', p\_request\_id;

END IF;

PERFORM 1

FROM Costumiers

WHERE

user\_id = p\_approver\_costumier\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costumier with id % does not exist', p\_approver\_costumier\_id;

END IF;

IF LENGTH(p\_comment) < 1 THEN

RAISE EXCEPTION 'Comment must be at least 1 character';

END IF;

BEGIN

SELECT requester\_user\_id INTO r\_requester\_user\_id

FROM Requests

WHERE

id = p\_request\_id

FOR UPDATE;

PERFORM 1

FROM Rental\_costume\_item\_requests

WHERE

request\_id = p\_request\_id

FOR UPDATE;

-- 3-> DENY

UPDATE Requests

SET

state\_id = 3

WHERE

id = p\_request\_id;

UPDATE Rental\_costume\_item\_requests

SET

approver\_costumier\_id = p\_approver\_costumier\_id

WHERE

request\_id = p\_request\_id;

CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## accept\_return\_costume\_item\_request

**Opis**:

* Pozwala zaakceptować żądanie o oddanie elementu stroju.
* Generuje powiadomienie rozszerzone o przekazany komentarz. (np. gdzie i kiedy można oddać).

**Implementacja**:

CREATE OR REPLACE PROCEDURE accept\_return\_costume\_item\_request(

p\_request\_id INTEGER,

p\_approver\_costumier\_id INTEGER,

p\_comment TEXT

) AS $$

DECLARE

r\_requester\_user\_id INT;

r\_costume\_item\_id INT;

notification\_content TEXT;

BEGIN

IF p\_request\_id IS NULL OR

p\_approver\_costumier\_id IS NULL OR

p\_comment IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

PERFORM 1

FROM Return\_costume\_item\_requests

WHERE

request\_id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % is not return\_costume\_item\_request', p\_request\_id;

END IF;

PERFORM 1

FROM Costumiers

WHERE

user\_id = p\_approver\_costumier\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costumier with id % does not exist', p\_approver\_costumier\_id;

END IF;

IF LENGTH(p\_comment) < 1 THEN

RAISE EXCEPTION 'Comment must be at least 1 character';

END IF;

BEGIN

SELECT requester\_user\_id INTO r\_requester\_user\_id

FROM Requests

WHERE

id = p\_request\_id

FOR UPDATE;

SELECT costume\_item\_id INTO r\_costume\_item\_id

FROM Return\_costume\_item\_requests

WHERE

request\_id = p\_request\_id

FOR UPDATE;

-- 2-> ACCEPT

UPDATE Requests

SET

state\_id = 2

WHERE

id = p\_request\_id;

UPDATE Return\_costume\_item\_requests

SET

approver\_costumier\_id = p\_approver\_costumier\_id

WHERE

request\_id = p\_request\_id;

notification\_content := 'Request with id ' || p\_request\_id || ' has been accepted. You can return costume item with id ' || r\_costume\_item\_id || '. ' || p\_comment;

CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## deny\_return\_costume\_item\_request

**Opis**:

* Pozwala odrzucić żądanie o oddanie elementu stroju.
* Generuje powiadomienie rozszerzone o przekazany komentarz. (np. dlaczego został odrzucony: wakacje).

**Implementacja**:

CREATE OR REPLACE PROCEDURE deny\_return\_costume\_item\_request(

p\_request\_id INTEGER,

p\_approver\_costumier\_id INTEGER,

p\_comment TEXT

) AS $$

DECLARE

r\_requester\_user\_id INT;

notification\_content TEXT := 'Request with id ' || p\_request\_id || ' has been denied.' || p\_comment;

BEGIN

IF p\_request\_id IS NULL OR

p\_approver\_costumier\_id IS NULL OR

p\_comment IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

PERFORM 1

FROM Return\_costume\_item\_requests

WHERE

request\_id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % is not return\_costume\_item\_request', p\_request\_id;

END IF;

PERFORM 1

FROM Costumiers

WHERE

user\_id = p\_approver\_costumier\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Costumier with id % does not exist', p\_approver\_costumier\_id;

END IF;

IF LENGTH(p\_comment) < 1 THEN

RAISE EXCEPTION 'Comment must be at least 1 character';

END IF;

BEGIN

SELECT requester\_user\_id INTO r\_requester\_user\_id

FROM Requests

WHERE

id = p\_request\_id

FOR UPDATE;

PERFORM 1

FROM Return\_costume\_item\_requests

WHERE

request\_id = p\_request\_id

FOR UPDATE;

-- 3-> DENY

UPDATE Requests

SET

state\_id = 3

WHERE

id = p\_request\_id;

UPDATE Return\_costume\_item\_requests

SET

approver\_costumier\_id = p\_approver\_costumier\_id

WHERE

request\_id = p\_request\_id;

CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## accept\_borrow\_costume\_item\_request

**Opis**:

* Pozwala zaakceptować żądanie o pożyczenie elementu stroju.
* Generuje powiadomienie rozszerzone o przekazany komentarz. (np. kiedy i skąd odebrać element).

**Implementacja**:

CREATE OR REPLACE PROCEDURE accept\_borrow\_costume\_item\_request(

p\_request\_id INTEGER,

p\_comment TEXT

) AS $$

DECLARE

r\_requester\_user\_id INT;

notification\_content TEXT;

BEGIN

IF p\_request\_id IS NULL OR

p\_comment IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

PERFORM 1

FROM Borrow\_costume\_item\_requests

WHERE

request\_id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % is not borrow\_costume\_item\_request', p\_request\_id;

END IF;

IF LENGTH(p\_comment) < 1 THEN

RAISE EXCEPTION 'Comment must be at least 1 character';

END IF;

BEGIN

SELECT requester\_user\_id INTO r\_requester\_user\_id

FROM Requests

WHERE

id = p\_request\_id

FOR UPDATE;

PERFORM 1

FROM Borrow\_costume\_item\_requests

WHERE

request\_id = p\_request\_id

FOR UPDATE;

-- 2-> ACCEPT

UPDATE Requests

SET

state\_id = 2

WHERE

id = p\_request\_id;

notification\_content := 'Request with id ' || p\_request\_id || ' has been accepted. ' || p\_comment;

CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## deny\_borrow\_costume\_item\_request

**Opis**:

* Pozwala odrzucić żądanie o pożyczenie elementu stroju.
* Generuje powiadomienie rozszerzone o przekazany komentarz. (np. dlaczego został odrzucony np. Nie mogę pożyczyć ze względu na wyjazd.).

**Implementacja**:

CREATE OR REPLACE PROCEDURE deny\_borrow\_costume\_item\_request(

p\_request\_id INTEGER,

p\_comment TEXT

) AS $$

DECLARE

r\_requester\_user\_id INT;

notification\_content TEXT := 'Request with id ' || p\_request\_id || ' has been denied.' || p\_comment;

BEGIN

IF p\_request\_id IS NULL OR

p\_comment IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Requests

WHERE

id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % does not exist', p\_request\_id;

END IF;

PERFORM 1

FROM Borrow\_costume\_item\_requests

WHERE

request\_id = p\_request\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Request with id % is not borrow\_costume\_item\_request', p\_request\_id;

END IF;

IF LENGTH(p\_comment) < 1 THEN

RAISE EXCEPTION 'Comment must be at least 1 character';

END IF;

BEGIN

SELECT requester\_user\_id INTO r\_requester\_user\_id

FROM Requests

WHERE

id = p\_request\_id

FOR UPDATE;

PERFORM 1

FROM Borrow\_costume\_item\_requests

WHERE

request\_id = p\_request\_id

FOR UPDATE;

-- 3-> DENY

UPDATE Requests

SET

state\_id = 3

WHERE

id = p\_request\_id;

CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## mark\_notification\_as\_read

**Opis**:

* Pozwala oznaczyć powiadomienie jako przeczytane.

**Implementacja**:

CREATE OR REPLACE PROCEDURE mark\_notification\_as\_read(

p\_notification\_id INTEGER

) AS $$

BEGIN

IF p\_notification\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Notifications

WHERE

id = p\_notification\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Notification with id % does not exist', p\_notification\_id;

END IF;

BEGIN

PERFORM 1

FROM Notifications

WHERE

id = p\_notification\_id

FOR UPDATE;

UPDATE Notifications

SET

marked\_as\_read = 'T'

WHERE

id = p\_notification\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed to update: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## rent\_costume\_item

**Opis**:

* Pozwala na wypożyczenie elementu stroju.

**Implementacja**:

CREATE OR REPLACE PROCEDURE rent\_costume\_item(

p\_user\_id INTEGER,

p\_costume\_item\_id INTEGER,

p\_done\_due\_request\_id INTEGER

) AS $$

DECLARE

r\_location\_id SMALLINT;

BEGIN

IF p\_user\_id IS NULL OR

p\_costume\_item\_id IS NULL OR

p\_done\_due\_request\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

BEGIN

CALL add\_rental(p\_user\_id, p\_costume\_item\_id, p\_done\_due\_request\_id, date\_trunc('minute', NOW()::TIMESTAMP));

SELECT home\_location\_id INTO r\_location\_id

FROM Users

WHERE

id = p\_user\_id;

CALL update\_costume\_item\_location(p\_costume\_item\_id, r\_location\_id);

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

return\_costume\_item

**Opis**:

* Pozwala na oddanie elementu stroju.

**Implementacja**:

CREATE OR REPLACE PROCEDURE return\_costume\_item(

p\_rental\_id INTEGER

) AS $$

BEGIN

IF p\_rental\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Rentals

WHERE

id = p\_rental\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Rental with id % does not exist', p\_rental\_id;

END IF;

BEGIN

PERFORM 1

FROM Rentals

WHERE

id = p\_rental\_id

FOR UPDATE;

UPDATE Rentals

SET

date\_of\_return = date\_trunc('minute', NOW()::TIMESTAMP)

WHERE

id = p\_rental\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

## borrow\_costume\_item

**Opis**:

* Pozwala na pożyczenie elementu stroju.

**Implementacja**:

CREATE OR REPLACE PROCEDURE borrow\_costume\_item(

p\_rental\_id INTEGER,

p\_new\_owner\_user\_id INTEGER,

p\_costume\_item\_id INTEGER,

p\_done\_due\_request\_id INTEGER

) AS $$

DECLARE

swap\_datetime TIMESTAMP;

BEGIN

IF p\_rental\_id IS NULL OR

p\_new\_owner\_user\_id IS NULL OR

p\_costume\_item\_id IS NULL OR

p\_done\_due\_request\_id IS NULL THEN

RAISE EXCEPTION 'All parameters cannot be NULL';

END IF;

PERFORM 1

FROM Rentals

WHERE

id = p\_rental\_id;

IF NOT FOUND THEN

RAISE EXCEPTION 'Rental with id % does not exist', p\_rental\_id;

END IF;

BEGIN

PERFORM 1

FROM Rentals

WHERE

id = p\_rental\_id

FOR UPDATE;

swap\_datetime := date\_trunc('minute', NOW()::TIMESTAMP);

CALL add\_rental(p\_new\_owner\_user\_id, p\_costume\_item\_id, p\_done\_due\_request\_id, swap\_datetime);

UPDATE Rentals

SET

date\_of\_return = swap\_datetime

WHERE

id = p\_rental\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Failed: %', SQLERRM;

END;

END;

$$ LANGUAGE plpgsql;

# Indeksy

PostgreSQL automatycznie tworzy unikalny indeks, gdy definiowany jest PK lub ograniczenie UNIQUE. Dobrą praktyką jest tworzone indeksu na FK, dodać i przetestować, czy jest poprawa. Zobacz sobie OLD\_ERD.jpg tam widać jakie kolumny mają UNIQUE itera U po lewej stronie nazwy kolumny. Jak jest więcej niż jedno U w tabeli oznacza to UNIQUE na kilku kolumnach a nie pojedyncze UNIQUE na każdej z kolumn.

## I1

Opis:

Zadaniem indexu jest

Implementacja:

# Użytkownicy

## Normal\_user

**Opis**:

* Za pomocę tego konta można wykonać operacje zwykłego użytkownika (np. wypożyczyć i oddać element stroju, złożyć requesty, zobaczyć stroje, powiadomienia itp).

**Uprawnienia:**

CREATE ROLE normal\_user WITH LOGIN PASSWORD 'zasxZSXA23!@#';

GRANT EXECUTE ON FUNCTION get\_user\_unresolved\_borrow\_requests TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_user\_current\_rentals TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_aprons TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_boots TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_petticoats TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_skirts TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_caftans TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_corsets TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_neck\_accessories TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_head\_accessories TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_belts TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_pants TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_fits\_shirts TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_user\_unread\_notifications TO normal\_user;

GRANT EXECUTE ON FUNCTION get\_user\_unclosed\_costume\_item\_requests TO normal\_user;

GRANT EXECUTE ON PROCEDURE add\_rental\_costume\_item\_request TO normal\_user;

GRANT EXECUTE ON PROCEDURE add\_return\_costume\_item\_request TO normal\_user;

GRANT EXECUTE ON PROCEDURE add\_borrow\_costume\_item\_request TO normal\_user;

GRANT EXECUTE ON PROCEDURE delete\_request TO normal\_user;

GRANT EXECUTE ON PROCEDURE accept\_borrow\_costume\_item\_request TO normal\_user;

GRANT EXECUTE ON PROCEDURE deny\_borrow\_costume\_item\_request TO normal\_user;

GRANT EXECUTE ON PROCEDURE borrow\_costume\_item TO normal\_user;

GRANT SELECT ON TABLE Detailed\_aprons TO normal\_user;

GRANT SELECT ON TABLE Detailed\_boots TO normal\_user;

GRANT SELECT ON TABLE Detailed\_petticoats TO normal\_user;

GRANT SELECT ON TABLE Detailed\_skirts TO normal\_user;

GRANT SELECT ON TABLE Detailed\_caftans TO normal\_user;

GRANT SELECT ON TABLE Detailed\_corsets TO normal\_user;

GRANT SELECT ON TABLE Detailed\_neck\_accessories TO normal\_user;

GRANT SELECT ON TABLE Detailed\_head\_accessories TO normal\_user;

GRANT SELECT ON TABLE Detailed\_belts TO normal\_user;

GRANT SELECT ON TABLE Detailed\_pants TO normal\_user;

GRANT SELECT ON TABLE Detailed\_shirts TO normal\_user;

GRANT SELECT ON TABLE Costume\_with\_costume\_items\_name TO normal\_user;

## Costumier

**Opis**:

* Za pomocę tego konta można wykonać operacje związane elementami strojów i wypożyczaniem (np. dodać element stroju, edytować go, zaakceptować request, zobaczyć statystyki związane z wypożyczeniami itp).

**Uprawnienia:**

CREATE ROLE costumier WITH LOGIN PASSWORD 'vfgESDy^%783';

GRANT EXECUTE ON FUNCTION get\_costume\_item\_rental\_history TO costumier;

GRANT EXECUTE ON FUNCTION get\_user\_rental\_history TO costumier;

GRANT EXECUTE ON FUNCTION get\_costumier\_unresolved\_requests TO costumier;

GRANT EXECUTE ON FUNCTION get\_user\_current\_rentals TO costumier;

GRANT EXECUTE ON PROCEDURE add\_color TO costumier;

GRANT EXECUTE ON PROCEDURE add\_collection TO costumier;

GRANT EXECUTE ON PROCEDURE add\_pattern TO costumier;

GRANT EXECUTE ON PROCEDURE add\_head\_accessory\_category TO costumier;

GRANT EXECUTE ON PROCEDURE add\_apron TO costumier;

GRANT EXECUTE ON PROCEDURE add\_head\_accessory TO costumier;

GRANT EXECUTE ON PROCEDURE add\_caftan TO costumier;

GRANT EXECUTE ON PROCEDURE add\_petticoat TO costumier;

GRANT EXECUTE ON PROCEDURE add\_corset TO costumier;

GRANT EXECUTE ON PROCEDURE add\_skirt TO costumier;

GRANT EXECUTE ON PROCEDURE add\_belt TO costumier;

GRANT EXECUTE ON PROCEDURE add\_shirt TO costumier;

GRANT EXECUTE ON PROCEDURE add\_pants TO costumier;

GRANT EXECUTE ON PROCEDURE add\_boots TO costumier;

GRANT EXECUTE ON PROCEDURE add\_neck\_accessory TO costumier;

GRANT EXECUTE ON PROCEDURE add\_apron TO costumier;

GRANT EXECUTE ON PROCEDURE update\_head\_accessory TO costumier;

GRANT EXECUTE ON PROCEDURE update\_caftan TO costumier;

GRANT EXECUTE ON PROCEDURE update\_petticoat TO costumier;

GRANT EXECUTE ON PROCEDURE update\_corset TO costumier;

GRANT EXECUTE ON PROCEDURE update\_skirt TO costumier;

GRANT EXECUTE ON PROCEDURE update\_belt TO costumier;

GRANT EXECUTE ON PROCEDURE update\_shirt TO costumier;

GRANT EXECUTE ON PROCEDURE update\_pants TO costumier;

GRANT EXECUTE ON PROCEDURE update\_boots TO costumier;

GRANT EXECUTE ON PROCEDURE update\_neck\_accessory TO costumier;

GRANT EXECUTE ON PROCEDURE add\_costume TO costumier;

GRANT EXECUTE ON PROCEDURE update\_costume\_item\_location TO costumier;

GRANT EXECUTE ON PROCEDURE accept\_rental\_costume\_item\_request TO costumier;

GRANT EXECUTE ON PROCEDURE accept\_return\_costume\_item\_request TO costumier;

GRANT EXECUTE ON PROCEDURE deny\_rental\_costume\_item\_request TO costumier;

GRANT EXECUTE ON PROCEDURE deny\_return\_costume\_item\_request TO costumier;

GRANT EXECUTE ON PROCEDURE rent\_costume\_item TO costumier;

GRANT EXECUTE ON PROCEDURE return\_costume\_item TO costumier;

GRANT SELECT ON TABLE Locations\_with\_settlements\_regions\_countries TO costumier;

GRANT SELECT ON TABLE Detailed\_aprons TO costumier;

GRANT SELECT ON TABLE Detailed\_boots TO costumier;

GRANT SELECT ON TABLE Detailed\_petticoats TO costumier;

GRANT SELECT ON TABLE Detailed\_skirts TO costumier;

GRANT SELECT ON TABLE Detailed\_caftans TO costumier;

GRANT SELECT ON TABLE Detailed\_corsets TO costumier;

GRANT SELECT ON TABLE Detailed\_neck\_accessories TO costumier;

GRANT SELECT ON TABLE Detailed\_head\_accessories TO costumier;

GRANT SELECT ON TABLE Detailed\_belts TO costumier;

GRANT SELECT ON TABLE Detailed\_pants TO costumier;

GRANT SELECT ON TABLE Detailed\_shirts TO costumier;

GRANT SELECT ON TABLE Costume\_with\_costume\_items\_name TO costumier;

GRANT SELECT ON TABLE Costume\_item\_count\_by\_collection\_and\_class TO costumier;

GRANT SELECT ON TABLE Costume\_item\_count\_by\_class TO costumier;

GRANT SELECT ON TABLE Current\_rentals\_count\_by\_costume\_item\_class TO costumier;

GRANT SELECT ON TABLE Current\_rentals\_count\_by\_user\_function TO costumier;

GRANT SELECT ON TABLE Detailed\_rentals TO costumier;

GRANT SELECT ON TABLE Detailed\_current\_rentals TO costumier;

GRANT SELECT ON TABLE Colors TO costumier;

GRANT SELECT ON TABLE Collections TO costumier;

GRANT SELECT ON TABLE Patterns TO costumier;

GRANT SELECT ON TABLE Genders TO costumier;

GRANT SELECT ON TABLE Head\_accessory\_categories TO costumier;

## Manager

**Opis**:

* Za pomocę tego konta można wykonać operacje związane z zarządzaniem użytkownikami (np. dodać lokalizację, użytkownika, nadać funkcje użytkownikowi, zobaczyć statystyki związane z użytkownikami itp).

**Uprawnienia:**

CREATE ROLE mamager WITH LOGIN PASSWORD 'YDjydgdse6F7ca6ki5@7bxZ';

GRANT EXECUTE ON FUNCTION get\_user\_function\_percentage TO mamager;

GRANT EXECUTE ON PROCEDURE add\_country TO mamager;

GRANT EXECUTE ON PROCEDURE add\_region TO mamager;

GRANT EXECUTE ON PROCEDURE add\_settlement TO mamager;

GRANT EXECUTE ON PROCEDURE add\_location TO mamager;

GRANT EXECUTE ON PROCEDURE add\_gender TO mamager;

GRANT EXECUTE ON PROCEDURE add\_role TO mamager;

GRANT EXECUTE ON PROCEDURE add\_type\_of\_voice TO mamager;

GRANT EXECUTE ON PROCEDURE add\_type\_of\_instrument TO mamager;

GRANT EXECUTE ON PROCEDURE add\_dance TO mamager;

GRANT EXECUTE ON PROCEDURE add\_state\_of\_request TO mamager;

GRANT EXECUTE ON PROCEDURE add\_user TO mamager;

GRANT EXECUTE ON PROCEDURE make\_user\_costumier TO mamager;

GRANT EXECUTE ON PROCEDURE make\_user\_dancer TO mamager;

GRANT EXECUTE ON PROCEDURE make\_user\_musician TO mamager;

GRANT EXECUTE ON PROCEDURE make\_user\_singer TO mamager;

GRANT EXECUTE ON PROCEDURE add\_voice\_to\_singer TO mamager;

GRANT EXECUTE ON PROCEDURE add\_instrument\_to\_musician TO mamager;

GRANT EXECUTE ON PROCEDURE add\_dance\_to\_dancer TO mamager;

GRANT SELECT ON TABLE Current\_rentals\_count\_by\_user\_function TO mamager;

GRANT SELECT ON TABLE User\_count\_by\_settlement TO mamager;

GRANT SELECT ON TABLE User\_function\_counts TO mamager;

GRANT SELECT ON TABLE Detailed\_users TO mamager;

GRANT SELECT ON TABLE Detailed\_singers TO mamager;

GRANT SELECT ON TABLE Detailed\_musicians TO mamager;

GRANT SELECT ON TABLE Detailed\_dancers TO mamager;

GRANT SELECT ON TABLE Singer\_count\_by\_voice\_type TO mamager;

GRANT SELECT ON TABLE Musician\_count\_by\_instrument\_type TO mamager;

GRANT SELECT ON TABLE Dancer\_count\_by\_dance\_type TO mamager;

GRANT SELECT ON TABLE Countries TO mamager;

GRANT SELECT ON TABLE Regions TO mamager;

GRANT SELECT ON TABLE Settlements TO mamager;

GRANT SELECT ON TABLE Locations TO mamager;

GRANT SELECT ON TABLE Genders TO mamager;

GRANT SELECT ON TABLE Users TO mamager;

GRANT SELECT ON TABLE Roles TO mamager;

GRANT SELECT ON TABLE Types\_of\_voices TO mamager;

GRANT SELECT ON TABLE Types\_of\_instruments TO mamager;

GRANT SELECT ON TABLE Dances TO mamager;

GRANT SELECT ON TABLE Costumiers TO mamager;

GRANT SELECT ON TABLE Singers TO mamager;

GRANT SELECT ON TABLE Singer\_voices TO mamager;

GRANT SELECT ON TABLE Musicians TO mamager;

GRANT SELECT ON TABLE Musician\_instrument TO mamager;

GRANT SELECT ON TABLE Dancers TO mamager;

GRANT SELECT ON TABLE Dancer\_dance TO mamager;

## Para\_admin

**Opis**:

* Za pomocę tego konta można wykonać wszystkie użytkowe funkcje i procedury, zobaczyć wszystkie widoki i wykonać operacje na tabelach (SELECET, INSERT, UPDATE, DELETE).

**Uprawnienia:**

CREATE ROLE para\_admin WITH LOGIN PASSWORD 'jydseF75@#cBjuF$%sufAQ3%nF^\*KpHF0';

GRANT EXECUTE ON FUNCTION get\_user\_unresolved\_borrow\_requests TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_user\_current\_rentals TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_aprons TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_boots TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_petticoats TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_skirts TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_caftans TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_corsets TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_neck\_accessories TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_head\_accessories TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_belts TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_pants TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_fits\_shirts TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_user\_unread\_notifications TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_user\_unclosed\_costume\_item\_requests TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_costume\_item\_rental\_history TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_user\_rental\_history TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_costumier\_unresolved\_requests TO para\_admin;

GRANT EXECUTE ON FUNCTION get\_user\_function\_percentage TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_rental\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_return\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_borrow\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE delete\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE accept\_borrow\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE deny\_borrow\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE borrow\_costume\_item TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_apron TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_head\_accessory TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_caftan TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_petticoat TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_corset TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_skirt TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_belt TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_shirt TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_pants TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_boots TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_neck\_accessory TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_apron TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_head\_accessory TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_caftan TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_petticoat TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_corset TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_skirt TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_belt TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_shirt TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_pants TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_boots TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_neck\_accessory TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_costume TO para\_admin;

GRANT EXECUTE ON PROCEDURE update\_costume\_item\_location TO para\_admin;

GRANT EXECUTE ON PROCEDURE accept\_rental\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE accept\_return\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE deny\_rental\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE deny\_return\_costume\_item\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE rent\_costume\_item TO para\_admin;

GRANT EXECUTE ON PROCEDURE return\_costume\_item TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_color TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_collection TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_pattern TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_head\_accessory\_category TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_country TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_region TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_settlement TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_location TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_gender TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_role TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_type\_of\_voice TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_type\_of\_instrument TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_dance TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_state\_of\_request TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_user TO para\_admin;

GRANT EXECUTE ON PROCEDURE make\_user\_costumier TO para\_admin;

GRANT EXECUTE ON PROCEDURE make\_user\_dancer TO para\_admin;

GRANT EXECUTE ON PROCEDURE make\_user\_musician TO para\_admin;

GRANT EXECUTE ON PROCEDURE make\_user\_singer TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_voice\_to\_singer TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_instrument\_to\_musician TO para\_admin;

GRANT EXECUTE ON PROCEDURE add\_dance\_to\_dancer TO para\_admin;

GRANT SELECT ON TABLE Locations\_with\_settlements\_regions\_countries TO para\_admin;

GRANT SELECT ON TABLE Detailed\_aprons TO para\_admin;

GRANT SELECT ON TABLE Detailed\_boots TO para\_admin;

GRANT SELECT ON TABLE Detailed\_petticoats TO para\_admin;

GRANT SELECT ON TABLE Detailed\_skirts TO para\_admin;

GRANT SELECT ON TABLE Detailed\_caftans TO para\_admin;

GRANT SELECT ON TABLE Detailed\_corsets TO para\_admin;

GRANT SELECT ON TABLE Detailed\_neck\_accessories TO para\_admin;

GRANT SELECT ON TABLE Detailed\_head\_accessories TO para\_admin;

GRANT SELECT ON TABLE Detailed\_belts TO para\_admin;

GRANT SELECT ON TABLE Detailed\_pants TO para\_admin;

GRANT SELECT ON TABLE Detailed\_shirts TO para\_admin;

GRANT SELECT ON TABLE Costume\_with\_costume\_items\_name TO para\_admin;

GRANT SELECT ON TABLE Costume\_item\_count\_by\_collection\_and\_class TO para\_admin;

GRANT SELECT ON TABLE Costume\_item\_count\_by\_class TO para\_admin;

GRANT SELECT ON TABLE Current\_rentals\_count\_by\_costume\_item\_class TO para\_admin;

GRANT SELECT ON TABLE Current\_rentals\_count\_by\_user\_function TO para\_admin;

GRANT SELECT ON TABLE Detailed\_rentals TO para\_admin;

GRANT SELECT ON TABLE Detailed\_current\_rentals TO para\_admin;

GRANT SELECT ON TABLE User\_count\_by\_settlement TO para\_admin;

GRANT SELECT ON TABLE User\_function\_counts TO para\_admin;

GRANT SELECT ON TABLE Detailed\_users TO para\_admin;

GRANT SELECT ON TABLE Detailed\_singers TO para\_admin;

GRANT SELECT ON TABLE Detailed\_musicians TO para\_admin;

GRANT SELECT ON TABLE Detailed\_dancers TO para\_admin;

GRANT SELECT ON TABLE Singer\_count\_by\_voice\_type TO para\_admin;

GRANT SELECT ON TABLE Musician\_count\_by\_instrument\_type TO para\_admin;

GRANT SELECT ON TABLE Dancer\_count\_by\_dance\_type TO para\_admin;

GRANT SELECT ON TABLE Not\_read\_notifications TO para\_admin;

GRANT SELECT ON TABLE Detailed\_rental\_costume\_item\_requests TO para\_admin;

GRANT SELECT ON TABLE Detailed\_return\_costume\_item\_requests TO para\_admin;

GRANT SELECT ON TABLE Detailed\_borrow\_costume\_item\_requests TO para\_admin;

GRANT SELECT ON TABLE Detailed\_costume\_item\_requests TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Colors TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Collections TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Patterns TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Head\_accessory\_categories TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Countries TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Regions TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Settlements TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Locations TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Genders TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Users TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Roles TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Types\_of\_voices TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Types\_of\_instruments TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Dances TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Costumiers TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Singers TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Singer\_voices TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Musicians TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Musician\_instrument TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Dancers TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Dancer\_dance TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Costumes\_items TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Head\_accessories TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Aprons TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Caftans TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Petticoats TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Corsets TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Skirts TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Belts TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Shirts TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Pants TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Boots TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Neck\_accessories TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Costumes TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE States\_of\_requests TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Requests TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Rental\_costume\_item\_requests TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Return\_costume\_item\_requests TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Borrow\_costume\_item\_requests TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Notifications TO para\_admin;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Rentals TO para\_admin;

# Przykłady użycia

## Akceptacja request’u wypożycz

**Opis**:

* Kostiumolog (id=10) sprawdza powiadomienia -> sprawdza nieodsłużone requesty -> daje approval do reqesta (id=25) wypożycz -> oznacza powiadomienie (id=34) jako przeczytane.

**Implementacja**:

SELECT \* FROM get\_user\_unread\_notifications(10);

SELECT \* FROM get\_costumier\_unresolved\_requests();

CALL accept\_rental\_costume\_item\_request(25, 10, 'Przyjdź 10.01.2025 do głównej siedziby o 17:15 po odbiór');

CALL mark\_notification\_as\_read(34);

## Finalizacja wypożyczenia

**Opis**:

* Kostiumolog finalizuje wypożyczenie z Akceptacja request’u wypożycz.

**Implementacja**:

CALL rent\_costume\_item(8, 12, 25);

## Odmowa request’u pożycz

**Opis**:

* Użytkownik (id=13) sprawdza powiadomienia -> sprawdza nieodsłużone requesty Pożycz -> daje deny do requesta (id=37) -> oznacza powiadomienie (id=45) jako przeczytane.

**Implementacja**:

SELECT \* FROM get\_user\_unread\_notifications(13);

SELECT \* FROM get\_user\_unresolved\_borrow\_requests(13);

CALL deny\_borrow\_costume\_item\_request(37, 'Potrzebuję ten element bo jest występ w środę');

CALL mark\_notification\_as\_read(45);

## Użytkownik wypożycza koszulę

**Opis**:

* Użytkownik (id=25) sprawdza swoje aktualne wypożyczenia -> sprawdza pasujące koszule do wypożyczenia -> składa request o wypożyczenie (zła koszula id=16) -> sprawdza stan requestów -> anuluje request (id=54) -> składa request o wypożyczenie (poprawna koszula id=17) -> sprawdza stan requestów.

**Implementacja**:

SELECT \* FROM get\_user\_current\_rentals(25);

SELECT \* FROM get\_fits\_shirts(25);

CALL add\_rental\_costume\_item\_request(25, 16);

SELECT \* FROM get\_user\_unclosed\_costume\_item\_requests(25);

CALL delete\_request(54);

CALL add\_rental\_costume\_item\_request(25, 17);

SELECT \* FROM get\_user\_unclosed\_costume\_item\_requests(25);

## Dodanie i poprawa informacji o fartuszku

**Opis**:

* Kostiumolog dodaje nowy fartuszek -> wyświetla informację o fartuszkach (zauważa błąd [5cm zamiast 55cm długości]) -> poprawa błąd.

**Implementacja**:

CALL add\_apron('Fartuszek #34', 1::SMALLINT, 2::SMALLINT, 7::SMALLINT, 2::SMALLINT, 5::SMALLINT, 3::SMALLINT);

SELECT \* FROM Detailed\_aprons;

CALL update\_apron(21, 'Fartuszek #34', 1::SMALLINT, 2::SMALLINT, 7::SMALLINT, 2::SMALLINT, 55::SMALLINT, 3::SMALLINT);

## Sprawdzenie wypożyczenia elementu stroju.

**Opis**:

* Kostiumolog sprawdza historię wypożyczenia butów (id=6)

**Implementacja**:

SELECT \* FROM get\_costume\_item\_rental\_history(6);

1. https://www.facebook.com/mietniowiacy [↑](#footnote-ref-1)