

**UNIWERSYTET RZESZOWSKI**

Karol Czyż, Sebastian Cupak

131360, 131361

Informatyka i Ekonometria

**Gymplify – system zarządzania siłownią**

Rzeszów, 2025r

Spis treści

[1. Tematyka projektu 2](#_Toc199758477)

[2. Realizacja projektu 2](#_Toc199758478)

[3. Opis tabel 2](#_Toc199758479)

[4. Główne funkcjonalności 4](#_Toc199758480)

[4.1. Zarządzanie Użytkownikami i Uprawnieniami (Pakiet PKG\_USER\_MGMT) 4](#_Toc199758481)

[4.2. Katalogowanie Struktur Treningowych (Pakiety PKG\_MUSCLE\_GROUP\_MGMT, PKG\_EXERCISE\_MGMT) 5](#_Toc199758482)

[4.3. Zarządzanie Planami Treningowymi (Pakiet PKG\_TRAINING\_PLAN\_MGMT) 5](#_Toc199758483)

[4.4. Zarządzanie Planami Personalnymi (Pakiet PKG\_PERSONAL\_PLAN\_MGMT) 5](#_Toc199758484)

[4.5. Statystyki i Raportowanie (Pakiet PKG\_APP\_STATISTICS) 5](#_Toc199758485)

[4.6. Automatyzacja i Zapewnienie Spójności Danych (Triggery) 6](#_Toc199758486)

[5. CRUD dla zarządzania użytkownikami 6](#_Toc199758487)

[5.1 Tworzenie Użytkownika (AddUser) 6](#_Toc199758488)

[5.2 Usuwanie Użytkownika (DeleteUser) 7](#_Toc199758489)

[5.3 Odczyt Użytkowników 8](#_Toc199758490)

[5.4 Aktualizacja Danych Użytkownika 9](#_Toc199758491)

[6. Dodatkowe funkcjonalności 11](#_Toc199758492)

[6.1 Trigger (TRG\_PREVENT\_PLAN\_DELETE\_IF\_LINKED) 11](#_Toc199758493)

[6.2 Procedura (PKG\_TRAINING\_PLAN\_MGMT.AddExerciseToPlan) 12](#_Toc199758494)

[6.3 Zaawansowane Funkcje Analityczne (Pakiet PKG\_APP\_STATISTICS) 13](#_Toc199758495)

[7. Prezentacja Interfejsu Oraz Elementów Backendu 15](#_Toc199758496)

[7.1 Prezentacja przykładowego CRUD’a dla użtykownika. 15](#_Toc199758497)

[7.2 Prezentacja panelu statystyk. 16](#_Toc199758498)

[8. Podsumowanie 17](#_Toc199758499)

# 1. Tematyka projektu

Projekt "Gymplify" dotyczy stworzenia kompleksowego systemu informatycznego wspierającego zarządzanie siłownią oraz aktywnością fizyczną jej użytkowników. Moduł bazodanowy, zrealizowany w technologii Oracle PL/SQL, stanowi rdzeń systemu, odpowiadając za przechowywanie danych, implementację kluczowej logiki biznesowej, zapewnienie integralności danych oraz dostarczanie interfejsów dla warstwy aplikacyjnej do manipulacji i odczytu informacji. System obejmuje zarządzanie użytkownikami (klienci, trenerzy, administratorzy), katalogiem ćwiczeń, grupami mięśniowymi, planami treningowymi (zarówno ogólnymi, jak i personalnymi przypisanymi do użytkowników), a także potencjalnie harmonogramami sesji i statystykami aktywności.

# 2. Realizacja projektu

Realizacja modułu bazodanowego systemu "Gymplify" opierała się na szczegółowym modelowaniu danych, co pozwoliło na stworzenie znormalizowanej i efektywnej struktury tabel w bazie Oracle. Kluczowa logika biznesowa, operacje CRUD (Create, Read, Update, Delete) dla poszczególnych encji, a także reguły walidacyjne i zapewnienie spójności danych zostały zaimplementowane bezpośrednio w bazie danych. Wykorzystano do tego celu pakiety PL/SQL, które enkapsulują powiązane ze sobą procedury i funkcje, oraz triggery bazodanowe, automatyzujące pewne procesy i egzekwujące reguły integralności w odpowiedzi na operacje DML. Takie podejście zapewnia wysoką wydajność, bezpieczeństwo i centralizację logiki blisko danych. Backend aplikacji Java Spring Boot komunikuje się z bazą danych poprzez wywoływanie tych procedur i funkcji PL/SQL.

# 3. Opis tabel

Poniżej przedstawiono opis kluczowych tabel w bazie danych systemu "Gymplify", które stanowią fundament dla jego funkcjonalności:

* **USERS (Użytkownicy)**  
  Rejestr wszystkich użytkowników systemu. Przechowuje dane logowania (USERNAME, PASSWORD\_HASH), kontaktowe (EMAIL), przypisaną rolę (ROLE, np. 'ADMIN', 'USER', 'TRAINER') oraz datę utworzenia konta (CREATED\_AT). Klucz USER\_ID (PK) jednoznacznie identyfikuje każdego użytkownika.
* **TRAINERS (Trenerzy)**  
  Ewidencja trenerów, powiązana z kontami użytkowników. Zawiera dane osobowe trenera (NAME, SURNAME), jego specjalizację (SPECIALIZATION) oraz dane kontaktowe (CONTACT). Każdy trener jest unikalnym użytkownikiem (USER\_ID - FK, UNIQUE) i posiada własny TRAINER\_ID (PK).
* **MUSCLE\_GROUPS (Grupy Mięśniowe)**  
  Tabela słownikowa definiująca grupy mięśniowe (np. "Klatka piersiowa", "Plecy"). Każda grupa posiada unikalny GROUP\_ID (PK), nazwę (GROUP\_NAME - UNIQUE) oraz opcjonalny, dłuższy opis (DESCRIPTION).
* **EXERCISES (Ćwiczenia)**  
  Katalog wszystkich ćwiczeń dostępnych w systemie. Dla każdego ćwiczenia (EXERCISE\_ID - PK) przechowywana jest nazwa (NAME), szczegółowy opis (DESCRIPTION) oraz powiązanie z konkretną grupą mięśniową (GROUP\_ID - FK do MUSCLE\_GROUPS).
* **TRAINING\_PLANS (Plany Treningowe)**  
  Przechowuje ogólne szablony planów treningowych. Każdy plan (PLAN\_ID - PK) posiada unikalną nazwę (NAME).
* **TRAINING\_EXERCISE (Ćwiczenia w Planie Treningowym)**  
  Tabela TRAINING\_EXERCISE jest tabelą łączącą, która definiuje przypisanie ćwiczeń (EXERCISE\_ID, klucz obcy) do konkretnych planów treningowych (PLAN\_ID, klucz obcy). Klucz główny złożony z kolumn (PLAN\_ID, EXERCISE\_ID) zapewnia unikalność każdego przypisania ćwiczenia do planu.
* **PERSONAL\_PLANS (Plany Personalne)**  
  Rejestruje przypisania planów treningowych (PLAN\_ID - FK) konkretnym użytkownikom (USER\_ID - FK), potencjalnie przez określonego trenera (TRAINER\_ID - FK). Każde takie przypisanie ma unikalny PERSONAL\_PLAN\_ID (PK).
* **TRAINER\_SESSIONS (Sesje Trenerów)**  
  Harmonogram sesji treningowych prowadzonych przez trenerów. Każdy wpis (SCHEDULE\_ID - PK) jest powiązany z trenerem (TRAINER\_ID - FK) i określa datę (SESSION\_DATE) oraz czas rozpoczęcia (START\_TIME) i zakończenia (END\_TIME) sesji.

Obraz zawierający tekst, numer, Czcionka, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

**Legenda i oznaczenia**

* **(Primary Key)** – klucz główny, unikalnie identyfikuje każdy rekord w tabeli.
* **(Foreign Key)** – klucz obcy, wskazuje na powiązanie z inną tabelą
* **1:1 (jeden do jednego) -** Jeden rekord z tabeli A jest powiązany z dokładnie jednym rekordem w tabeli B i odwrotnie.
* **1:N (jeden do wielu) -** Jeden rekord z tabeli A może być powiązany z wieloma rekordami w tabeli B, ale rekord z tabeli B może być powiązany tylko z jednym rekordem z tabeli A.
* **M:N (wiele do wielu) -** Rekordy z tabeli A mogą być powiązane z wieloma rekordami z tabeli B i odwrotnie.

**Relacje dla wszystkich tabel:**

* USERS → PERSONAL\_PLANS (1:N)
* USERS → TRAINERS (1:1)
* TRAINERS → PERSONAL\_PLANS (1:N)
* TRAINERS → TRAINER\_SESSIONS (1:N)
* MUSCLE\_GROUPS → EXERCISES (1:N)
* TRAINING\_PLANS → PERSONAL\_PLANS (1:N)
* TRAINING\_PLANS ↔ EXERCISES (M:N, przez TRAINING\_EXERCISE)

# 4. Główne funkcjonalności

Logika biznesowa i operacje na danych w systemie "Gymplify" zostały w dużej mierze zaimplementowane bezpośrednio w bazie danych Oracle za pomocą języka PL/SQL. Takie podejście zapewnia wysoką wydajność, spójność danych oraz centralizację reguł biznesowych. Kluczowe funkcjonalności zostały zgrupowane w pakiety PL/SQL, a integralność danych jest dodatkowo wspierana przez triggery bazodanowe. Poniżej przedstawiono główne obszary funkcjonalne realizowane przez moduł PL/SQL:

## 4.1. Zarządzanie Użytkownikami i Uprawnieniami (Pakiet PKG\_USER\_MGMT)

* **Ewidencja Kont:** Kompleksowe zarządzanie kontami użytkowników, w tym ich tworzenie z walidacją unikalności nazwy użytkownika i adresu email, odczyt szczegółowych informacji (w tym zaszyfrowanego hasła do celów uwierzytelniania przez warstwę aplikacji), aktualizacja danych profilowych oraz ról.
* **Bezpieczeństwo:** Procedury przyjmują zaszyfrowane hasło, podkreślając, że logika hashowania powinna być realizowana w warstwie aplikacyjnej przed przekazaniem danych do bazy.
* **Operacje CRUD:** Zapewnienie pełnego zestawu operacji tworzenia, odczytu, aktualizacji i usuwania użytkowników, z uwzględnieniem kaskadowego usuwania powiązanych danych (np. informacji o trenerze, jeśli użytkownik był trenerem).

## 4.2. Katalogowanie Struktur Treningowych (Pakiety PKG\_MUSCLE\_GROUP\_MGMT, PKG\_EXERCISE\_MGMT)

* **Zarządzanie Grupami Mięśniowymi:** Umożliwia tworzenie, modyfikację, usuwanie i przeglądanie informacji o grupach mięśniowych, stanowiących podstawę kategoryzacji ćwiczeń. Zapewniona jest unikalność nazw grup.
* **Ewidencja Ćwiczeń:** Kompleksowe zarządzanie katalogiem ćwiczeń, w tym ich dodawanie z przypisaniem do konkretnej grupy mięśniowej, walidacja istnienia grupy, aktualizacja opisów i nazw, oraz usuwanie ćwiczeń.

## 4.3. Zarządzanie Planami Treningowymi (Pakiet PKG\_TRAINING\_PLAN\_MGMT)

* **Tworzenie i Modyfikacja Planów:** Definiowanie ogólnych szablonów planów treningowych, zarządzanie ich nazwami z zapewnieniem unikalności.
* **Kompozycja Planów:** Przypisywanie ćwiczeń do konkretnych planów treningowych.
* **Integralność Danych:** Walidacja istnienia planów i ćwiczeń przed ich powiązaniem oraz obsługa usuwania ćwiczeń z planów. Trigger bazodanowy (TRG\_PREVENT\_PLAN\_DELETE\_IF\_LINKED) zapobiega usunięciu planu, jeśli jest on aktywnie wykorzystywany.

## 4.4. Zarządzanie Planami Personalnymi (Pakiet PKG\_PERSONAL\_PLAN\_MGMT)

* **Przypisywanie Planów Użytkownikom:** Umożliwia przypisanie ogólnego planu treningowego (z TRAINING\_PLANS) konkretnemu użytkownikowi, potencjalnie przez określonego trenera.
* **Walidacja Przypisań:** Zapewnienie, że użytkownik, trener (jeśli dotyczy) oraz plan treningowy istnieją w systemie przed dokonaniem przypisania.
* **Modyfikacja i Usuwanie:** Zarządzanie cyklem życia przypisanego planu personalnego, w tym możliwość jego aktualizacji lub usunięcia.
* **Trigger Walidacyjny (TRG\_PERSONAL\_PLANS\_VALIDATE):** Zapewnia, że podczas tworzenia lub aktualizacji planu personalnego, wszystkie powiązane encje (użytkownik, trener, plan ogólny) istnieją w bazie danych.

## 4.5. Statystyki i Raportowanie (Pakiet PKG\_APP\_STATISTICS)

* **Agregacja Danych:** Udostępnianie procedur i funkcji do zbierania zagregowanych danych statystycznych z systemu, takich jak całkowita liczba użytkowników, liczba użytkowników według ról, liczba nowych użytkowników w zadanym okresie, statystyki dotyczące trenerów (liczba, specjalizacje, obciążenie), popularność ćwiczeń w planach, liczba ćwiczeń według grup mięśniowych, oraz ogólne wskaźniki aktywności systemu.
* **Wsparcie dla Warstwy Aplikacyjnej:** Dane statystyczne są zwracane głównie za pomocą kursorów (SYS\_REFCURSOR), co ułatwia ich przetwarzanie i prezentację w warstwie aplikacji.

## 4.6. Automatyzacja i Zapewnienie Spójności Danych (Triggery)

Oprócz triggerów walidacyjnych (jak TRG\_PREVENT\_PLAN\_DELETE\_IF\_LINKED czy TRG\_PERSONAL\_PLANS\_VALIDATE), system może wykorzystywać inne triggery do automatyzacji pewnych zadań, np. aktualizacji daty modyfikacji, logowania zmian (audytu) czy bardziej złożonych reguł integralności, które nie są łatwo implementowalne za pomocą standardowych ograniczeń bazodanowych.

# 5. CRUD dla zarządzania użytkownikami

Pakiet PKG\_USER\_MGMT centralizuje logikę biznesową związaną z zarządzaniem kontami użytkowników w systemie Gymplify. Odpowiada za tworzenie nowych użytkowników, ich usuwanie, modyfikację danych oraz pobieranie informacji o użytkownikach na różne sposoby.

## 5.1 Tworzenie Użytkownika (AddUser)

* **Procedura:** AddUser
* **Parametry wejściowe:**
  + p\_username (TYP: USERS.USERNAME%TYPE): Nazwa nowego użytkownika.
  + p\_password\_hash (TYP: USERS.PASSWORD\_HASH%TYPE): Zaszyfrowane hasło użytkownika.
  + p\_email (TYP: USERS.EMAIL%TYPE): Adres email nowego użytkownika.
  + p\_role (TYP: USERS.ROLE%TYPE): Rola przypisana nowemu użytkownikowi.
* **Parametry wyjściowe:**
  + p\_user\_id (TYP: USERS.USER\_ID%TYPE): ID nowo utworzonego użytkownika.
  + p\_success (TYP: BOOLEAN): Flaga informująca o powodzeniu operacji (TRUE - sukces, FALSE - błąd).
* **Logika:**
  + **Walidacja Danych:** Sprawdza, czy przekazana nazwa użytkownika (p\_username) oraz email (p\_email) nie są puste po usunięciu białych znaków. Email jest konwertowany na małe litery, a rola na wielkie litery. Weryfikuje również, czy podane p\_password\_hash i p\_role nie są puste oraz czy rola jest jedną z dopuszczalnych wartości ('ADMIN', 'USER', 'TRAINER').
  + **Generowanie ID:** Wykorzystuje sekwencję users\_seq do wygenerowania unikalnego user\_id.
  + **Operacja SQL:** Wykonuje instrukcję INSERT INTO USERS (user\_id, username, password\_hash, email, role, created\_at) VALUES (p\_user\_id, v\_username\_trimmed, p\_password\_hash, v\_email\_trimmed, v\_role\_trimmed, SYSTIMESTAMP), wstawiając nowego użytkownika do bazy danych.
  + **Obsługa Błędów Unikalności:** Przechwytuje wyjątek DUP\_VAL\_ON\_INDEX (np. gdy nazwa użytkownika lub email już istnieje). Transakcja jest wycofywana, p\_success ustawiane na FALSE, a odpowiedni komunikat o błędzie jest przygotowywany (choć w obecnej implementacji wypisywany przez DBMS\_OUTPUT).
  + **Ogólna Obsługa Błędów:** Inne błędy Oracle są przechwytywane, transakcja jest wycofywana, p\_success ustawiane na FALSE.
  + **Transakcja:** Jawnie zarządza transakcją poprzez COMMIT po pomyślnym wstawieniu lub ROLLBACK w przypadku błędu.
* **Cel:** Umożliwia bezpieczne dodanie nowego użytkownika do systemu z walidacją danych wejściowych i obsługą konfliktów unikalności.

## 5.2 Usuwanie Użytkownika (DeleteUser)

* **Procedura:** DeleteUser
* **Parametr wejściowy:**
  + p\_user\_id (TYP: USERS.USER\_ID%TYPE): ID użytkownika do usunięcia.
* **Parametr wyjściowy:**
  + p\_success (TYP: BOOLEAN): Flaga powodzenia operacji.
* **Logika:**
  + **Walidacja ID:** Sprawdza, czy p\_user\_id nie jest NULL i czy użytkownik o podanym ID istnieje.
  + **Obsługa Zależności (Trener):** Jeśli usuwany użytkownik jest również trenerem (ma wpis w tabeli TRAINERS), procedura najpierw usuwa powiązane z tym trenerem rekordy z tabel TRAINER\_SESSIONS i PERSONAL\_PLANS, a następnie usuwa wpis z tabeli TRAINERS.
  + **Obsługa Zależności (Użytkownik):** Usuwa powiązane rekordy z tabel PERSONAL\_PLANS dla danego user\_id. *Uwaga: Usunięcie z WEIGHT\_MEASUREMENTS i WEIGHT\_LEADERBOARD zostało usunięte z przykładu, dostosuj jeśli te tabele istnieją i mają zależności.*
  + **Operacja SQL:** Wykonuje DELETE FROM USERS WHERE user\_id = p\_user\_id.
  + **Sprawdzenie Wyniku:** Weryfikuje SQL%ROWCOUNT, aby upewnić się, że użytkownik został faktycznie usunięty.
  + **Transakcja:** Używa SAVEPOINT dla częściowej obsługi błędów oraz COMMIT na końcu udanej operacji lub ROLLBACK w głównym bloku wyjątku.
* **Cel:** Usunięcie użytkownika z systemu wraz z powiązanymi z nim danymi w tabelach zależnych, zapewniając spójność danych.

## 5.3 Odczyt Użytkowników

* **Odczyt wszystkich użytkowników (GetAllUsers)**
  + **Procedura:** GetAllUsers
  + **Parametry wyjściowe:**
    - p\_users\_cursor (TYP: SYS\_REFCURSOR): Kursor zawierający listę wszystkich użytkowników.
    - p\_success (TYP: BOOLEAN): Flaga powodzenia.
  + **Logika:** Otwiera kursor p\_users\_cursor dla zapytania SELECT user\_id, username, email, role, password\_hash, created\_at FROM USERS ORDER BY username. W przypadku braku danych lub błędu, otwiera pusty kursor, ustawiając p\_success odpowiednio.
  + **Cel:** Dostarczenie pełnej listy użytkowników systemu, posortowanej alfabetycznie. *Uwaga: Zwracanie password\_hash w tym widoku może być kwestią bezpieczeństwa do przemyślenia.*
* **Odczyt użytkownika po emailu (GetUserByEmailFunc - Funkcja)**
  + **Funkcja:** GetUserByEmailFunc
  + **Parametr wejściowy:** p\_email (TYP: USERS.EMAIL%TYPE).
  + **Zwraca:** SYS\_REFCURSOR zawierający dane użytkownika o podanym emailu (lub pusty kursor).
  + **Logika:** Konwertuje p\_email na małe litery, usuwa białe znaki, następnie otwiera kursor z danymi użytkownika.
  + **Cel:** Umożliwia wyszukanie użytkownika na podstawie jego adresu email.
* **Odczyt użytkownika po emailu (GetUserByEmailProc - Procedura)**
  + **Procedura:** GetUserByEmailProc
  + **Parametry wejściowe:** p\_email (TYP: USERS.EMAIL%TYPE).
  + **Parametry wyjściowe:** p\_user\_cursor (TYP: SYS\_REFCURSOR), p\_success (TYP: BOOLEAN).
  + **Logika:** Podobna do GetUserByEmailFunc, ale zwraca status powodzenia przez parametr OUT.
  + **Cel:** Wyszukanie użytkownika po adresie email z dodatkową informacją o powodzeniu operacji.
* **Odczyt użytkownika po ID (GetUserById)**
  + **Procedura:** GetUserById
  + **Parametry wejściowe:** p\_user\_id (TYP: USERS.USER\_ID%TYPE).
  + **Parametry wyjściowe:** p\_user\_cursor (TYP: SYS\_REFCURSOR), p\_success (TYP: BOOLEAN).
  + **Logika:** Waliduje p\_user\_id i sprawdza istnienie użytkownika. Jeśli użytkownik istnieje, otwiera kursor z jego danymi. W przeciwnym razie zwraca pusty kursor i ustawia p\_success na FALSE.
  + **Cel:** Pobranie szczegółowych informacji o konkretnym użytkowniku na podstawie jego ID.

## 5.4 Aktualizacja Danych Użytkownika

* **Aktualizacja szczegółów użytkownika (UpdateUserDetails)**
  + **Procedura:** UpdateUserDetails
  + **Parametry wejściowe:**
    - p\_user\_id (TYP: USERS.USER\_ID%TYPE): ID użytkownika do aktualizacji.
    - p\_username (TYP: USERS.USERNAME%TYPE): Nowa nazwa użytkownika.
    - p\_email (TYP: USERS.EMAIL%TYPE): Nowy adres email.
  + **Parametr wyjściowy:** p\_success (TYP: BOOLEAN).
  + **Logika:**
    - **Walidacja:** Sprawdza istnienie użytkownika oraz czy nowe wartości username i email nie są puste i nie przekraczają dozwolonych długości. Email jest konwertowany na małe litery.
    - **Operacja SQL:** Wykonuje UPDATE USERS SET username = ..., email = ... WHERE user\_id = ....
    - **Obsługa Błędów Unikalności:** Przechwytuje DUP\_VAL\_ON\_INDEX i ustawia p\_success na FALSE.
    - **Transakcja:** Zarządza transakcją.
  + **Cel:** Modyfikacja nazwy użytkownika i adresu email dla istniejącego konta, z walidacją danych.
* **Aktualizacja roli użytkownika (UpdateUserRole)**
  + **Procedura:** UpdateUserRole
  + **Parametry wejściowe:**
    - p\_user\_id (TYP: USERS.USER\_ID%TYPE): ID użytkownika.
    - p\_new\_role (TYP: USERS.ROLE%TYPE): Nowa rola dla użytkownika.
  + **Parametr wyjściowy:** p\_success (TYP: BOOLEAN).
  + **Logika:**
    - **Walidacja:** Sprawdza istnienie użytkownika oraz czy nowa rola nie jest pusta i czy jest jedną z dozwolonych wartości. Rola jest konwertowana na wielkie litery.
    - **Operacja SQL:** Wykonuje UPDATE USERS SET role = ... WHERE user\_id = ....
    - **Transakcja:** Zarządza transakcją.
  + **Cel:** Zmiana przypisanej roli dla istniejącego użytkownika, z walidacją poprawności nowej roli.

# 6. Dodatkowe funkcjonalności

Oprócz podstawowych operacji CRUD dla poszczególnych tabel, system "Gymplify" wykorzystuje dedykowane procedury, funkcje oraz triggery bazodanowe PL/SQL do realizacji bardziej złożonych operacji biznesowych, automatyzacji procesów oraz zapewnienia integralności danych. Poniżej opisano wybrane przykłady:

## 6.1 Trigger (TRG\_PREVENT\_PLAN\_DELETE\_IF\_LINKED)

* CREATE OR REPLACE TRIGGER trg\_prevent\_plan\_delete\_if\_linked  
   BEFORE DELETE  
   ON training\_plans  
   FOR EACH ROW  
  DECLARE  
   v\_exercise\_count NUMBER;  
   v\_personal\_plan\_count NUMBER;  
  BEGIN  
   SELECT *COUNT*(*\**)  
   INTO v\_exercise\_count  
   FROM training\_exercise  
   WHERE plan\_id = :OLD.**plan\_id**;  
    
   IF v\_exercise\_count > 0 THEN  
   *RAISE\_APPLICATION\_ERROR*(-20303, 'Nie można usunąć planu treningowego ID ' || :OLD.plan\_id ||  
   '. Najpierw usuń wszystkie przypisane do niego ćwiczenia (' ||  
   v\_exercise\_count || ').');  
   END IF;  
    
   SELECT *COUNT*(*\**)  
   INTO v\_personal\_plan\_count  
   FROM personal\_plans  
   WHERE plan\_id = :OLD.**plan\_id**;  
    
   IF v\_personal\_plan\_count > 0 THEN  
   *RAISE\_APPLICATION\_ERROR*(-20304, 'Nie można usunąć planu treningowego ID ' || :OLD.plan\_id ||  
   '. Jest on przypisany do użytkowników (' || v\_personal\_plan\_count || ').');  
   END IF;  
  END;  
  /
* **Cel:** Kluczowy mechanizm ochrony integralności referencyjnej. Jego zadaniem jest uniemożliwienie usunięcia rekordu z tabeli TRAINING\_PLANS (reprezentującego ogólny szablon planu treningowego), jeśli ten plan jest aktywnie wykorzystywany w systemie. Zapobiega to osieroceniu rekordów w tabelach zależnych.
* **Logika:**
  + Trigger jest uruchamiany automatycznie **przed** każdą operacją usunięcia (DELETE) na tabeli TRAINING\_PLANS, dla każdego indywidualnego wiersza, który ma zostać usunięty (FOR EACH ROW). Pseudorekord :OLD odnosi się do wartości wiersza przed operacją usunięcia.
  + Najpierw trigger sprawdza, czy w tabeli TRAINING\_EXERCISE (tabela łącząca plany z ćwiczeniami) istnieją jakiekolwiek wpisy powiązane z plan\_id usuwanego planu. Jeśli v\_exercise\_count jest większy od zera, oznacza to, że plan zawiera przypisane ćwiczenia. W takim przypadku trigger przerywa operację usunięcia, zgłaszając błąd aplikacji (RAISE\_APPLICATION\_ERROR z kodem -20303) i informując użytkownika o konieczzości wcześniejszego usunięcia powiązanych ćwiczeń.
  + Następnie, jeśli pierwsze sprawdzenie nie wykazało zależności, trigger weryfikuje, czy usuwany plan jest przypisany do jakichkolwiek użytkowników w tabeli PERSONAL\_PLANS. Jeśli v\_personal\_plan\_count jest większy od zera, trigger również przerywa operację usunięcia, zgłaszając błąd (RAISE\_APPLICATION\_ERROR z kodem -20304) i informując o istniejących przypisaniach do użytkowników.
  + Dopiero jeśli oba sprawdzenia nie wykażą żadnych aktywnych powiązań, trigger kończy działanie, a operacja DELETE na tabeli TRAINING\_PLANS może być kontynuowana.
* **Typ:** Trigger walidacyjny, działający na poziomie wiersza, kluczowy dla utrzymania spójności danych w relacjach między planami treningowymi, ćwiczeniami w tych planach oraz przypisaniami tych planów do użytkowników.

## 6.2 Procedura (PKG\_TRAINING\_PLAN\_MGMT.AddExerciseToPlan)

PROCEDURE *AddExerciseToPlan*(p\_plan\_id IN training\_exercise.plan\_id%TYPE,  
 p\_exercise\_id IN training\_exercise.exercise\_id%TYPE) AS  
 v\_is\_valid BOOLEAN;  
BEGIN  
 v\_is\_valid := *fn\_validate\_plan\_exercise*(p\_plan\_id, p\_exercise\_id);  
 INSERT INTO training\_exercise (plan\_id, exercise\_id) VALUES (p\_plan\_id, p\_exercise\_id);  
 COMMIT;  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN ROLLBACK;  
 *RAISE\_APPLICATION\_ERROR*(-20305,  
 'Ćwiczenie ID ' || p\_exercise\_id || ' jest już przypisane do planu ID ' || p\_plan\_id ||  
 '.');  
 WHEN OTHERS THEN ROLLBACK; RAISE;  
END *AddExerciseToPlan*;

* **Cel:** Umożliwia przypisanie istniejącego ćwiczenia do istniejącego planu treningowego.
* **Parametry wejściowe:** p\_plan\_id (ID planu), p\_exercise\_id (ID ćwiczenia).
* **Logika:**
  1. Waliduje istnienie podanego planu treningowego oraz ćwiczenia w bazie danych za pomocą wewnętrznej funkcji fn\_validate\_plan\_exercise. W przypadku braku którejkolwiek z encji, zgłaszany jest błąd aplikacji.
  2. Wykonuje operację INSERT do tabeli TRAINING\_EXERCISE, tworząc powiązanie między planem a ćwiczeniem.
  3. Zatwierdza transakcję (COMMIT) po pomyślnym dodaniu.
  4. Obsługuje potencjalny błąd naruszenia unikalności (DUP\_VAL\_ON\_INDEX), jeśli próbuje się dodać ćwiczenie, które już jest w danym planie, zgłaszając odpowiedni błąd aplikacji. Inne błędy Oracle są również przechwytywane i propagowane.

## 6.3 Zaawansowane Funkcje Analityczne (Pakiet PKG\_APP\_STATISTICS)

Jednym z kluczowych przykładów dodatkowych, zaawansowanych funkcjonalności realizowanych w module bazodanowym "Gymplify" jest pakiet PKG\_APP\_STATISTICS. Jego głównym zadaniem jest dostarczanie kompleksowych, zagregowanych danych statystycznych i analitycznych, które wykraczają poza proste operacje CRUD na pojedynczych tabelach. Pakiet ten umożliwia głębszą analizę danych systemowych, wspierając podejmowanie decyzji biznesowych, monitorowanie trendów oraz ocenę efektywności działania siłowni i zaangażowania użytkowników.

Pełny opis wszystkich funkcji i procedur zawartych w pakiecie PKG\_APP\_STATISTICS znajduje się w sekcji **4.5. Statystyki i Raportowanie**. Poniżej przedstawiono przykład jednej z bardziej złożonych funkcji tego pakietu, ilustrującej jego możliwości:

* **Analiza Obciążenia Trenerów (PKG\_APP\_STATISTICS.GetTrainerWorkloadStats\_Typed)**
  + **Cel:** Dostarczenie szczegółowych danych na temat obciążenia pracą i zaangażowania poszczególnych trenerów. Informacje te mogą być kluczowe dla zarządzania zasobami ludzkimi, planowania grafików, identyfikacji najbardziej i najmniej obciążonych trenerów oraz oceny ich efektywności.
  + **Typ:** Funkcja
  + **Parametry wejściowe:** Brak
  + **Zwracany typ:** SYS\_REFCURSOR
  + **Logika:**
    1. Funkcja łączy dane z tabel TRAINERS, PERSONAL\_PLANS oraz TRAINER\_SESSIONS.
    2. Dla każdego trenera oblicza:
       - Imię i nazwisko (trainer\_full\_name).
       - Specjalizację (specialization).
       - Liczbę unikalnych klientów przypisanych do trenera poprzez plany personalne (assigned\_clients\_count).
       - Całkowitą liczbę zaplanowanych sesji treningowych (total\_scheduled\_sessions).
       - Liczbę sesji zaplanowanych w ciągu najbliższych 7 dni (sessions\_next\_7\_days).
    3. Wykorzystuje lewe złączenia (LEFT JOIN) oraz podzapytania agregujące (COUNT, GROUP BY), aby zapewnić, że wszyscy trenerzy są uwzględnieni, nawet jeśli nie mają przypisanych klientów lub zaplanowanych sesji (w takich przypadkach wartości liczbowe będą wynosić 0 dzięki użyciu NVL).
    4. Wyniki są zwracane jako kursor, posortowane malejąco według liczby przypisanych klientów, a następnie alfabetycznie według nazwiska trenera.
  + **Znaczenie:** Ta funkcja jest przykładem, jak PL/SQL może być wykorzystany do przygotowania złożonych zestawień danych, które wymagają agregacji informacji z wielu powiązanych tabel. Zamiast obciążać warstwę aplikacyjną wieloma zapytaniami i logiką łączenia danych, baza danych dostarcza gotowy, przetworzony zestaw informacji, co jest bardziej efektywne i upraszcza logikę po stronie aplikacji.

Inne funkcje w pakiecie PKG\_APP\_STATISTICS, takie jak identyfikacja najpopularniejszych ćwiczeń (GetMostPopularExercisesInPlans\_Typed) czy najczęściej przypisywanych planów treningowych (GetMostAssignedTrainingPlans), również demonstrują zdolność modułu PL/SQL do dostarczania wartościowych informacji analitycznych, które wspierają zarządzanie siłownią "Gymplify".

## 6.4 Sekwencja (USER\_SEQ)

Sekwencja USERS\_SEQ, służy do generowania unikalnych identyfikatorów dla tabeli USERS. Jej celem jest zapewnienie automatycznej, rosnącej wartości user\_id dla każdego nowego użytkownika w systemie. Sekwencja ta zaczyna się od wartości 1 i zwiększa się o 1 przy każdym użyciu, nie używa buforowania (NOCACHE), co zapewnia większą niezawodność w przypadku awarii systemu, oraz nie zawija się z powrotem do początku po osiągnięciu maksymalnej wartości (NOCYCLE), co zapobiega duplikatom identyfikatorów.

v\_success := *fn\_add\_sequence*(  
 'CREATE SEQUENCE users\_seq START WITH 1 INCREMENT BY 1 NOCACHE NOCYCLE',  
 'USERS\_SEQ'  
 );  
DBMS\_OUTPUT.*PUT\_LINE*('PRC\_INITIALIZE\_DATABASE: Wynik USERS\_SEQ: ' ||  
 CASE WHEN v\_success THEN 'OK' ELSE 'BŁĄD/ISTNIEJE' END);

## 6.5 Typy złożone (PKG\_PERSONAL\_PLAN\_MGMT)

W pakiecie PKG\_PERSONAL\_PLAN\_MGMT zastosowano typy złożone PL/SQL, które służą do przechowywania oraz zwracania bardziej złożonych struktur danych związanych z przypisaniami planów treningowych użytkownikom. Typy te zostały zadeklarowane w specyfikacji pakietu i są wykorzystywane głównie przez funkcje zwracające zestawy danych powiązanych z użytkownikami, trenerami oraz planami treningowymi.

Przykładowym typem złożonym jest ty\_personal\_plan\_record, który jest rekordem (czyli strukturą danych zawierającą pola różnych typów). Został on zdefiniowany w następujący sposób:

TYPE ty\_personal\_plan\_record IS RECORD  
 (  
 personal\_plan\_id personal\_plans.personal\_plan\_id%TYPE,  
 user\_id personal\_plans.user\_id%TYPE,  
 username users.username%TYPE,  
 trainer\_id personal\_plans.trainer\_id%TYPE,  
 trainer\_name trainers.name%TYPE,  
 trainer\_surname trainers.surname%TYPE,  
 plan\_id personal\_plans.plan\_id%TYPE,  
 plan\_name training\_plans.name%TYPE   
 );

Typ ten grupuje dane z kilku tabel (personal\_plans, users, trainers, training\_plans) w jeden logiczny rekord, który reprezentuje przypisanie planu użytkownikowi przez trenera. Taki rekord może być następnie wykorzystany w kodzie pakietu do zwrócenia pełnych informacji o przypisaniu.

Przykładem użycia tego typu w kodzie jest funkcja GetPersonalPlanById, która zwraca rekord typu ty\_personal\_plan\_record na podstawie podanego identyfikatora przypisania:

FUNCTION *GetPersonalPlanById*(  
 p\_personal\_plan\_id IN personal\_plans.personal\_plan\_id%TYPE  
) RETURN ty\_personal\_plan\_record AS  
 v\_plan\_rec ty\_personal\_plan\_record;  
BEGIN  
 SELECT pp.personal\_plan\_id,  
 pp.user\_id,  
 u.username,  
 pp.trainer\_id,  
 t.name,  
 t.surname,  
 pp.plan\_id,  
 tp.name  
 INTO  
 v\_plan\_rec.personal\_plan\_id, v\_plan\_rec.user\_id, v\_plan\_rec.username,  
 v\_plan\_rec.trainer\_id, v\_plan\_rec.trainer\_name, v\_plan\_rec.trainer\_surname,  
 v\_plan\_rec.plan\_id, v\_plan\_rec.plan\_name  
 FROM personal\_plans pp  
 JOIN users u ON pp.user\_id = u.user\_id  
 JOIN trainers t ON pp.trainer\_id = t.trainer\_id  
 JOIN training\_plans tp ON pp.plan\_id = tp.plan\_id  
 WHERE pp.personal\_plan\_id = p\_personal\_plan\_id;  
 RETURN v\_plan\_rec;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 *RAISE\_APPLICATION\_ERROR*(-20401, 'Personalny plan o ID ' || p\_personal\_plan\_id || ' nie istnieje.');  
 WHEN OTHERS THEN  
 RAISE;  
END *GetPersonalPlanById*;

W tym fragmencie kodu zmienna v\_plan\_rec jest typu ty\_personal\_plan\_record, a jej pola są wypełniane danymi pobranymi z zapytania SQL łączącego cztery tabele. Funkcja ta pozwala zwrócić kompletne informacje o przypisaniu planu, gotowe do dalszego wykorzystania np. w interfejsie użytkownika czy raporcie.

# 7. Prezentacja Interfejsu Oraz Elementów Backendu

## 7.1 Prezentacja przykładowego CRUD’a dla użtykownika.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

*Rysunek 1. Panel zarządzania użytkownikami*

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

*Rysunek 2. Panel dodawania użytkownika*

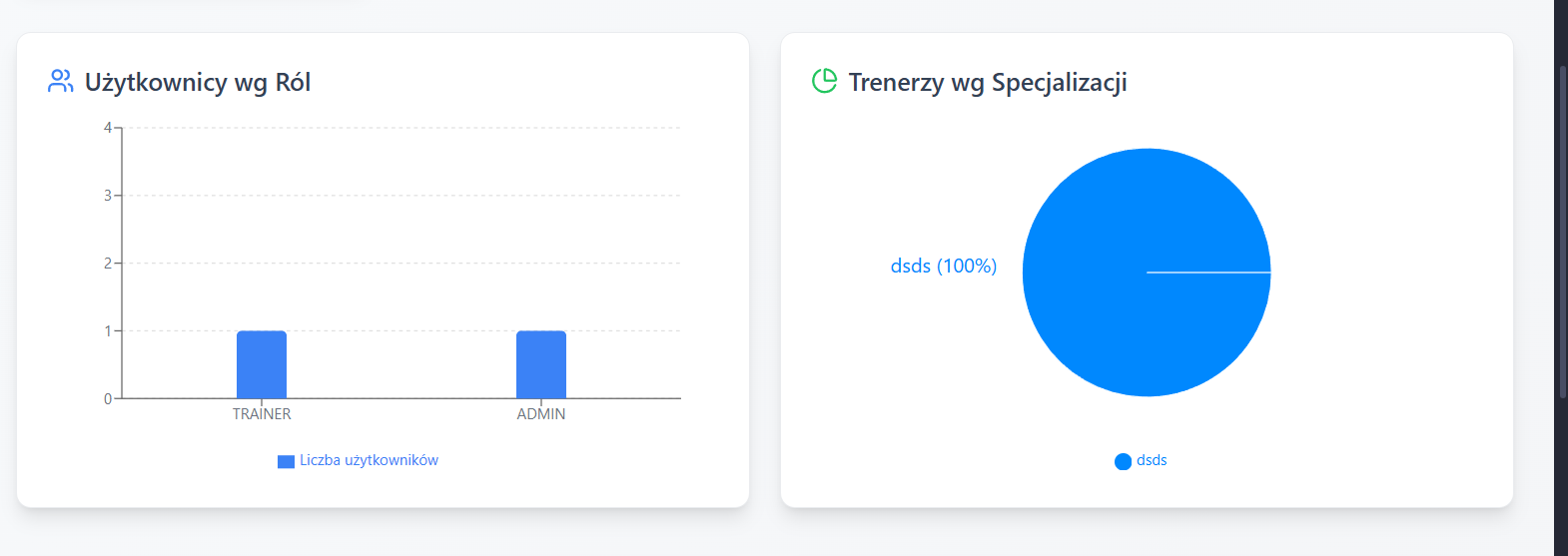
Obraz zawierający tekst, oprogramowanie, zrzut ekranu, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

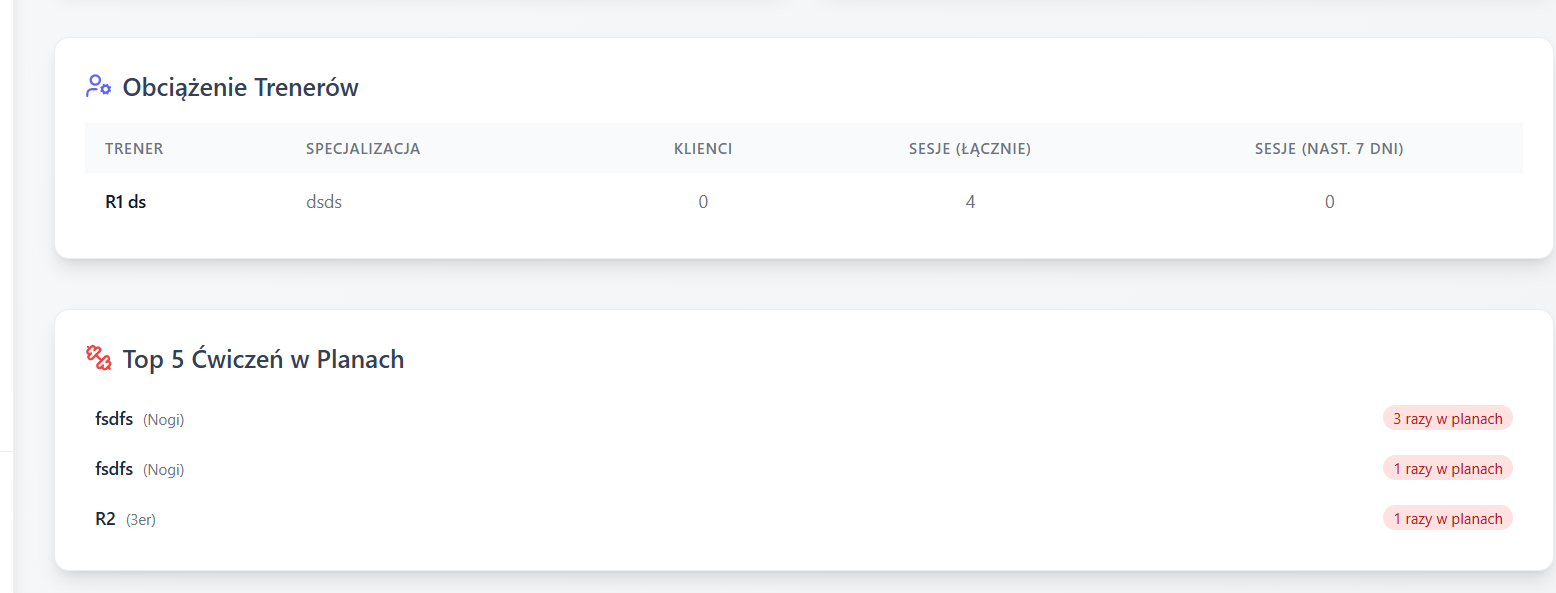
*Rysunek 3. Panel edycji użytkownikami*

## 7.2 Prezentacja panelu statystyk.





*Rysunek 1. Panel statystyk*



*Rysunek 2. Panel statystyk*

# 8. Podsumowanie

Projekt "Gymplify" ma na celu stworzenie wszechstronnego systemu informatycznego do zarządzania siłownią i wspierania aktywności fizycznej jej użytkowników. Centralnym elementem architektury jest moduł bazodanowy zrealizowany w technologii Oracle PL/SQL, który pełni rolę repozytorium danych oraz silnika logiki biznesowej.

**Kluczowe cechy i funkcjonalności modułu bazodanowego:**

1. **Model Danych:** Zastosowano znormalizowany model danych obejmujący użytkowników (USERS), trenerów (TRAINERS), grupy mięśniowe (MUSCLE\_GROUPS), ćwiczenia (EXERCISES), ogólne plany treningowe (TRAINING\_PLANS), powiązania ćwiczeń z planami (TRAINING\_EXERCISE), plany personalne użytkowników (PERSONAL\_PLANS) oraz harmonogramy sesji trenerów (TRAINER\_SESSIONS).
2. **Enkapsulacja Logiki Biznesowej:** Kluczowe operacje CRUD oraz bardziej złożone reguły biznesowe zostały zaimplementowane jako procedury i funkcje w pakietach PL/SQL. Takie podejście zapewnia:
   * **Centralizację:** Logika znajduje się blisko danych, co ułatwia zarządzanie i utrzymanie.
   * **Wydajność:** Operacje wykonywane bezpośrednio w bazie danych są często szybsze niż przetwarzanie danych w warstwie aplikacji.
   * **Bezpieczeństwo:** Kontrolowany dostęp do danych poprzez zdefiniowane interfejsy (procedury/funkcje).
3. **Zarządzanie Użytkownikami (PKG\_USER\_MGMT):** Pakiet ten oferuje kompleksowe zarządzanie kontami użytkowników, w tym ich tworzenie, odczyt (pojedynczo, wszystkich, po emailu), aktualizację danych i ról oraz usuwanie z obsługą zależności (np. powiązanych danych trenera).
4. **Zarządzanie Strukturami Treningowymi (PKG\_MUSCLE\_GROUP\_MGMT, PKG\_EXERCISE\_MGMT, PKG\_TRAINING\_PLAN\_MGMT):** Pakiety te umożliwiają zarządzanie katalogiem grup mięśniowych, ćwiczeń oraz szablonami planów treningowych, w tym przypisywanie ćwiczeń do planów.
5. **Zarządzanie Planami Personalnymi (PKG\_PERSONAL\_PLAN\_MGMT):** Pozwala na przypisywanie ogólnych planów treningowych konkretnym użytkownikom, z opcjonalnym wskazaniem trenera.
6. **Statystyki i Raportowanie (PKG\_APP\_STATISTICS):** Dedykowany pakiet do agregacji danych statystycznych, takich jak liczba użytkowników według ról, popularność ćwiczeń, obciążenie trenerów czy ogólne wskaźniki aktywności systemu. Dane są udostępniane głównie poprzez kursory (SYS\_REFCURSOR) dla warstwy aplikacyjnej.
7. **Integralność Danych:** Zapewniana przez ograniczenia bazodanowe (klucze główne, obce, unikalne) oraz dodatkowo przez triggery (np. TRG\_PREVENT\_PLAN\_DELETE\_IF\_LINKED zapobiegający usunięciu planu, jeśli jest powiązany) i logikę walidacyjną w procedurach.
8. **Interfejs dla Aplikacji Backendowej:** Moduł PL/SQL dostarcza jasno zdefiniowany interfejs programistyczny (API) dla aplikacji backendowej (np. Java Spring Boot), która komunikuje się z bazą poprzez wywoływanie procedur i funkcji. Dokumentacja zawiera wskazówki dotyczące integracji, w tym konfiguracji SimpleJdbcCall w Spring JDBC oraz rozwiązywania typowych problemów (np. ORA-17041).