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

Autorzy KKK: Kaczor Filip, Kowalski Jakub, Kogut Jakub

Spis Treści

[Spis pojęć 6](#_Toc186071114)

[Opis projektu 6](#_Toc186071115)

[Wymagania Klient 6](#_Toc186071116)

[Wymagania Strój 6](#_Toc186071117)

[Wymagania User 6](#_Toc186071118)

[Wymagania Kostiumolog 7](#_Toc186071119)

[Schemat bazy danych 8](#_Toc186071120)

[Tabele 9](#_Toc186071121)

[Countries 9](#_Toc186071122)

[Regions 10](#_Toc186071123)

[Settlements 12](#_Toc186071124)

[Locations 14](#_Toc186071125)

[Genders 16](#_Toc186071126)

[Users 17](#_Toc186071127)

[Roles 21](#_Toc186071128)

[Types\_of\_voices 22](#_Toc186071129)

[Types\_of\_instruments 23](#_Toc186071130)

[Dances 24](#_Toc186071131)

[Costumiers 25](#_Toc186071132)

[Singers 27](#_Toc186071133)

[Singer\_voices 29](#_Toc186071134)

[Musicians 31](#_Toc186071135)

[Musician\_instrument 33](#_Toc186071136)

[Dancers 35](#_Toc186071137)

[Dancer\_dance 37](#_Toc186071138)

[Colors 39](#_Toc186071139)

[Collections 40](#_Toc186071140)

[Costumes\_items 41](#_Toc186071141)

[Patterns 44](#_Toc186071142)

[Head\_accessory\_categories 45](#_Toc186071143)

[Head\_accessories 46](#_Toc186071144)

[Aprons 48](#_Toc186071145)

[Caftans 50](#_Toc186071146)

[Petticoats 52](#_Toc186071147)

[Corsets 54](#_Toc186071148)

[Skirts 56](#_Toc186071149)

[Belts 58](#_Toc186071150)

[Shirts 60](#_Toc186071151)

[Pants 63](#_Toc186071152)

[Boots 65](#_Toc186071153)

[Neck\_accessories 67](#_Toc186071154)

[Costumes 69](#_Toc186071155)

[States\_of\_requests 76](#_Toc186071156)

[Requests 77](#_Toc186071157)

[Rental\_costume\_item\_requests 79](#_Toc186071158)

[Return\_costume\_item\_requests 81](#_Toc186071159)

[Borrow\_costume\_item\_requests 84](#_Toc186071160)

[Notifications 87](#_Toc186071161)

[Rentals 89](#_Toc186071162)

[Wyzwalacze 92](#_Toc186071163)

[prevent\_invalid\_costume\_item\_insert 92](#_Toc186071164)

[prevent\_invalid\_costume\_insert\_update 95](#_Toc186071165)

[prevent\_invalid\_request\_delete 96](#_Toc186071166)

[prevent\_invalid\_request\_insert 97](#_Toc186071167)

[prevent\_invalid\_rental\_costume\_item\_request\_insert 99](#_Toc186071168)

[prevent\_invalid\_return\_costume\_item\_request\_insert 100](#_Toc186071169)

[prevent\_invalid\_borrow\_costume\_item\_request\_insert 101](#_Toc186071170)

[prevent\_invalid\_notification\_insert 103](#_Toc186071171)

[prevent\_invalid\_rental\_insert 104](#_Toc186071172)

[Widoki 105](#_Toc186071173)

[Locations\_with\_settlements\_regions\_countries 105](#_Toc186071174)

[User\_count\_by\_settlement 106](#_Toc186071175)

[User\_function\_counts 107](#_Toc186071176)

[Detailed\_users 108](#_Toc186071177)

[Detailed\_singers 110](#_Toc186071178)

[Detailed\_musicians 112](#_Toc186071179)

[Detailed\_dancers 114](#_Toc186071180)

[Singer\_count\_by\_voice\_type 116](#_Toc186071181)

[Musician\_count\_by\_instrument\_type 117](#_Toc186071182)

[Dancer\_count\_by\_dance\_type 118](#_Toc186071183)

[~~Widok1~~ 119](#_Toc186071184)

[Costume\_item\_count\_by\_collection\_and\_class 120](#_Toc186071185)

[Costume\_item\_count\_by\_class 125](#_Toc186071186)

[Detailed\_aprons 127](#_Toc186071187)

[Detailed\_boots 128](#_Toc186071188)

[Detailed\_petticoats 129](#_Toc186071189)

[Detailed\_skirts 130](#_Toc186071190)

[Detailed\_caftans 131](#_Toc186071191)

[Detailed\_corsets 132](#_Toc186071192)

[Detailed\_neck\_accessories 133](#_Toc186071193)

[Detailed\_head\_accessories 134](#_Toc186071194)

[Detailed\_belts 135](#_Toc186071195)

[Detailed\_pants 136](#_Toc186071196)

[Detailed\_shirts 137](#_Toc186071197)

[Costume\_with\_costume\_items\_name 138](#_Toc186071198)

[Not\_read\_notifications 140](#_Toc186071199)

[Detaild\_rental\_costume\_item\_requests 141](#_Toc186071200)

[Detaild\_return\_costume\_item\_requests 142](#_Toc186071201)

[Detaild\_borrow\_costume\_item\_requests 143](#_Toc186071202)

[~~Widok1~~ 144](#_Toc186071203)

[Current\_rentals\_count\_by\_costume\_item\_class 145](#_Toc186071204)

[Current\_rentals\_count\_by\_user\_function 147](#_Toc186071205)

[Funkcje 148](#_Toc186071206)

[check\_costume\_inconsistency 148](#_Toc186071207)

[check\_rental\_inconsistency 158](#_Toc186071208)

[get\_costume\_item\_rental\_history 160](#_Toc186071209)

[F1 162](#_Toc186071210)

[Procedury 164](#_Toc186071211)

[add\_country 164](#_Toc186071212)

[add\_region 166](#_Toc186071213)

[add\_settlement 168](#_Toc186071214)

[add\_location 170](#_Toc186071215)

[add\_gender 172](#_Toc186071216)

[add\_role 173](#_Toc186071217)

[add\_type\_of\_voice 174](#_Toc186071218)

[add\_type\_of\_instrument 175](#_Toc186071219)

[add\_dance 176](#_Toc186071220)

[add\_color 177](#_Toc186071221)

[add\_collection 178](#_Toc186071222)

[add\_pattern 179](#_Toc186071223)

[add\_head\_accessory\_category 180](#_Toc186071224)

[add\_state\_of\_request 181](#_Toc186071225)

[add\_user 182](#_Toc186071226)

[make\_user\_costumier 186](#_Toc186071227)

[make\_user\_singer 188](#_Toc186071228)

[make\_user\_musician 190](#_Toc186071229)

[make\_user\_dancer 192](#_Toc186071230)

[add\_voice\_to\_singer 194](#_Toc186071231)

[add\_instrument\_to\_musician 196](#_Toc186071232)

[add\_dence\_to\_dancer 198](#_Toc186071233)

[add\_apron 200](#_Toc186071234)

[add\_head\_accessory 204](#_Toc186071235)

[add\_caftan 208](#_Toc186071236)

[add\_petticoat 212](#_Toc186071237)

[add\_corset 216](#_Toc186071238)

[add\_skirt 220](#_Toc186071239)

[add\_belt 224](#_Toc186071240)

[add\_shirt 227](#_Toc186071241)

[add\_pants 232](#_Toc186071242)

[add\_boots 236](#_Toc186071243)

[add\_neck\_accessory 239](#_Toc186071244)

[add\_costume 243](#_Toc186071245)

[add\_rental\_costume\_item\_request 250](#_Toc186071246)

[add\_return\_costume\_item\_request 252](#_Toc186071247)

[add\_borrow\_costume\_item\_request 254](#_Toc186071248)

[add\_notification 257](#_Toc186071249)

[add\_rental 259](#_Toc186071250)

[update\_costume\_item\_location 261](#_Toc186071251)

[delete\_request 263](#_Toc186071252)

[accept\_rental\_costume\_item\_request 265](#_Toc186071253)

[deny\_rental\_costume\_item\_request 268](#_Toc186071254)

[accept\_return\_costume\_item\_request 271](#_Toc186071255)

[deny\_return\_costume\_item\_request 274](#_Toc186071256)

[accept\_borrow\_costume\_item\_request 277](#_Toc186071257)

[deny\_borrow\_costume\_item\_request 279](#_Toc186071258)

[Indeksy 288](#_Toc186071259)

[I1 288](#_Toc186071260)

[Użytkownicy 288](#_Toc186071261)

[U1 288](#_Toc186071262)

[Pszykłady użycia 288](#_Toc186071263)

[P1 288](#_Toc186071264)

[P1 288](#_Toc186071265)

# Spis pojęć

Klient – Kierownik Zespół Pieśni i Tańca Mietniowiacy.

User – Członek zespołu.

Users – Członkowe zespołu.

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

Kostiumolog – Członek zespołu odpowiedzialny za utrzymanie Strój.

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

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

# 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 Users przeprowadzanie wyporzyczeń Strój. Przedstawicielem Klient od spraw ww. projetu jest członek zespołu – Jakub Kowalski.

## Wymagania Klient

Wyporzyczyć i Oddać Strój odbywa się wyłącznie za zgodą Kostiumolog.

Możliwe jest porzyczenie Strój innemu User. Obdywa się to bez zgody Kostiumolog, natomiast wymagana jest zgoda od User obecnie posiadającego Strój takiej tranzakcji.

User, Kostiumolog zostają poinformowani o zgłoszeniach o Wyporzyczyć, Oddać lub porzyczyć.

~~Edycje informacji o User i Strój powinny być odnotowywane (kto i kiedy).~~

User może pełnić funkcje: tanczerz, śpaiewaka, 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 Wyporzyczyć to lokalizacja to adres User, jeśli Oddać to miejsce ośrodka do którego został zwrócony Strój.

### Wymagania User

User może zobaczyć Strój, Strój należący do innego User bez informacji o lokalizacji oraz swoje Strój.

User może Wyporzyczyć Strój lub go Oddać.

User może zainicjować porzyczenie Strój (punkt 3) nie zależnie do tego czy jest on obecnie właścicielem Strój, czy chce go dopiero porzyczyć.

User może zobaczyć stan Wyporzyczyć, Oddać lub porzyczenia.

User może anulować Wyporzyczyć, Oddać lub porzyczenie Strój.

User może wyrazić zgodę na porzyczenie lub jej nie wyrazić.

User może zobaczyć i oznaczyć jako przeczytaną informację o decyzji związanej z Wyporzyczyć, Oddać i porzyczeniem Strój.

User może zobaczyć historię swoich Wyporzyczyć, Oddać i porzyczeń Strój.

User może edytwać informacje o sobie.

### Wymagania Kostiumolog

Kostiumolog odziedziczy po User.

Kostiumolog może zobaczyć Strój, Strój należący do innego User (zmiana względem User)

Kostiumolog może edytować informacjie o Strój.

Kostiumolog może zobaczyć proźby o Wyporzyczyć i Oddać.

Kostiumolog może wyrazić zgodę na Wyporzyczyć lub Oddać Strój lub jej nie wyrazić.

Kostiumolog może zobaczyć historię Wyporzyczyć, Oddać i porzyczeń Strój innych Users.

# A diagram of a computer Description automatically generated with medium confidenceSchemat bazy danych

# Tabele

## Countries

Opis:  
Tabela zawiera informacje o krajach.

Warunki integralności:  
Nazwa danego kraju może wystąpić tylko raz.

Inplementacja:

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.

Inplementacja:

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 miastach/wsiach danego regionu.  
Region – klucz obcy do tabeli Regions

Warunki integralności:  
Nazwa danego miasta/wsi 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 Settements

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 odpowiada male, id = 2 -> female i id = 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 przyechowywane to: pierwsze imie, nazwisko, data urodzenia, adres email, numer telefonu (kierunkowy), płeć, lokalizacje zamieszkania, wysokość, obwód w pasie, obwód w klatce piersiowej, obwód głowy, obwód szyji, długość nogi (od pasa do kostki), długość ręki (od ramienia do nadgarstku), długość torsu (od szyji 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 posadać 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 ~\* '^\+?[0-9]{7,9}$');

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);

ALTER TABLE Users

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

ALTER TABLE Users

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

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 zestwa dodatkowych atrybutów odpowiadających kostiumatologowi.  
W przypadku usunięcia członka zespołu, dodatkowe dane z tej tabeli też powinny zostać usunięte.  
TODO? – Miejsce pracy może odwoływać się tylko do wybranych Lokacji.

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

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 zestwa 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 tabli 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 zestwa dodatkowych atrybutów odpowiadających człokowi 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 instrumenice 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 zestwa dodatkowych atrybutów odpowiadających człokowi 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 muie tancerz.   
Dancer\_id – klucz obcy do tabeli Dancers.  
Taniec – klucz obcy do tabeli Types\_of\_dances.

Warunki integralności:  
Tancerz może umieć grać tylko raz tym samym instrumentem.  
W przypadku usunięcia rekordu z tabli 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 to 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 zanjduje.  
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 musi być 1 albo 2 albo 3 (male, female, 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 tabli 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 tabli 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 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 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\_max\_waist\_circumference\_value CHECK (min\_waist\_circumference <= max\_waist\_circumference);

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\_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 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 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 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 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 rakres 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. obwód) powinny być większe od 0, wartość max ma być większ 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, szyii 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 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 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 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 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 tabli 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 ozdub na szyję: rekomendowany rakres obwodu szyii.   
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. obwód) powinny być większe od 0, wartość max ma być większ 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 stróju. Kilka strojów może zawierać ten sam element.

Warunki integralności:  
Nazwa stroju musi być unikalna.  
Gender\_id musi być 1 albo 2 albo 3 (male, female, bigender).  
Strój musi być zgodny pod względem rekomendowanej płci i kolekcji do której neleżą 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 rządania.

Warunki integralności:

Dany stan może być tylko raz.  
Status PENDING ma mieć id = 1, ACCEPT id = 2, DENY id = 3.

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 wyporzyczony i jaki Kostiumonolog dane żadzanie obsłużył. NULL w approver\_costumier\_id oznacza to że dowolny Kostiumolog może zakceptować 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 stworzycz jedynie jeśli element stroju jest nie wyporzyczony – TRIGGER prevent\_invalid\_rental\_costume\_item\_request\_insert  
Jeśli został usunięty request to dane z tej tabeli też powinny zstać 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 informacjie o request (z tablicy Requests): jaki element stroju będzie oddany i jaki Kostiumonolog dane żadzanie obsłużył. NULL w approver\_costumier\_id oznacza to że dowolny Kostiumolog może zakceptować 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 zstać 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 porzyczony i jaki członek zespołu dane żadzanie ma zakceptować.  
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 zstać usunięte.  
approver\_user\_id musi posiadać strój przed jego porzyczeniem, nie można porzyczyć 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 powidomieniach dostępnych dla użytkowinika: 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 wyporzyczeniach: kto, co, na podstawie jakiego requestu, czas wyporzyczenia 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 wyporzyczenia.  
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 SELECT zmienić na PERFORM  
TODO? Dodac dla update???

## prevent\_invalid\_costume\_item\_insert

Opis:

Zadaniem wyzwalacza jest uniemożliwienie stworzenie elementu stroju który posiadałby dodatkowe informacjie 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 informacjie specyficzne dla więcej niż jednego danego typy requestu. Request nie może byś 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 elemtu stroju który już jest wyporzyczony.

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 Rental\_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 skierwany do osoby która nie posiada rządanego elementu stroju oraz uniwmożliwienie porzyczenia 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 wyporzyczenia jeśli user\_id i costume\_item\_id nie są zgodne z tymi zawartymi w request zezwalającym na wyporzyczenie.

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:  
Umożliwa 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 człoków 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:  
Poazuje 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 informacje o członkach chóru (z informacją jakimi głosami potrafi śpierwać).

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:  
Pokazjue 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 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 człoknów 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 człoknów 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 człoknów baletu umnie 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

;

## ~~Widok1~~

~~Opis:  
Pokazuje informacje o członkach odpowiedzialnych za stroje (z lokalizacją w której pracują).~~

~~Implementacja:~~

## 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 elemntów 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 informacje o fartuszku.

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 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 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 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 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 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 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 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 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 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 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

INNER JOIN Aprons ia

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

INNER JOIN Costumes\_items a

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

INNER JOIN Caftans ica

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

INNER JOIN Costumes\_items ca

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

INNER JOIN Petticoats ip

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

INNER JOIN Costumes\_items p

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

INNER JOIN Corsets ico

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

INNER JOIN Costumes\_items co

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

INNER JOIN Skirts isk

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

INNER JOIN Costumes\_items sk

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

INNER JOIN Belts ib

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

INNER JOIN Costumes\_items b

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

INNER JOIN Shirts ish

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

INNER JOIN Costumes\_items sh

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

INNER JOIN Pants ipa

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

INNER JOIN Costumes\_items pa

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

INNER JOIN Boots ibo

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

INNER JOIN Costumes\_items bo

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

INNER JOIN Neck\_accessories ine

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

INNER JOIN Costumes\_items ne

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

INNER JOIN Head\_accessories ih

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

INNER 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, content, datetime, due\_to\_request\_id ) AS

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

FROM Notifications

WHERE marked\_as\_read = 'F'

;

## Detaild\_rental\_costume\_item\_requests

Opis:  
Pokazuje informacje o requestach - wyporzyczenie.

Implementacja:

CREATE OR REPLACE VIEW Detaild\_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

;

## Detaild\_return\_costume\_item\_requests

Opis:  
Pokazuje informacje o requestach - oddaj.

Implementacja:

CREATE OR REPLACE VIEW Detaild\_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

;

## Detaild\_borrow\_costume\_item\_requests

Opis:  
Pokazuje informacje o requestach - pożycz.

Implementacja:

CREATE OR REPLACE VIEW Detaild\_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

;

## ~~Widok1~~

~~Opis:  
Pokazuje informacje o wyporzyczeniach.~~

~~Implementacja:~~

POPROSTU SELECT Z TABELI

JAK CHCIEC ZOBACZYC INFO O USER POLACZYĆ Z WIDOKIEM

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

Opis:  
Pokazuje ile elementów stroju jest wyporzyczonych 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 wyporzyczonych 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

;

# Funkcje

## check\_costume\_inconsistency

Opis:

Sprawdza czy elementy stroju mają nie zgodną płeć i kolekcję. Zwraca TRUE kiedy występuje nie konsekwencja.

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 Aprons

        WHERE costume\_item\_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;

    END IF;

    IF f\_belt\_id IS NOT NULL THEN

        PERFORM 1

        FROM Belts

        WHERE costume\_item\_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;

    END IF;

    IF f\_boots\_id IS NOT NULL THEN

        PERFORM 1

        FROM Boots

        WHERE costume\_item\_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;

    END IF;

    IF f\_caftan\_id IS NOT NULL THEN

        PERFORM 1

        FROM Caftans

        WHERE costume\_item\_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;

    END IF;

    IF f\_corset\_id IS NOT NULL THEN

        PERFORM 1

        FROM Corsets

        WHERE costume\_item\_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;

    END IF;

    IF f\_petticoat\_id IS NOT NULL THEN

        PERFORM 1

        FROM Petticoats

        WHERE costume\_item\_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;

    END IF;

    IF f\_skirt\_id IS NOT NULL THEN

        PERFORM 1

        FROM Skirts

        WHERE costume\_item\_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;

    END IF;

    IF f\_shirt\_id IS NOT NULL THEN

        PERFORM 1

        FROM Shirts

        WHERE costume\_item\_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;

    END IF;

    IF f\_pants\_id IS NOT NULL THEN

        PERFORM 1

        FROM Pants

        WHERE costume\_item\_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;

    END IF;

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

        PERFORM 1

        FROM Neck\_accessories

        WHERE costume\_item\_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;

    END IF;

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

        PERFORM 1

        FROM Head\_accessories

        WHERE costume\_item\_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;

    END IF;

    IF f\_apron\_id IS NOT NULL THEN

        PERFORM 1

        FROM Aprons

        WHERE costume\_item\_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 Belts

        WHERE costume\_item\_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 Boots

        WHERE costume\_item\_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 Caftans

        WHERE costume\_item\_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 Corsets

        WHERE costume\_item\_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 Petticoats

        WHERE costume\_item\_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 Skirts

        WHERE costume\_item\_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 Shirts

        WHERE costume\_item\_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 Pants

        WHERE costume\_item\_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 Neck\_accessories

        WHERE costume\_item\_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 Head\_accessories

        WHERE costume\_item\_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 wyporzyczeniu 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 wyporzyczeń danego elementu stroju:  
nazwę, imię, nazwisko, daty wyporzyczeń

Implementacja:

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

    f\_costume\_element\_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

        ci.name,

        u.first\_name,

        u.last\_name,

        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

    WHERE

        ci.id = f\_costume\_element\_id

    ORDER BY

        date\_of\_rental

    ASC;

END;

$$ LANGUAGE plpgsql;

## F1

Opis:  
Pokazuje informacje o requestach, które mają zoastać obsłużone przez Kostumatologa.

Implementacja:

# Procedury

TODO poprawić procedury sprawdzić czy wartości nie są NULL (tam gdzie stosowne)

## add\_country

Opis:

Pozwala dodać kraj, który spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_country(

    p\_country\_name VARCHAR(30)

) AS $$

BEGIN

    IF LENGTH(p\_country\_name) > 30 THEN

        RAISE EXCEPTION 'Country name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_region(

p\_region\_name VARCHAR(30),

p\_country\_id SMALLINT

) AS $$

BEGIN

IF LENGTH(p\_region\_name) > 30 THEN

RAISE EXCEPTION 'Region name exceeded 30 characters';

END IF;

PERFORM 1

FROM Countris

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:

TODO w tabeli misto/wieś zmienic opis na miejscowość.

Pozwala dodać miejscowość, która spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_settlement(

    p\_settlement\_name VARCHAR(30),

    p\_region\_id SMALLINT

) AS $$

BEGIN

    IF LENGTH(p\_settlement\_name) > 30 THEN

        RAISE EXCEPTION 'Settlement name exceeded 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 warónki 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 LENGTH(p\_location\_street) > 30 THEN

        RAISE EXCEPTION 'Street name exceeded 30 characters';

    END IF;

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

        RAISE EXCEPTION 'Postal code exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_gender(

    p\_gender\_name VARCHAR(30)

) AS $$

BEGIN

    IF LENGTH(p\_gender\_name) > 25 THEN

        RAISE EXCEPTION 'Gender name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_role(

    p\_role\_name VARCHAR(20)

) AS $$

BEGIN

    IF LENGTH(p\_gender\_name) > 20 THEN

        RAISE EXCEPTION 'Gender name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_type\_of\_voice(

    p\_type\_of\_voice\_name VARCHAR(10)

) AS $$

BEGIN

    IF LENGTH(p\_type\_of\_voice\_name) > 10 THEN

        RAISE EXCEPTION 'Type of voice name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_type\_of\_instrument(

    p\_type\_of\_instrument\_name VARCHAR(20)

) AS $$

BEGIN

    IF LENGTH(p\_type\_of\_instrument\_name) > 20 THEN

        RAISE EXCEPTION 'Type of instrument name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_dance(

    p\_dance\_name VARCHAR(20)

) AS $$

BEGIN

    IF LENGTH(p\_dance\_name) > 20 THEN

        RAISE EXCEPTION 'Dance name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_color(

    p\_color\_name VARCHAR(25)

) AS $$

BEGIN

    IF LENGTH(p\_color\_name) > 25 THEN

        RAISE EXCEPTION 'Color name exceeded 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, który spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_collection(

    p\_collection\_name VARCHAR(20)

) AS $$

BEGIN

    IF LENGTH(p\_collection\_name) > 20 THEN

        RAISE EXCEPTION 'Collection name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_pattern(

    p\_pattern\_name VARCHAR(20)

) AS $$

BEGIN

    IF LENGTH(p\_pattern\_name) > 20 THEN

        RAISE EXCEPTION 'Pattern name exceeded 20 characters';

    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, który spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_head\_accessory\_category(

    p\_head\_accessory\_category\_name VARCHAR(20)

) AS $$

BEGIN

    IF LENGTH(p\_head\_accessory\_category\_name) > 20 THEN

        RAISE EXCEPTION 'Head accessory category name exceeded 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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_state\_of\_request(

    p\_state\_of\_request\_name VARCHAR(15)

) AS $$

BEGIN

    IF LENGTH(p\_state\_of\_request\_name) > 15 THEN

        RAISE EXCEPTION 'State of request name exceeded 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

TODO można dodać do user że tors leg i arm nie może być większy od height jako CHECK i w procedurze

Opis:

Pozwala dodać użytkownika, który spełnia warónki 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 (50),

    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 LENGTH(p\_user\_first\_name) > 25 THEN

        RAISE EXCEPTION 'First name exceeded 25 characters';

    END IF;

    IF LENGTH(p\_user\_last\_name) > 30 THEN

        RAISE EXCEPTION 'Last name exceeded 30 characters';

    END IF;

    IF LENGTH(p\_user\_email) > 50 OR p\_user\_email ~\* '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' THEN

        RAISE EXCEPTION 'Wrong email format, max characters - 50';

    END IF;

    IF LENGTH(p\_user\_phone\_number) > 12 OR p\_user\_phone\_number ~\* '^\+?[0-9]{7,9}$' THEN

        RAISE EXCEPTION 'Wrong phone number format, max characters - 12';

    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\_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 funkcę Kostumatologa, którą będzie pełnił oraz jej parametry, która spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE make\_user\_costumier(

    p\_user\_id INT,

    p\_role\_id INT,

    p\_work\_location\_id INT

) AS $$

BEGIN

    PERFORM 1

    FROM Roles

    WHERE

        id = p\_region\_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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_singer

Opis:

Pozwala dodać członkowi zespołu funkcę członka chóru, którą będzie pełnił oraz jej parametry, która spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE make\_user\_singer(

    p\_user\_id INT,

    p\_role\_id INT

) AS $$

BEGIN

    PERFORM 1

    FROM Roles

    WHERE

        id = p\_region\_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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_musician

Opis:

Pozwala dodać członkowi zespołu funkcę członka kapeli, którą będzie pełnił oraz jej parametry, która spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE make\_user\_musician(

    p\_user\_id INT,

    p\_role\_id INT

) AS $$

BEGIN

    PERFORM 1

    FROM Roles

    WHERE

        id = p\_region\_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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## make\_user\_dancer

Opis:

Pozwala dodać członkowi zespołu funkcę członka baletu, którą będzie pełnił oraz jej parametry, która spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE make\_user\_dancer(

    p\_user\_id INT,

    p\_role\_id INT

) AS $$

BEGIN

    PERFORM 1

    FROM Roles

    WHERE

        id = p\_region\_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);

        COMMIT;

    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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_voice\_to\_singer(

    p\_singer\_id INT,

    p\_type\_of\_voice\_id INT

) AS $$

BEGIN

    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\_user\_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);

        COMMIT;

    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 warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_instrument\_to\_musician(

    p\_musician\_id INT,

    p\_type\_of\_instrument\_id INT

) AS $$

BEGIN

    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\_singer\_id FOR UPDATE;

        INSERT INTO Musician\_instrument (musician\_id, type\_of\_instrument\_id)

        VALUES (p\_musician\_id, p\_type\_of\_instrument\_id);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_dence\_to\_dancer

Opis:

Pozwala dodać taniec któr umie tańczyć członek baletu, który spełnia warónki spójności bazy danych.

Implementacja:

CREATE OR REPLACE PROCEDURE add\_dance\_to\_dancer(

    p\_dancer\_id INT,

    p\_dance\_id INT

) AS $$

BEGIN

    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 warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    PERFORM 1

    FROM Patterns

    WHERE

        id = p\_pattern\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Pattern with id % does not exist', p\_pattern\_id;

    END IF;

    IF p\_apron\_length <= 0 THEN

        RAISE EXCEPTION 'Length must be greater than 0';

    END IF;

    IF LENGTH(p\_apron\_name) > 30 THEN

        RAISE EXCEPTION 'Apron name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_head\_accessory

Opis:

Pozwala dodać akcesorum na głowe, który spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Head accessory name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_caftan

Opis:

Pozwala dodać kaftan, który spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Caftan name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_petticoat

Opis:

Pozwala dodać halkę, która spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Petticoat name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_corset

Opis:

Pozwala dodać gorset, który spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Corset name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_skirt

Opis:

Pozwala dodać spódnicę, która spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Skirt name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_belt

Opis:

Pozwala dodać pas, który spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Belt name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_shirt

Opis:

Pozwala dodać koszulę, która spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Shirt name exceeded 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\_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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_pants

Opis:

Pozwala dodać spodnie, które spełnia warónki 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

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    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 THEN

        RAISE EXCEPTION 'Pants name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_boots

Opis:

Pozwala dodać buty, który spełnia warónki 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 SMALLINT

) AS $$

DECLARE

    i\_id INT;

BEGIN

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    IF p\_boots\_shoe\_size <= 0 THEN

        RAISE EXCEPTION 'Shoe size must be greater than 0';

    END IF;

    IF LENGTH(p\_boots\_name) > 30 THEN

        RAISE EXCEPTION 'Boots name exceeded 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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_neck\_accessory

Opis:

Pozwala dodać akcesorum na szyję, które spełnia warónki 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\_waist\_circumference SMALLINT,

    p\_neck\_accessory\_max\_waist\_circumference SMALLINT

) AS $$

DECLARE

    i\_id INT;

BEGIN

    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;

    PERFORM 1

    FROM Colors

    WHERE

        id = p\_color\_id;

    IF NOT FOUND THEN

        RAISE EXCEPTION 'Color with id % does not exist', p\_color\_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;

    IF p\_neck\_accessory\_min\_waist\_circumference <= 0 THEN

        RAISE EXCEPTION 'Min waist circumference must be greater than 0';

    END IF;

    IF p\_neck\_accessory\_max\_waist\_circumference >= p\_neck\_accessory\_min\_waist\_circumference THEN

        RAISE EXCEPTION 'Max waist circumference must be greater or equal than min waist circumference';

    END IF;

    IF LENGTH(p\_neck\_accessory\_name) > 30 THEN

        RAISE EXCEPTION 'Neck accessory name exceeded 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\_waist\_circumference, max\_waist\_circumference)

        VALUES (i\_id, p\_neck\_accessory\_min\_waist\_circumference, p\_neck\_accessory\_max\_waist\_circumference);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_costume

Opis:

Pozwala dodać strój, który spełnia warónki 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

    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\_shirt\_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 wyporzyczenie elementu stroju, które spełnia warónki 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

    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);

        COMMIT;

    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 warónki 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

    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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_borrow\_costume\_item\_request

Opis:

Pozwala dodać żądanie o porzyczeniu stroju, który spełnia warónki 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

    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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

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

    END;

END;

$$ LANGUAGE plpgsql;

## add\_notification

Opis:

Pozwala dodać powiadoienie, który spełnia warónki 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

    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;

    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 wyporzyczenie, który spełnia warónki 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

    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;

    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

    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 warónków spójności bazy danych. Tylko z state z id = 1 (PENDING).

Implementacja:

CREATE OR REPLACE PROCEDURE delete\_request(

    p\_request\_id INTEGER

) AS $$

DECLARE

    r\_state\_id INT;

BEGIN

    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 wyporzyczenia elementu stroju. Generuje powiadomienie rozszerzone o przekazany komentarz. (np. gdzie i kiedy 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

    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;

    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;

        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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

            RAISE EXCEPTION 'Failed: %', SQLERRM;

    END;

END;

$$ LANGUAGE plpgsql;

## deny\_rental\_costume\_item\_request

Opis:

Pozwala odrzucić żądanie o wyporzyczenie 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

    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;

    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;

        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);

        COMMIT;

    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 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

    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;

    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;

        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);

        COMMIT;

    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

    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;

    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;

        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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

            RAISE EXCEPTION 'Failed: %', SQLERRM;

    END;

END;

$$ LANGUAGE plpgsql;

## accept\_borrow\_costume\_item\_request

Opis:

Pozwala zaakceptować żądanie o porzyczenie elementu stroju. Generuje powiadomienie rozszerzone o przekazany komentarz. (np. kiedy i zkąd 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

    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;

    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;

        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);

        COMMIT;

    EXCEPTION

        WHEN OTHERS THEN

            RAISE EXCEPTION 'Failed: %', SQLERRM;

    END;

END;

$$ LANGUAGE plpgsql;

## deny\_borrow\_costume\_item\_request

Opis:

Pozwala odrzucić żądanie o porzyczenie elementu stroju. Generuje powiadomienie rozszerzone o przekazany komentarz. (np. dlaczego został odrzucony, poprostu nie chce porzyczyć burak).

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

    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;

    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;

        UPDATE Requests

        SET

            state\_id = 3

        WHERE

            id = p\_request\_id;

        CALL add\_notification(r\_requester\_user\_id, notification\_content, p\_request\_id);

        COMMIT;

    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

    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 wyporzyczenie 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 $$

BEGIN

    BEGIN

        CALL add\_rental(p\_user\_id, p\_costume\_item\_id, p\_done\_due\_request\_id, date\_trunc('minute', NOW()::TIMESTAMP))

    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

    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 porzyczenie 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

    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.

## I1

Opis:

Zadaniem indexu jest

Implementacja:

# Użytkownicy

## U1

# Pszykłady użycia

## P1

Opis:

Konstumatolog sprwadza reqesty -> daje approval do reqest.

## P1

Opis:

Konstumatolog sprawdza ilość wyporzyczonych elementów na klasę -> wyświetla rentals dla klasy najbardziej obciążonej -> wysyła powiadomienie do urzytkowinka o oddanie elementu.

1. https://www.facebook.com/mietniowiacy [↑](#footnote-ref-1)