POLITECHNIKA WROCŁAWSKA
KATEDRA INFORMATYKI TECHNICZNEJ

BAZY DANYCH 2
KARTA PROJEKTU

LIGA KOSZYKARSKA

AUTORZY:

RADOSŁAW LIS 241385
MACIEJ BIAŁKOWSKI 241285
KAMIL PICHETA 235882
WT-TN-13

PROJEKT
PROWADZĄCY - DR INŻ. ROMAN PTAK

WROCŁAW, DN. 22 PAŹDZIERNIKA 2019

Spis treści

1 Wstęp

1.1 Otoczenie systemu

Projekt zakłada stworzenie systemu do zarządzania ligą koszykarską. System ten ma być systemem uniwersalnym dla dowolnej ligi koszykarskiej. Będzie on pozwalał na zapisywanie wyników meczów w lidze, zdarzeń na tych meczach czy różnych statystyk. Dane te będą zapisywane w bazie danych. Dzięki aplikacji dostępowej kibice będą mogli śledzić interesujące ich dane, a administrator będzie mógł te dane zmieniać.

Aplikacja wykorzysta klasyczną budowę aplikacji z biblioteką *SWING*. Za pomoca rozsuwanych menu umieszczonych w belce znajdującej się na górze panelu aplikacji będzie można poruszać się po różnych widokach. Możliwe będzie otrzymywanie tabel w scrollowalnej postaci, zaś wszystkie akcje będa podejmowane za pomocą przycisków z wykorzystaniem list rozwijanych w celu podjęcia bardziej złożonego wyboru jak np. wybór drużyny. Wiele akcji wymusi otworzenie okien dialogowych, w których będziemy mogli podjąć pewne akcje lub zobaczyć oczekiwane rezultaty. Najbardziej rozbudowane okno dialogowe służące do aktualizowania meczu w trakcie jego trwania będzie bazowało na przyciskach zwiększających daną statystykę. Po zatwierdzeniu formularza mecz zostanie zakończony i oficjalnie zakończy to dodawanie statystyk meczu. Po jego zatwierdzeniu edycja będzie możliwa.

Aplikacja pozwala kibicowi przeglądać informacje o lidze koszykówki. Może on wyszukiwać drużyny i przeglądać ich rozpiskę meczy oraz statystyki poszczególnych zawodników. Może też przeglądać wyniki meczy już zakończonych. Administrator aplikacji rozszerza możliwości zwyklego użytkownika, o edycje danych zawartych w bazie. Statystyki graczy zostają podliczane na podstawie historii meczy zawartych w tabeli. Każdy mecz może być obsługiwany na żywo, tj. aplikacja pozwala śledzić rozgrywkę w trakcie jej trwania pod warunkiem że administrator na bieżąco aktualizuje dane o trafieniach i zdarzeniach na meczu.

Program jest standardowa aplikacją napisaną w języku Java. Komponenty graficzne ułożone są z wykorzystaniem graficznej biblioteki swing z wykorzystaniem buildera. Aplikacja łączy się z baza danych której serwer zostanie postawiony za pomocą Oracle Express Edition. W celu ułatwienia komunikacji z bazą danych poza Java Data Base Connectivity wykorzystamy również framework Hibernate służący do mapowania obiektów w bazie danych na struktury.

1.2 Wymagania funkcjonalne

- Kibic może przeglądać wyniki meczów i zdarzenia które miały miejsca na boisku.
- Kibic może przeglądać statystykidanej drużyny (filtrując je i sortując) lub poszczególnych zawodników.
- Administator tworząc dany mecz może do jednej drużyny dodać maksymalnie dwunastu zawodników wraz z odpowiadającymi im numerami na boisku oraz trenera. Musi też wybrać którzy zawodnicy będą rozpoczynać mecz.
- Administrator może dodać rzut, zgodnie z następującymi założeniami:
  - na początku trzeba określić czy rzut był za 1, za 2 czy za 3,
  - następnie określić czy rzut był trafiony czy nietrafiony,
  - jeśli rzut trafiony - dodanie punktów zawodnikowi (oraz drużynie).
- Administrator może dodać zdarzenie w meczu ustalając czas w którym miało ono miejsce. Zdarzeniami mogą być na przykład:
  - faul danego zawodnika
  - aut
  - strata piłki
  - przechwyt piłki
  - przerwa
  - wykluczenie zawodnika
  - zmiana zawodnika

  Ustala on wtedy kto brał udział w zdarzeniu (zawodnik/drużyna).
- Administrator może edytować rzut.
- Administrator może edytować zdarzenie.
- Administrator może edytować mecz.
- Administrator może usunąć rzut.
- Administrator może usunąć zdarzenie.
- Administrator może usunąć mecz.
- System generuje statystyki graczy i drużyn w oparciu o zdarzenia.

1.3 Zakres

Ze względu na dużą różnorodność zdarzeń, które mogą wystąpić na meczu, nasza aplikacja może posiadać dużo pomniejszych tabel reprezentujących wszystkie możliwe zdarzenia. Rozważenie wszystkich możliwości stworzy dużą ilość rekordów i rozbuduje aplikacje dostępową o dużą ilość powtarzalnego kodu, dlatego na potrzeby implementacji ograniczymy się tylko do wybranych zdarzeń(planowany rozkład tabel w punkcie 2).

1.4 Wymagania niefunkcjonalne

- Aplikacja będzie wykorzystywać technologię Java SE 9,
- Grafika aplikacji zaprojektowana z wykorzystaniem biblioteki Swing
- Mapowanie struktur za pomocą frameworku Hibernate,
- Baza danych zostanie utworzona wykorzystując Oracle XE,
- Skrypt tworzący bazę danych będzie napisany w języku SQL,
- Tylko administrator może dokonywać zmian w bazie danych.

1.5 Analiza Liczności

Przewidywana ilość danych:

- Liczba drużyn - 10
- Liczba trenerów - 10
- Liczba zawodników - 10*15=150 (po 15 w szerokim składzie)
- Liczba meczy - (T1 vs T2) * 2 = (9+8+...+2+1)*2 = 90
- Liczba rzutów - 93.5 (rzuty za dwa i za trzy), w tym 42.4 trafione i 28.6 (rzuty za 1), w tym 22.7 trafione). Czyli łącznie 136 rzutów w tym 67 trafionych.
- Liczba zbiórek - 69 (136-67)
- Liczba asyst - 29
- Liczba fauli - 24
- Liczba strat - 17
- Liczba przechwytów - 9
- Liczba bloków - 6
- Liczba zmian - 30
- Liczba czasów - 10
- Sumarycznie - (136+69+29+24+17+9+6+30+10)*90 = 29700 zdarzeń w sezonie. https://www.teamrankings.com/nba/stats/

Przewidywana liczba użytkowników:

- Zakładamy że liczba osób która w tym samym czasie chce korzystać z aplikacji, a tym samym odwoływać się do bazy danych może być dosyć duża, zaczynająca się od kilku tysięcy a kończąca nawet na kilkuset tysiącach. Liczba jest w dużej mierze uzależniona od ligi którą aplikacja zamiar obsługiwać.

1.5.1 Uzasadnienie doboru bazy

Po przeprowadzeniu wstępnej analizy zdecydowaliśmy się skorzystać w naszym programie z bazy danych firmy Oracle, która doskonale poradzi sobie z ilością danych jaką przewidujemy. Wykorzystanie wersji 'Express Edition' wydało nam się rozsądnym wyborem, ze względu na znajomość system.u Liczba użytkowników w skrajnych przypadkach może powodować opóźnienie w uzyskaniu wyników zapytań, jednakże z przeprowadzonej analizy uznaliśmy te bazę za odpowiednią.

2 Przewidywany rozkład tabel

2.1 Tabele

2.1.1 Tabele podstawowe

players	**teams**	**coaches**	**matches**	**incidents**	**referees**	**shots**
license_nrP	team_id	license_nrC	match_id	incident_id	license_nrR	incident_id
name	name	name	1team	match_id	name	shot_id
surname		surname	2team	incident_time	surname	player_id
team_id		team_id	first_ref	incident_quarter	birth_date	shot_type
jersey_nr		birth_date	second_ref			hit
birth_date			date			assist_pl_id

2.1.2 Tabele dotyczące poszczególnych zdarzeń

assists	**fouls**	**turnovers**	**steals**	**def_rebounds**	**off_rebounds**	**blocks**	**substitutions**	**timeouts**
shot_id	incident_id	incident_id	turnover_id	shot_id	shot_id	shot_id	incident_id	incident_id
player_id	foul_id	turnover_id	player_id	player_id	player_id	player_id	playerON_id	team_id
	player_id	player_id					playerOUT_id	
	player2_id							
	foul_type							

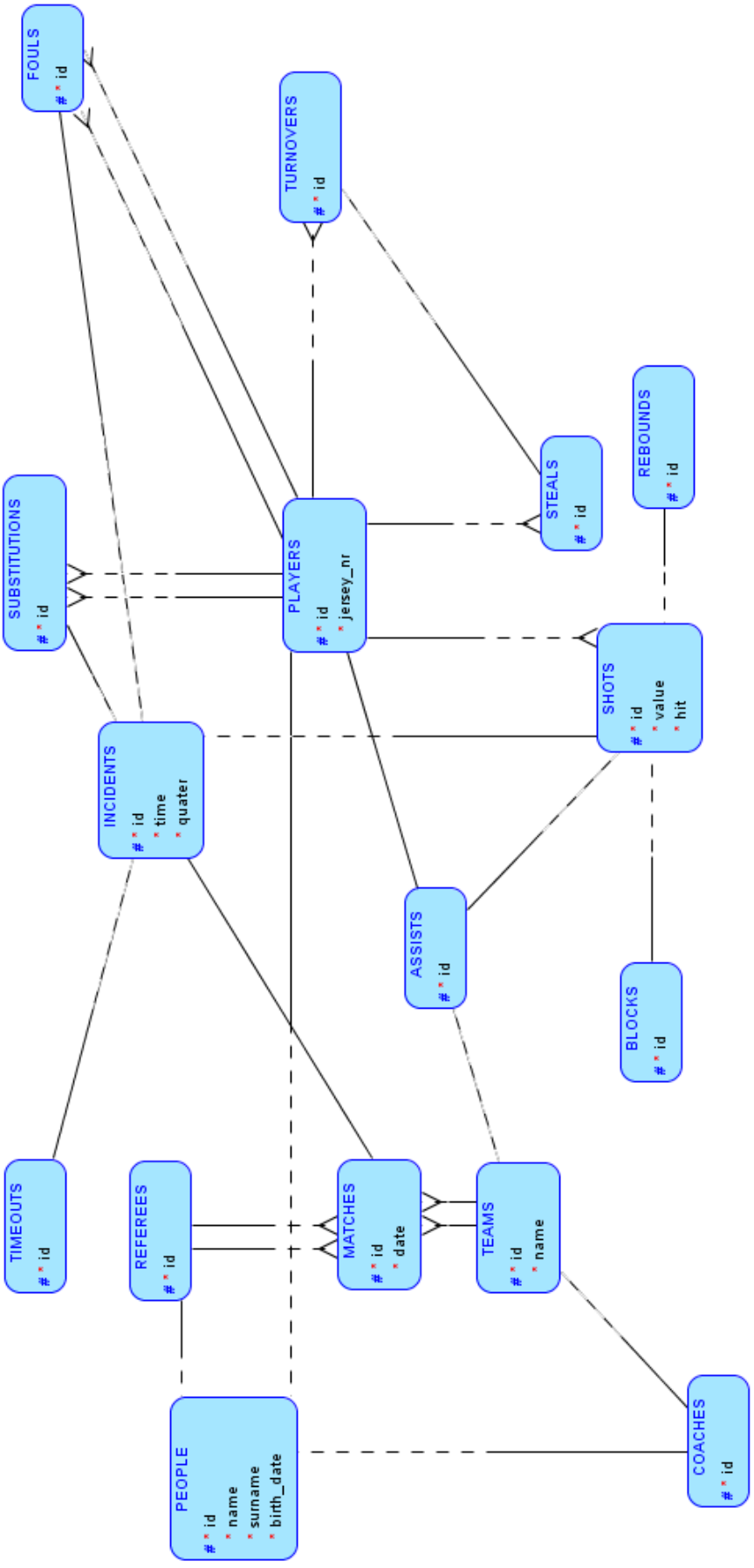
3 Projekt Bazy Danych

3.1 Model logiczny i normalizacja

Wstępne tabele zawarte w punkcie 2.1.1 oraz 2.1.2 poddaliśmy normalizacji której proces wyglądał następująco:

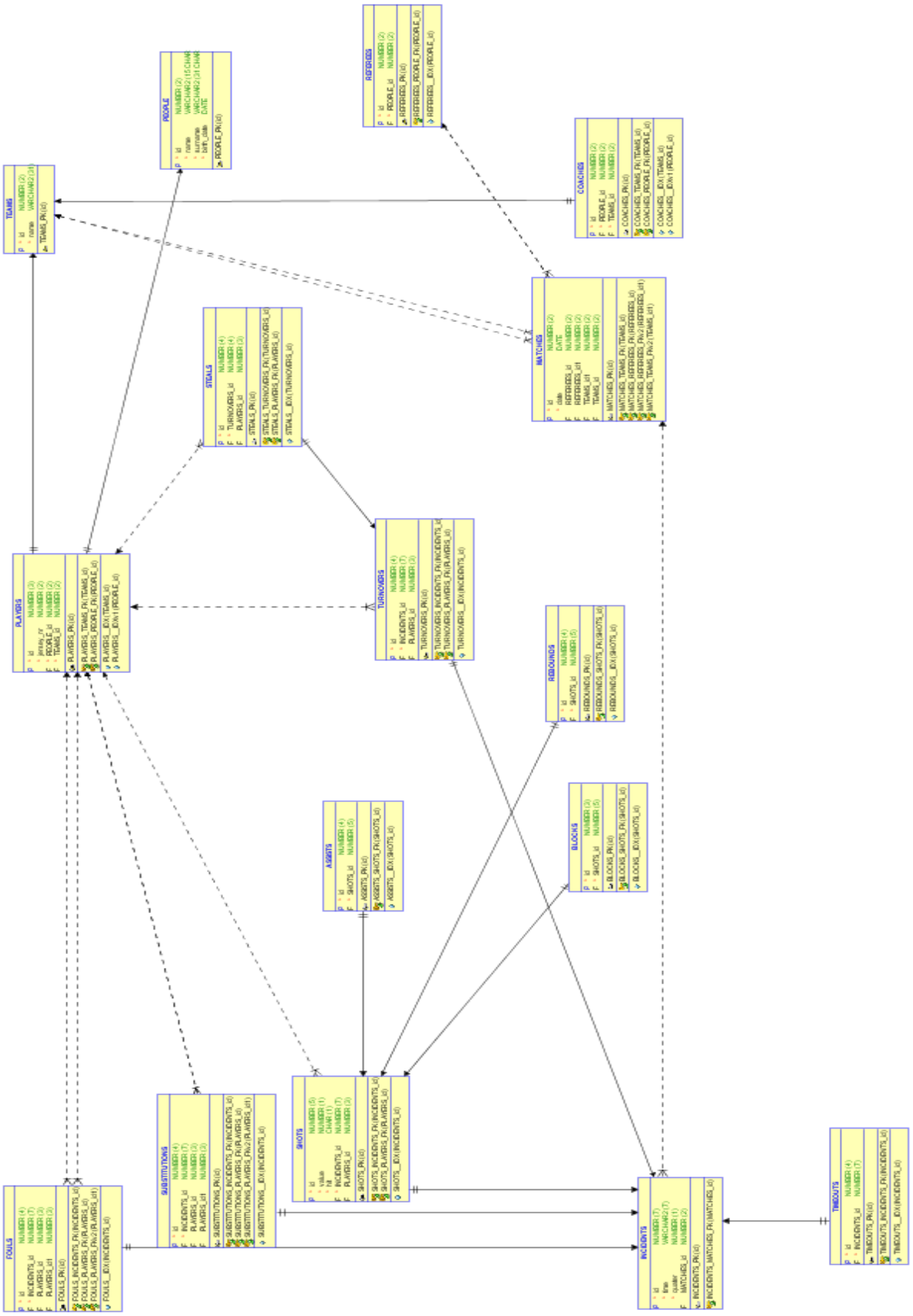
- **Pierwsza postać normalna** - Atomowość uzyskaliśmy na etapie wymyślania tabel, a minimalne alokacje pamięci osiągneliśmy poprzez dopasowanie typów do analizy liczności dla najgorszego przypadku. Dzięki temu tabela nawet w najmniej optymistycznej wersji da radę pomieścić dane, a baza nie będzie zajmowała więcej miejsca niż jest to konieczne.

- **Druga postać normalna** - Przeprowadziliśmy normalizacje aby gracz, trener i sędzia posiadali klucz obcy do tabeli osób i rozszerzali powtarzalne dane takie jak imię nazwisko i wiek o swoje unikalne atrybuty. Myląca nazwa 'id' u trenera i sędziego jest wymagana, ponieważ klucz obcy nie jest losowo generowaną wartością, a numerem licencji dającej uprawnienia do wykonywania zawodu.
- **Trzecia postać normalna** - Nasz schemat bazy nie przewiduje zbędnych kolumn informacyjnych nie odnoszących się do klucza rekordu. Wszystkie kolumny zawierają tylko niezbędne informacje do monitorowania sezonu ligi koszykarskiej i wszystkie atrybuty zależą ściśle od klucza jakim są identyfikatory graczy/drużyn czy też zdarzeń bądź meczy.



Rysunek 1: ERD - logiczny

3.2 Model fizyczny i ograniczenia integralności danych



3.3 Inne elementy schematu – mechanizmy przetwarzania danych

W celu ułatwienia odnajdywania interesujących danych z tabeli przy jednoczesnej minimalizacji pamięci, zdecydowaliśmy się utworzyć widoki.

Pierwszym widokiem, który ułatwi zapytanie o rozkład meczy, będzie po prostu widok realizujący całe zapytanie.
"Rozkład":
(nazwa_drużyny_gospodarzy;nazwa_drużyny_gości;data)

- nazwa_drużyny_gospodarzy - tabela matches, team1_id;
- nazwa_drużyny_gości - tabela matches, team2_id;
- wynik - tabela shots, zliczona ilość punktów na podstawie tabeli shots występujących w meczu(złączone wartości dla obu drużyn)
- data - tabela matches, data i godzina w której odbywa się mecz;

Widokiem który ułatwi wykonywanie statystyki zdarzeń z meczy.
"Lista zdarzeń":
(id_meczu;kwarta;minuta;sekunda;id_zdarzenia)

- id_meczu - tabela matches, id meczu w którym wystąpiło dane przewinienie;
- czas - tabela matches , minuta i kwarta w której następuje przewinienie ;
- nazwa_zdarzenia - tabela incidents, nazwa tabeli która zawiera klucz obcy do danego rekordu w tabeli incydents

Ostatnimi w naszej koncepcji widokami, są tabele zawierające spis wszystkich drużyn oraz zawodników wraz z ich statystykami. Widoki te będą stworzone z użyciem specjalnych zapytań, które zliczą statystyki z innych tabel. Widoki mają zaoszczędzić czas potrzebny w aplikacji na wyciąganie poszczególnych danych. Przewidziane widoki to:
"Statystyki drużyn":

- Nazwa_drużyny - tabela teams, nazwa drużyny;
- Rozegrane mecze - tabela matches, zliczone wystąpienia drużyny w tabeli
- Wygrane mecze - tabela matches,zliczone wystąpienia drużyny w tabeli pod warunkiem że drużyna zdobyła w meczu więcej punktów od przeciwnika(wygrała)
- Punkty Ligowe - tabela matches, zliczone mecze wygrane przemnożone przez ligową wartość zwycięstwa oraz remisy zliczone w ten sam sposób
- Średnia liczba rzutów - tabela shots, zliczone rzuty podzielone przez ilość meczy, na podstawie drużyn zawodników rzucających
- Liczba Rzutów - tabela shots, zliczone rzuty pogrupowane na podstawie drużyn zawodników rzucających
- Skuteczność Rzutów - tabela shots, zliczone wszystkie rzuty pod warunkiem że trafione podzielone przez wszystkie rzuty danej drużyny
- Średnia liczba punktów - tabela shots, zliczone punkty podzielone przez ilość meczy, na podstawie drużyn zawodników rzucających

- Liczba punktów - tabela shots, zliczone punkty pogrupowane na podstawie drużyn zawodników rzucających
- Średnia liczba fauli - tabela fouls, zliczone faule pogrupowane według kluczy drużyn, zawodników popełniających faul
- Średnia wymuszonych fauli -tabela fouls, zliczone faule pogrupowane według kluczy drużyn, zawodników faulowanych
- Średnia strat - tabela turnover, zliczone straty pogrupowane według kluczy drużyn zawodników;

Dalsze punkty ze względu na rozmiar projektu zostaną pominięte w implementacji

- Średnia liczba rzutów wolnych - tabela shots, zliczone rzuty wolne podzielone przez ilość meczy, na podstawie drużyn zawodników rzucających
- Liczba rzutów wolnych - tabela shots, zliczone rzuty wolne pogrupowane na podstawie drużyn zawodników rzucających
- Skuteczność rzutów wolnych - tabela shots, zliczone wszystkie rzuty wolne pod warunkiem że trafione podzielone przez wszystkie rzuty wolne drużyny
- Średnia liczba rzutów za trzy - tabela shots, zliczone rzuty za 3 podzielone przez ilość meczy, na podstawie drużyn zawodników rzucających
- Liczba rzutów za trzy - tabela shots, zliczone rzuty za 3 pogrupowane na podstawie drużyn zawodników rzucających
- Skuteczność rzutów za trzy - tabela shots, zliczone wszystkie rzuty za 3 pod warunkiem że trafione
- Średnia asyst - tabela assists, zliczone asysty pogrupowane według kluczy drużyn zawodników;
- Średnia zbiorek - tabela rebounds, zliczone zbiórki pogrupowane według kluczy drużyn zawodników;
- Średnia bloków - tabela blocks, zliczone bloków pogrupowane według kluczy drużyn zawodników blokujących
- Średnia przechwytów - tabela steals, zliczone przechwyty pogrupowane według kluczy drużyn zawodników;

"Statystyki zawodników":

- Imie - tabela person;
- Nazwisko- tabela person;
- jersey_nr - tabela player;
- Średnia liczba rzutów - tabela shots, zliczone rzuty podzielone przez ilość meczy, na podstawie zawodników rzucających
- Liczba Rzutów - tabela shots, zliczone rzuty pogrupowane na podstawie zawodników rzucających
- Skuteczność Rzutów - tabela shots, zliczone wszystkie rzuty pod warunkiem że trafione podzielone przez wszystkie rzuty zawodnika

- Średnia liczba punktów - tabela shots, zliczone punkty podzielone przez ilość meczy, pogrupowane na podstawie zawodników rzucających
- Liczba punktów - tabela shots, zliczone punkty pogrupowane na podstawie zawodników rzucających
- Średnia liczba fauli - tabela fouls, zliczone faule pogrupowane według kluczy zawodników popełniających faul
- Średnia strat - tabela turnover, zliczone straty pogrupowane według kluczy zawodników;

Podobnie jak poprzednio, dalsze punkty ze względu na rozmiar projektu zostaną pominięte w implementacji

- Średnia liczba rzutów wolnych - tabela shots, zliczone rzuty wolne podzielone przez ilość meczy, na podstawie zawodników rzucających
- Liczba rzutów wolnych - tabela shots, zliczone rzuty wolne pogrupowane na podstawie drużyn zawodników rzucających
- Skuteczność rzutów wolnych - tabela shots, zliczone wszystkie rzuty wolne pod warunkiem że trafione podzielone przez wszystki rzuty wolne zawodnika
- Średnia liczba rzutów za trzy - tabela shots, zliczone rzuty za 3 podzielone przez ilość meczy, na podstawie zawodników rzucających
- Liczba rzutów za trzy - tabela shots, zliczone rzuty za 3 pogrupowane na podstawie zawodników rzucających
- Skuteczność rzutów za trzy - tabela shots, zliczone wszystkie rzuty za 3 pod warunkiem że trafione podzielone przez wszystki rzuty wolne zawodnika
- Średnia asyst - tabela assists, zliczone assysty pogrupowane według kluczy zawodników;
- Średnia zbiorek - tabela rebounds, zliczone zbiórki pogrupowane według kluczy zawodników;
- Średnia bloków - tabela blocks, zliczone bloków pogrupowane według kluczy zawodników blokujących
- Średnia przejęć - tabela steals, zliczone przejęcia pogrupowane według kluczy zawodników;

Rozważaliśmy specjalną tabele statystyk i wyzwalacze aktualizujące dane po dodaniu zdarzenia. Dzięki rozwiązaniu z widokami uniknęliśmy tych mechanizmów i zdecydowaliśmy, że przy obecnym stanie tabel nie są nam potrzebne żadne wyzwalacze.

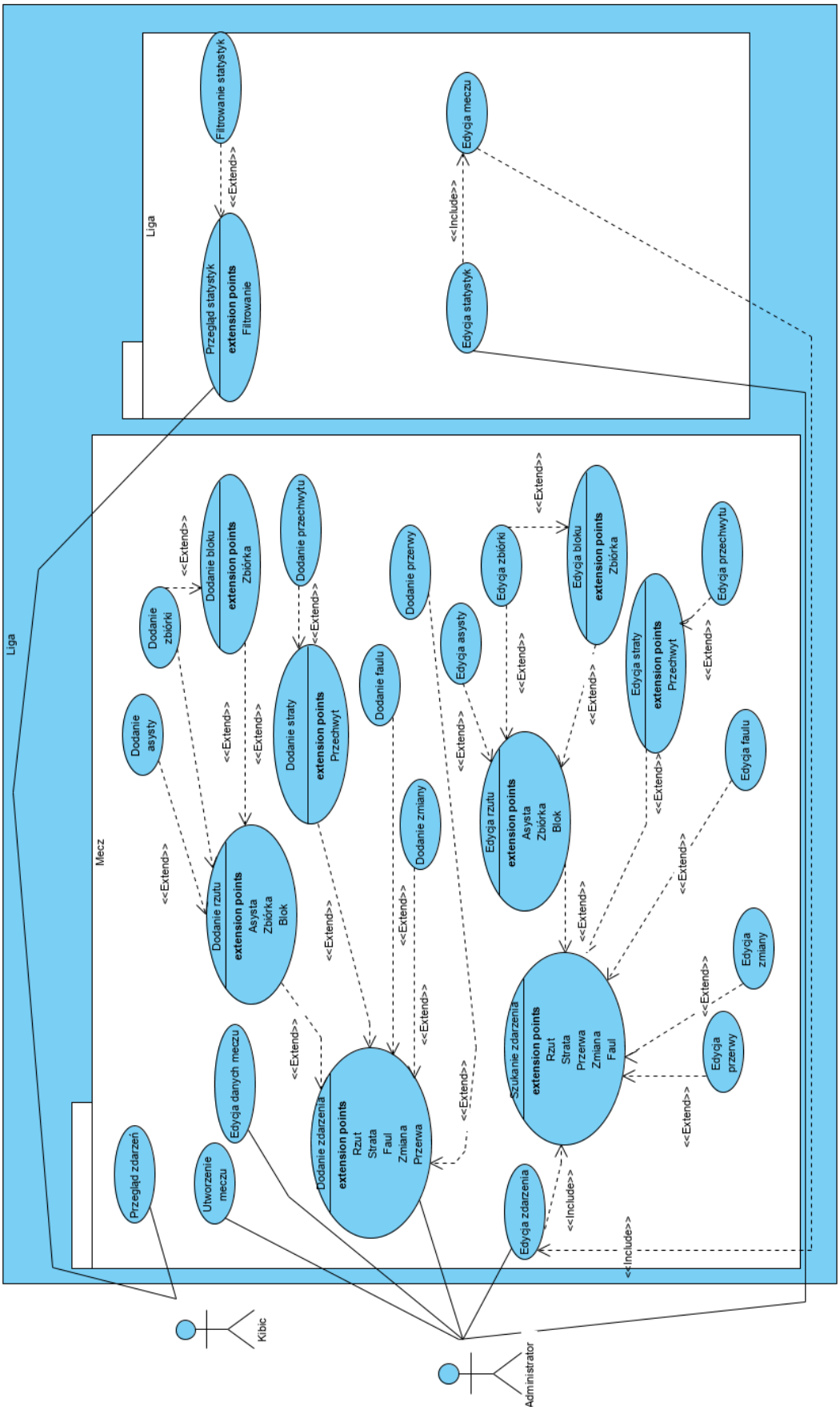
3.4 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Mechanizmy zabezpieczeń obejmują nadanie tabelom i widokom ograniczenia dostępu dla operacji. Przede wszystkim widoki aktualizują się na podstawie rekordów bazy danych, więc wszystkie są obarczone prawem jedynie do odczytu. Cała baza danych jest zabezpieczona hasłem, które ogranicza dostęp do tabel.

4 Projekt aplikacji użytkownika

4.1 Architektura aplikacji i diagramy projektowe

Zakładamy, że aplikacja okienkowa będzie wyposażona w kilka paneli uruchamianych poprzez elementy menu w górnej belce okna. Program musi zezwalać użytkownikowi na wyświetlanie danych i nie dać mu możliwości edytowania ich bez potwierdzenia, że jest on administratorem, dlatego jednym z przycisków będzie logowanie do funkcji moderatora, która udostępni przejście do metod dodawania/edycji/usuwania danych. W pozostałych rozsuwanych menu będą znajdowały się odnośniki do wszystkich przypadków użycia z których może korzystać każdy kibic.



Rysunek 2: Diagram przypadków użycia

4.2 Interfejs graficzny i struktura menu

Interfejs graficzny będzie opisany komponentami pakietu Swing. Zakładamy, że w dużej mierze będą składały się z tabel przewijanych i przycisków. Niektóre bardziej złożone operacje, takie jak dodawanie rekordów będą wymuszały otworzenie specjalnych okien dialogowych z formularzem. Dopiero po wpisaniu wszystkich wymaganych danych, będziemy mogli zatwierdzić formularz, który aplikacja przetłumaczy go na gotowe obiekty i doda je do bazy danych.

4.3 Projekt wybranych funkcji systemu

Wyświetl listę zawodników - Metoda uruchamiana z rozwijanego menu. Metoda wywoła utworzenie zapytania w języku JPQL, i otworzy transakcje z serwerem bazy, aby wykonać komendę. Zakładamy utworzenie specjalnych struktur na które framework 'hibernate' zmapuje rekordy tabeli i ułatwi ich wyświetlanie w przewijanych listach panelu aplikacji. Metoda zostanie zakończona po wyświetleniu na ekranie wszystkich odnalezionych obiektów.

Dodaj rzut - Bardziej skomplikowana metoda, uruchamiana z menu konkretnego meczu, wymagająca otworzenia nowego formularza (warto nadmienić że metoda wymaga potwierdzenia, że użytkownik ma prawa administratora). Po dodaniu niezbędnych informacji do utworzenia rekordu w tabeli **'incidents'**(kwarta i czas) przejdziemy do formularza o danych rzutu. Listy rozwijane i przyciski ułatwią nam zaznaczenie niezbędnych właściwości rekordu. Przy zatwierdzaniu system sprawdzi czy wszystkie wymagane pola zostały uzupełnione danymi oraz czy typ zgadza się z wymaganym formatem. Jeśli wszystkie warunki zostanę spełnione metoda przetłumaczy formularz na obiekty, otworzy transakcje i doda obiekty do bazy danych.

4.4 Metoda podłączania do bazy danych – integracja z bazą danych

Same zapytania będą wykorzystywały JPA czyli specyfikacje ułatwiającą komunikację programu z bazą danych. Rekordy na obiekty (oraz obiekty na rekordy) będą mapowane z wykorzystaniem frameworku hibernate. Potrzebne klasy, na które będziemy mapowali wyniki zapytań zostaną utworzone w fazie implementacji i zostaną zapisane w pamięci programu.

4.5 Projekt zabezpieczeń na poziomie aplikacji

Jak wcześniej wspomnieliśmy aplikacja musi posiadać autoryzację do poziomu administratora, który jest uprawniony do edycji danych. Będzie się to odbywało poprzez specjalne okno dialogowe, które sprawdzi poprawność podawanego hasła. Bez tego punktu nie będziemy w stanie dostać się do żadnego panelu dodawania/usuwania czy edycji rekordów bazy. Ze względu na chwilowy brak komercyjnego przeznaczenia aplikacji zdecydowaliśmy pominąć specjalne tabele na informacje o moderatorach oraz algorytmy szyfrowania haseł. Hasło będzie zapisane w pewnym bloku programu, a aplikacja porówna tylko zadaną wartość ze zmienną systemową Jednak cały blok będzie łatwo rozszerzalnym elementem aplikacji, który będzie można w przyszłości udoskonalić o sprawdzanie poprawności z danymi z zewnętrznych źródeł i algorytmy szyfrowania danych.

5 Implementacja systemu baz danych

5.1 Tworzenie tabel i definiowanie ograniczeń

Skrypt bazy danych został wygenerowany automatycznie, przy pomocy Oracle Data Modeler używając stworzonego przez nas diagramu ERD.Powstałe tabele są zgodne z wcześniej przeprowadzoną przez nas analizą liczność. W tabelach ASSISTS , REBOUNDS, TIMEOUTS i BLOCKS zostało dodane po jednej kolumnie na id gracza (PLAYERS_id).

Przykładowe fragmenty skryptu bazy:

Listing 1: Fragment skryptu tworzącego tabele ASSISTS

```
CREATE TABLE assists (
    id          NUMBER(4) NOT NULL,
    shots_id    NUMBER(5) NOT NULL,
    players_id    NUMBER(3) NOT NULL
);

CREATE UNIQUE INDEX assists__idx ON
    assists (shots_id, players_id
    ASC );

ALTER TABLE assists ADD CONSTRAINT assists_pk PRIMARY KEY ( id );
```

Listing 2: Fragment skryptu tworzącego tabele BLOCKS

```
CREATE TABLE blocks (
    id          NUMBER(3) NOT NULL,
    shots_id    NUMBER(5) NOT NULL,
    players_id    NUMBER(3) NOT NULL
);

CREATE UNIQUE INDEX blocks__idx ON
    blocks (
        shots_id,players_id
    ASC );

ALTER TABLE blocks ADD CONSTRAINT blocks_pk PRIMARY KEY ( id );
```

Listing 3: Fragment skryptu dodającego klucze obce

```
ALTER TABLE assists
    ADD CONSTRAINT assists_shots_fk FOREIGN KEY ( shots_id )
        REFERENCES shots ( id );

ALTER TABLE assists
    ADD CONSTRAINT assists_players_fk FOREIGN KEY ( players_id )
        REFERENCES players ( id );

ALTER TABLE blocks
    ADD CONSTRAINT blocks_shots_fk FOREIGN KEY ( shots_id )
        REFERENCES shots ( id );

ALTER TABLE blocks
    ADD CONSTRAINT blocks_players_fk FOREIGN KEY ( players_id )
        REFERENCES players ( id );
```

Listing 4: Fragment skryptu wypełniającego baze przykładowymi danymi

```
TO_DATE('31-PA -1995','DD-MON-YYYY'), 'Jakub', 'Pietrus');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (1, TO_DATE('26-LUT
    -1995','DD-MON-YYYY'), 'Radek', 'Lis');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (2, TO_DATE('01-STY
    -1988','DD-MON-YYYY'), 'Kamil', 'Picheta');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (3, TO_DATE('05-LIS
    -1994','DD-MON-YYYY'), 'Maciek', 'Bialkowski');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (4, TO_DATE('09-LIS
    -1900','DD-MON-YYYY'), 'Roman', 'Rozanski');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (5, TO_DATE('20-SIE
    -1996','DD-MON-YYYY'), 'Orly', 'Scullion');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (6, TO_DATE('05-LIS
    -1976','DD-MON-YYYY'), 'Abdel', 'Whysall');
insert into PEOPLE ("ID", birth_date, "NAME", SURNAME) values (7, TO_DATE('23-WRZ
    -1983','DD-MON-YYYY'), 'Sybille', 'Grellier');
```

5.2 Implementacja mechanizmów przetwarzania danych

Mechanizmy przetwarzania danych w obrębie bazy opiera się na indeksach i widokach. Większość indeksów została wygenerowana na kluczach obcych obiektów przez program w którym projektowaliśmy diagramy ERD i który przetworzył nasz projekt na bazę danych, jednakże musieliśmy wzbogacić automatycznie wygenerowany kod o dodatkowe indeksy.

Indeksowane kolumny wybraliśmy na podstawie analizy potrzeb naszego programu. Zdecydowaliśmy że istotnymi dla nas informacjami są rzuty, a właściwie ich wartość i logiczna zmienna opisująca trafienie dlatego wzbogaciliśmy indeks tej tabeli o wymienione wartości, wygląda on następująco:

```
CREATE UNIQUE INDEX shots__idx ON
        shots (id,value,incidents_id DESC );
```

To wzbogacenie oraz indeksy na kluczach obcych zamykają pulę indeksów przyspieszających pracę bazy danych na potrzeby użytkowania aplikacji.

Wygenerowane przez nas widoki są zapytaniami generującymi statystyki aby przyspieszyć proces działania aplikacji i wygodę odnoszenia się do danych. Dodatkowo pozwala nam zrezygnować z wyzwalaczy ponieważ wszystko jest zliczane na podstawie rekordów tabeli. Ich wykorzystanie daje nam również dodatkowe zabezpieczenie integralności danych, ponieważ widoki te są nieedytowalne. Wszystkie zapytania generujące widoki zawiera skrypt generujący bazę danych. Przykładowo widok genereujący statystyki gracza wygląda następująco:

Listing 5: Widok PLAYERS_STAT

```
CREATE OR REPLACE FORCE VIEW PLAYERS_STAT
("imie", "nazwisko", "druzyna", "rozegrane_mecze", "liczba_pkt",
"celne_2pt", "oddane_2pt", "skutecznosc_2pt", "celne_3pt", "oddane_3pt",
"skutecznosc_3pt", "celne_1pt", "oddane_1pt", "skutecznosc_1pt") AS
SELECT  e.name as "imie",
        e.surname as "nazwisko",
        t.name as "druzyna",
        NVL((select count(id) from matches where teams_id1 = p.teams_id
        or teams_id = p.teams_id),0) as "rozegrane_mecze",
        NVL((select sum(value) from shots where players_id = p.id AND
        hit='1'),0)  as "liczba_pkt",
        (select count(id) from shots where players_id = p.id
        and value=2 and hit='1') as "celne_2pt",
        (select count(id) from shots where players_id = p.id and value=2)
        as "oddane_2pt",
```

```
16          NVL(ROUND(100*(select count(id) from shots where players_id = p.id
17          and value=2 AND hit='1')/
18               NULLIF((select count(id) from shots where players_id = p.id
19               and value=2), 0),1),0) ||'%' as "skutecznosc_2pt",
20          (select count(id) from shots where players_id = p.id and value=3
21          and hit='1') as "celne_3pt",(select count(id)
22          from shots where players_id = p.id and value=3) as "oddane_3pt",
23          NVL(ROUND(100*(select count(id) from shots where players_id = p.id
24          and value=3 AND hit='1')/
25               NULLIF((select count(id) from shots where players_id = p.id
26               and value=3), 0),1),0) ||'%' as "skutecznosc_3pt",
27
28          (select count(id) from shots where players_id = p.id
29          and value=1 and hit='1') as "celne_1pt",
30          (select count(id) from shots where players_id = p.id
31          and value=1) as "oddane_1pt",
32          NVL(ROUND(100*(select count(id) from shots where
33          players_id = p.id and value=1 AND hit='1')/
34               NULLIF((select count(id) from shots
35               where players_id = p.id and value=1), 0),1),0) ||'%'
36               as "skutecznosc_1pt"
37
38  FROM players p join people e on p.people_id = e.id
39  JOIN teams t on (t.id=p.teams_id)
40  ORDER BY "liczba_pkt" DESC;
```

Sam kod nie mówi wiele jednak schemat utworzonej tabeli i prezentacja na podstawie danych wygląda następująco:

Columns | Data | Grants | Dependencies | Details | Triggers | SQL | Errors

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
1	imie	VARCHAR2(15 CHAR)	No	(null)	1	(null)	YES	YES	YES
2	nazwisko	VARCHAR2(31 CHAR)	No	(null)	2	(null)	YES	YES	YES
3	druzyna	VARCHAR2(31)	No	(null)	3	(null)	NO	NO	NO
4	rozegrane_mecze	NUMBER	Yes	(null)	4	(null)	NO	NO	NO
5	liczba_pkt	NUMBER	Yes	(null)	5	(null)	NO	NO	NO
6	celne_2pt	NUMBER	Yes	(null)	6	(null)	NO	NO	NO
7	oddane_2pt	NUMBER	Yes	(null)	7	(null)	NO	NO	NO
8	skutecznosc_2pt	VARCHAR2(41)	Yes	(null)	8	(null)	NO	NO	NO
9	celne_3pt	NUMBER	Yes	(null)	9	(null)	NO	NO	NO
10	oddane_3pt	NUMBER	Yes	(null)	10	(null)	NO	NO	NO
11	skutecznosc_3pt	VARCHAR2(41)	Yes	(null)	11	(null)	NO	NO	NO
12	celne_1pt	NUMBER	Yes	(null)	12	(null)	NO	NO	NO
13	oddane_1pt	NUMBER	Yes	(null)	13	(null)	NO	NO	NO
14	skutecznosc_1pt	VARCHAR2(41)	Yes	(null)	14	(null)	NO	NO	NO

	imie	nazwisko	druzyna	rozegrane_mecze	liczba_pkt	celne_2pt	oddane_2pt	skutecznosc_2pt	celne_3pt	oddane_3pt	skutecznosc_3pt	celne_1pt	oddane_1pt	skutecznosc_1pt
1	Wileen	Dumingo	Flashdog	2	34	17	30	56,7%	0	0	0%	0	0	0%
2	Gherardo	Juhruke	Meeveo	2	28	14	30	46,7%	0	0	0%	0	0	0%
3	Brandtr	Erricker	Meeveo	2	21	0	0	0%	7	20	35%	0	0	0%
4	Norrie	Heinrich	Flashdog	2	15	0	0	0%	5	20	25%	0	0	0%
5	Vere	MacNeish	Meeveo	2	13	0	0	0%	0	0	0%	13	15	86,7%
6	Peterus	Titherington	Flashdog	2	12	0	0	0%	0	0	0%	12	15	80%
7	Jakub	Pietrus	Flashdog	2	0	0	0	0%	0	0	0%	0	0	0%
8	Radek	Lis	Flashdog	2	0	0	0	0%	0	0	0%	0	0	0%
9	Kamil	Picheta	Voonyx	2	0	0	0	0%	0	0	0%	0	0	0%
10	Maciek	Bialkowski	Jamia	2	0	0	0	0%	0	0	0%	0	0	0%
11	Roman	Rozanski	Edgetag	2	0	0	0	0%	0	0	0%	0	0	0%
12	Orly	Scullion	Demimbu	2	0	0	0	0%	0	0	0%	0	0	0%
13	Abdel	Whysall	Meeveo	2	0	0	0	0%	0	0	0%	0	0	0%
14	Sybille	Grellier	Flashdog	2	0	0	0	0%	0	0	0%	0	0	0%
15	Leanora	Espinoy	Voonyx	2	0	0	0	0%	0	0	0%	0	0	0%
16	Theodora	Eastway	Flashdog	2	0	0	0	0%	0	0	0%	0	0	0%
17	Doralynne	Pattingson	Shuffledrive	2	0	0	0	0%	0	0	0%	0	0	0%
18	Otha	Barnwill	Browsedrive	2	0	0	0	0%	0	0	0%	0	0	0%
19	Tybalt	Giacopetti	Brainbox	2	0	0	0	0%	0	0	0%	0	0	0%
20	Travers	Kunneke	Shufflebeat	2	0	0	0	0%	0	0	0%	0	0	0%
21	Jillayne	Pawelski	Flashdog	2	0	0	0	0%	0	0	0%	0	0	0%
22	Carrie	Blanchet	Shuffledrive	2	0	0	0	0%	0	0	0%	0	0	0%
23	Buck	Gors	Voonyx	2	0	0	0	0%	0	0	0%	0	0	0%
24	Gertrudis	Vasilischev	Brainbox	2	0	0	0	0%	0	0	0%	0	0	0%

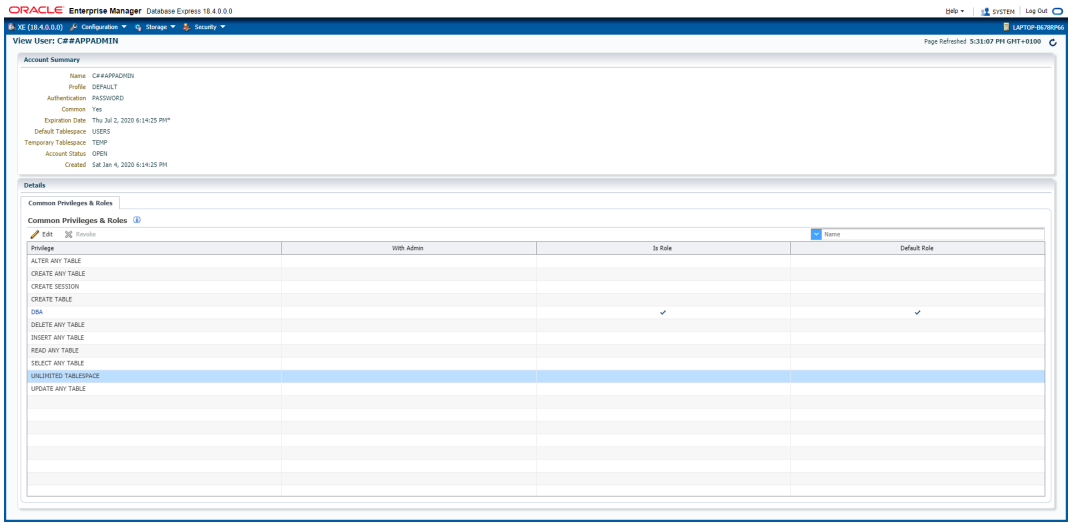
Rysunek 3: Wyniki działania zapytania tworzącego widok PLAYERS_STAT

Baza danych zawiera zadeklarowane we wcześniejszych etapach projektu widoki, przy czym statystyki gracza zostały rozdzielone na dwie tabele ze względu na rozmiar aplikacji i potrzebę okrojenia danych w aplikacji dostępowej, która jest kolejnym etapem projektu.

5.3 Implementacja uprawnień i innych zabezpieczeń

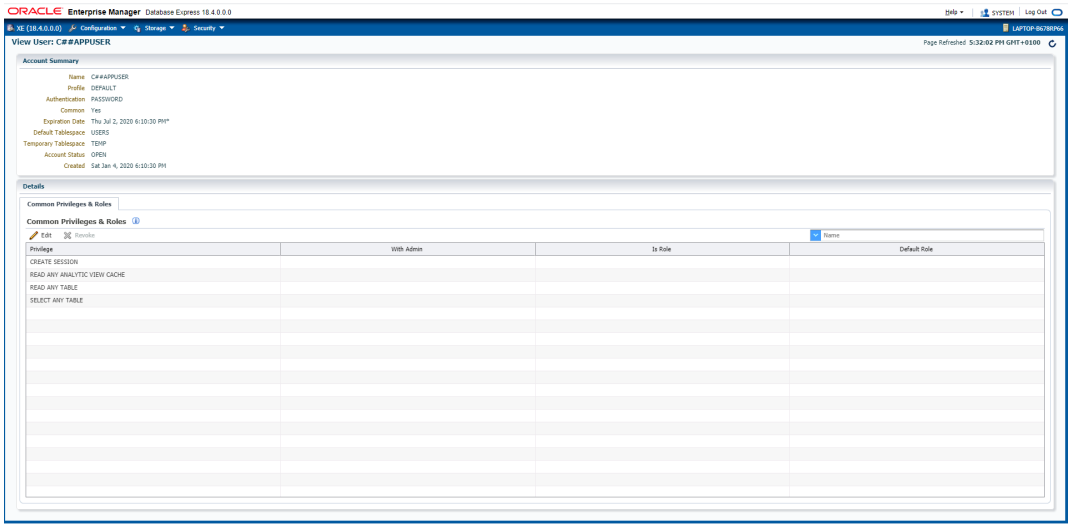
Zabezpieczenia osiągnęliśmy poprzez utworzenie dwóch oddzielnych użytkowników. Poza domyślnym logowaniem do systemu zaimplementowaliśmy dwóch użytkowników c##appAdmin oraz c##appUser .Zostali oni utworzeni poprzez aplikacje webową 'Enterprise Menager', dostarczaną przez firmę Oracle wraz z instalacją bazy danych. Ze względu na lepszą przejrzystość GUI nie zamieszczamy kodu SQL służącego do konfigurowania ograniczeń użytkowników a tylko przedstawimy ich zastosowanie i prezentacje w aplikacji.

- Administrator / c##appadmin - posiada uprawnienia do wszelkiego rodzaju edycji danych w tabelach jak również prawa do tworzenia tabel i widoków. To on implementuje skrypt bazodanowy. Zalogowanie do tego trybu jest niezbędne, aby dodać dodawać informacje o meczach lub edytować już istniejące.



Rysunek 4: Dane użytkownika 'c##appadmin' wraz z przywilejami

- Użytkownik/Kibic / c##appuser - posiada uprawnienia do odczytów danych z wszystkich tabel i widoków. Ze względu na przeznaczenie bazy danych nie musimy wybiórczo nadawać przywileju do konkretnych tabel, ponieważ i tak nie jest on wstanie naruszyć integralności danych ze względu na ograniczenie 'read-only', a z poziomu aplikacji dostępowej będzie jedynie możliwe wyświetlenie odpowiednich statystyk.



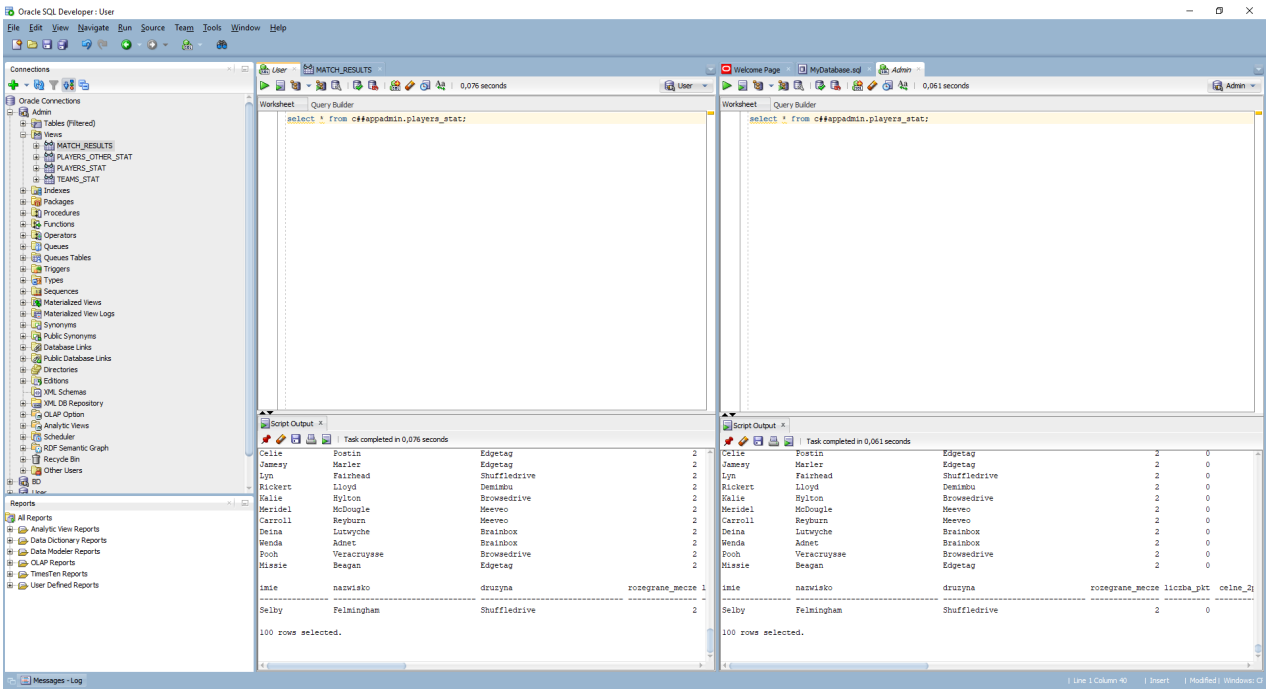
Rysunek 5: Dane użytkownika ’c##appUser’ wraz z przywilejami

Implementowany przez nas podział zwiększa bezpieczeństwo związane z zachowaniem integralności. Nasza baza powinna zezwalać nieautoryzowanym użytkownikom wyłącznie na odczyt danych i taki efekt osiągamy. Dostęp do użytkownika ’kibic’ jest szyfrowany domyślnym hasłem, które będzie zapisane w pliku konfiguracyjny aplikacji dostępowej co utrudnia jakikolwiek dostęp do bazy z aplikacji nie utworzonej przez nas.

Poprawne logowanie pozwoli nam na korzystanie z zapytań jako użytkownik ’administrator’ , a aplikacja wyświetli dodatkowe przyciski pozwalające na edycje danych. Aby upewnić się, że wszystko działa zodnie założeń, przeprowadziłem testy dla całego CRUD-u, które pokazują że osiągnęliśmy porządny widok, kolejno zapytania po lewej strony są wykonywane z poziomu ’c##appUser’ a po prawo z poziomu ’c##appAdmin’:

Rysunek 6: Zestawienia zapytań z poziomu kibica i administratora dla tabel

Poza zapytaniami dla tabel należy jeszcze sprawdzić poprawność wyciągania danych z widoków:

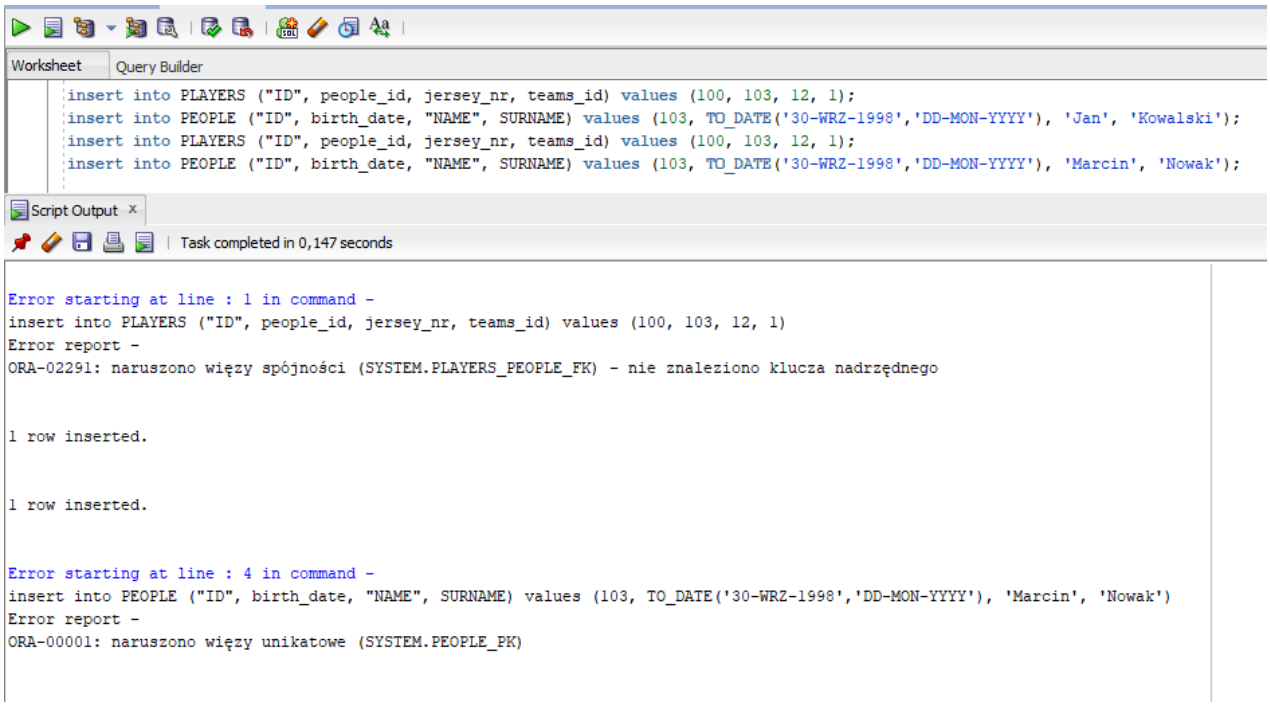


Rysunek 7: Zestawienia zapytań z poziomu kibica i administratora dla widoków

Widzimy że przeprowadzone testy potwierdzają założenia tworzenia podanych użytkowników i wprowadzają pożądane zabezpieczenia.

5.4 Testowanie bazy danych na przykładowych danych

Testowanie systemu, na podstawie różnych komend i zapytań pozwoliło na określenie zbudowanej bazy jako zupełnie poprawnej. Na poniższym screenie zostało przedstawione dodawanie gracza o nieistniejącym id osoby, następnie dodanie osoby o tym id i ponowne dodanie zawodnika. Za pierwszym razem nie powiodło się z oczywistych przyczyn, lecz za drugim razem wszystko przebiegło w pełni poprawnie. Następnie zaszła próba dodania osoby o już istniejącym id, która oczywiście się nie powiodła



Rysunek 8: Przykładowe komendy

6 Implementacja i testy aplikacji

Z powodu braku wystarczającej ilości czasu zdecydowaliśmy się na uproszczoną wersję aplikacji. Nasza aplikacja będzie w stanie zaprezentować możliwości połączenia z bazą danych, pobrania z niej informacji oraz modyfikacji danych znajdujących się w niej. Do prezentacji wyników posłuży nam kosola stworzona w polu tekstowym aplikacji SWING. Z uwagi na działalność aplikacji pomijamy punkt prezentacji interfejsu użytkownikam zastępujemy go za to interfejsem programistycznym.

6.1 Zmapowane obiekty reprezentujące rekordy

6.1.1 Interfejs programistyczny

Zgodnie z konwencją programowania aplikacji w sposób trójwarstwowy, warstwa dostępu do danych została odpowiednio wydzielona tworząc interfejs programistyczny, dostępowy do bazy danych. Zawiera on metody neizbędne do pokazania użyteczności aplikacji a wygląda on następująco:

Listing 6: interfejs programistyczny zawierający metody operacji na na zewnętrzej BD

```
1  //interfejs programistyczny zawieraj ce metody operacji na na zewn Ztrzej
       bazie danych
2  public interface IDataAccess {
3
4      //metody logowania do odpowiedniego panelu
5      public boolean loginAsUser();
6      public boolean loginAsAdmin();
7
8      //select from tables
9      public List<PeopleEntity> getAllPeople();
10     public List<PlayersEntity> getAllPlayers();
11     public TeamsEntity getTeamById(int id);
```

```
12
13      //select from views
14      public List<PlayersStatEntity> getPlayersStats();
15      public List<PlayersOtherStatEntity> getPlayersOtherStats();
16      public List<MatchResultsEntity> getMatchResults();
17
18      //crud funkcjonalno Żci, metody sprawdzaj ce operacje na poszczeg lnych
        tabelach domy Żlnie tworzonymi obiektami
19
20      //crud #1 - sprawdzone dzia Ćanie - kontroler - metoda shortTestPerson
21      public PeopleEntity createSimplePerson();
22      public void addSimplePerson();
23      public PeopleEntity getSimplePerson();
24      public void updateSimplePerson();
25      public void removeSimplePerson();
26
27      //crud #2 - sprawdzone dzia Ćanie - kontroler - metoda shortTestPlayer
28      public PlayersEntity createSimplePlayer();
29      public void addSimplePlayer();
30      public PlayersEntity getSimplePlayer();
31      public void updateSimplePlayer();
32      public void removeSimplePlayer();
33
34      //pomocniczy getter osoby
35      public PeopleEntity getPersonById(Byte id);
36
37      // #3 - przetestowane dzia Ćanie - dodawanie/usuwanie zdrzenia z przypisanym
         rzutem
38      public void addThrow();
39      public void removeThrow();
40
41      public MatchResultsEntity getMatchById(long id);
42  }
```

Metody te pozwalają zaprezentować poprawne odwoływanie się do bazy danych.

6.2 Instalacja i konfigurowanie systemu

Aplikacja wykorzystuje Java 9. Funkcje powinny odpowiednio działać do każdej wyższej wersji javy, jednak nie zaleca się stosowania poprzednich wersji ze względu na możliwe błędy składniowe. Nasza aplikacja łączy się z bazą danych za pomocą frameworku Hibrnate. Konfiguracja połączenia znajduje się w pliku "presistance.xml". Konfigurujemy w nim 2 rodzaje połączenia - jako użytkownik lub jako administrator.

Wdrażanie aplikacji wymaga posiadania na urządzeniu Java SE Development Kit 9 oraz IDE (zintegrowanego środowiska programistycznego) pozwalającego na tworzenie i modyfikacje programów w języku Java z narzędziem Maven (preferowany Eclipse).

Wymagane jest posiadanie również bazy danych zawierającej te same tabele i widoki. Z racji braku hostowania globalnie bazy danych, wymaga to utworzenia bazy Oracle Express Edition lokalnie na swoim urządzeniu i zbudowanie tabel i widoków załączonym przez nas skryptem.

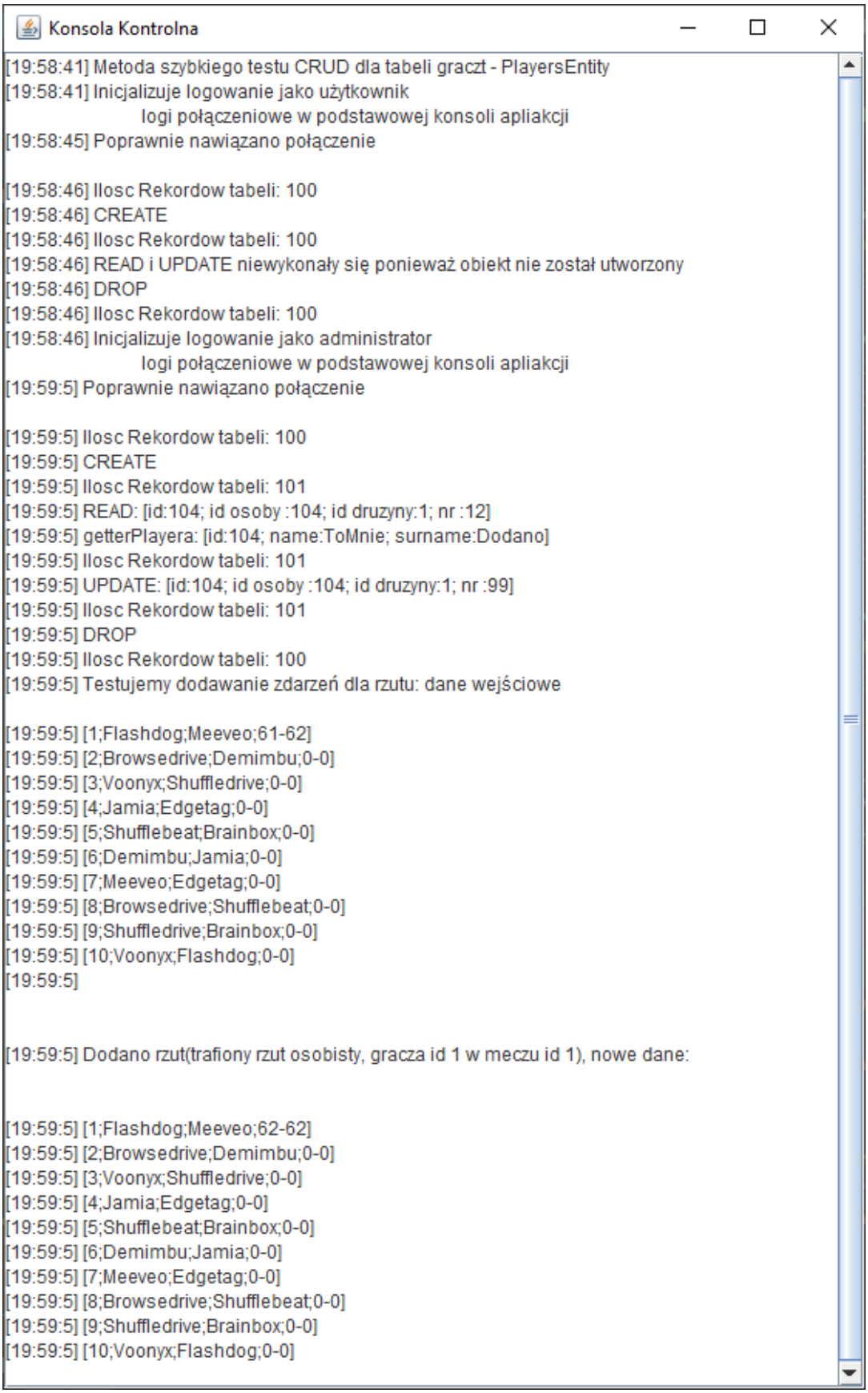
Następnie należy utworzyć użytkowników, nadając im określone login, hasło oraz wymagane uprawnienia. Loginy i hasła do tych użytkowników należy skonfigurować w pliku "persistance.xml" naszej aplikacji, przejrzysty plik nie pozostawia wątpliwości gdzie powinny znaleźć się określone parametry. Po zapisaniu plików możemy przejść do ostatniego kroku, którym jest zbudowanie aplikacji i pozwolenie framework'owi na połączenie z bazą danych.

6.3 Instrukcja użytkowania aplikacji

Ze względu na ograniczenie rozbudowy aplikacji, może być ona użytkowana za pomocą hardkodowanych, indywidualnych - dla każdej tabeli - testów CRUD, w których można sprawdzić poprawność działania bazy jak i samej aplikacji. W tym celu należy uruchomić IDE i wybrać odpowiedni scenariusz.

6.4 Testowanie opracowanych funkcji systemu

Dla tabeli osób w aplikacji został zaimplementowany test CRUD. Jak widać na poniższym obarazku najpierw logujemy się jako użytkownik. Z tego powodu nie powiodły się próby CREATE, UPDATE oraz DROP. READ nie wykonało się, ponieważ obiekt nie został utworzony. Następnie logujemy się jako administrator i wszytkie testy CRUD przechodzą pomyślnie. Ostatnim etapem testu jest test dodawania zdarzeń dla rzutu. Wyświetlony jest najpierw pobrany z bazy widok MATCH_RESULTS, następnie dodany zostaje rzut i finalnie znowu pobrany zostaje widok. Jak widać w meczu o id 1 wynik został pomyślnie zmieniony.



Rysunek 9: Wynik testu

6.5 Implementacja interfejsu dostępu do bazy danych

Interfejs jest implementowany przez specjalną klase implementacyjną nazwaną zgodnie z javową konwencją <nazwainterfejsu>-impl(od słowa implementation), która określa sposó odwołań do bazy danych. Wykorzystuje on obiekt zwany 'Menadżerem encji' który na podstawie zmapowanych struktór wie do których tabel się odwoływać na podstawie typu parametrów parametrów. Za komunikacje odpowiadaja trzy parametry:

Listing 7: parametry służące do komunikacji

```
1     private PersistenceProvider persistenceProvider ;
2     private EntityManagerFactory entityManagerFactory;
3     private EntityManager entityManager ;
```

Tworzenie konstruktor "fabryki menadżerów encji" jest parametryzowany stringiem określającym dane użytkownika. Odwołuje sie on do nazwy zawartej w pliku persistance.xml dlatego poprawna konfiguracja użytkownika jest niezbędna do działania aplikacji. Konstruktory tych parametrów są określone w metodach logowania któr esa prezentowane w kolejnym punkcji.
Interfejs implementuje podstawowe metody pozwalające odwoływac się do bazy danych aby wyciągać, umieszczać, edytować czy też usuwac dane z tabel. Przykłady funkcji:

Listing 8: metoda addSimplePlayer

```
1     @Override
2     public void addSimplePlayer() {
3         PlayersEntity player = createSimplePlayer();
4         try {
5             entityManager.getTransaction().begin();
6             entityManager.persist(player);
7             entityManager.getTransaction().commit();
8         }catch(Exception e){
9             System.out.println("[Wyjatek remove simple player]\n"+e.getMessage());
10        }
11    }
```

Listing 9: metoda removeSimplePlayer

```
1        @Override
2     public void removeSimplePlayer() {
3         PlayersEntity player = createSimplePlayer();
4         try {
5             entityManager.getTransaction().begin();
6             player = entityManager.find(PlayersEntity.class, player.getId());
7             entityManager.remove(player);
8             entityManager.getTransaction().commit();
9         }catch(Exception e){
10            System.out.println("[Wyjatek remove simple player]\n"+e.getMessage());
11        }
12    }
```

Listing 10: metoda updateSimplePlayer

```
1     @Override
2     public void updateSimplePlayer() {
3         PlayersEntity player = createSimplePlayer();
4         try {
5             entityManager.getTransaction().begin();
6             player = entityManager.find(PlayersEntity.class, player.getId());
7             player.setJerseyNr(99);
8             entityManager.getTransaction().commit();
9         }catch(Exception e){
10            System.out.println("[Wyjatek update simple player]\n"+e.getMessage());
11        }
12    }
```

6.6 Implementacja wybranych funkcjonalności systemu

Jedną z podstawowych funkcjonalności jest logowanie użytkowników do odpowiedniego panelu. Zależnie od tego czy jest zwykłym użytkownikiem czy administratorem jest wykonywane odpowiednie połączenie oraz uruchomiony zostaje menadżer encji:

Listing 11: metoda loginAsUser

```
@Override
    public boolean loginAsUser() {
        try{
            this.persistenceProvider =  new HibernatePersistenceProvider();
            this.entityManagerFactory  = persistenceProvider.createEntityManagerFactory("UserPersistenceUnit",hashMap);
            this.entityManager = entityManagerFactory.createEntityManager();
        }catch(Exception e){
            return false;
        }
        return true;
    }
```

Metoda 'loginAsAdmin' zawiera analogiczne ciało, z tą różmnicą że łączy sie jako administrator do bazy co zezwala z pozycji programu na dodawanie, edytowanie i usuwanie rekordów. Będąc zalogowany jako użytkownik możemy tylko odczytywac dane z bazy.

Drugą z ważnych funkcjonalności jest przedstawienie zaimplementowanych w bazie widoków za pomocą specjalnych metod, które wyświetlają wszystkie dane, atrybuty należące do danego widoku:

Listing 12: metoda getMatchResults

```
@Override
    public List<MatchResultsEntity> getMatchResults() {
        final String SELECT_QUERY ="from MatchResultsEntity ";
        return entityManager.createQuery(SELECT_QUERY, MatchResultsEntity.class).getResultList();
    }
```

6.7 Implementacja mechanizmów bezpieczeństwa

Mechanizm bezpieczeństwa został zrealizowany jako logowanie osobno do panelu administratora i osobno do panelu użytkownika za pomocą odpowiednich danych logowania. Pomiędzy dwoma kontami istnieje rozróżnienie ze względu na funkcjonalności z których może korzystać administrator lub użytkownik. Administrator posiada pełne prawo do kontrolowania i edycji bazy danych za pomocą komend CREATE, UPDATE oraz DROP, a użytkownik może jedynie odczytać dane lub wybrane informacje o nich z tabeli.

Efekt działania zaimplementowanego mechanizmu bezpieczeństwa został ukazany na rysunku 9.

7 Podsumowanie projektu

Podsumowując realizację projektu, jesteśmy usatysfakcjonowani mechanizmami zaimplementowanymi zarówno w bazie danych jak i samej aplikacji. Udało nam się rozwiązywać napotkane problemy, o istnieniu których nawet nie wiedzieliśmy.

Realizując tak duży projekt warto zacząć od nabycia istotnej i niezbędnej wiedzy teorytycznej o wykorzystywanych mechanizmach, ponieważ nasz projekt zdefiniował wykorzystanie technologii w których nikt z nas dotąd nie pracował, co powodowało problemy w implementacjach na kolejnych etapach. Jednakże wstępny przegląd narzędzi, pozwolił nam zadeklarować, które będą dobrze współpracowały ze sobą i finalnie aplikacja pokazuje poprawne nawiązanie połączeń i wymianę danych .

Niestety czas poświęcony zwłaszcza na konfigurację mapowania w podejściu database-first, spowodował naszą decyzje o rezygnacji z interfejsu użytkownika, który potrzebował dużej ilości kodu obejmującego warstwy logiki biznesowej i prezentacji. Na szczęście nie był sednem projektu i pozwolono nam z niego zrezygnować. Opracowując nasz projekt nabyliśmy wiedzę o modelowaniu warstwy danych aplikacji.

Dowiedzieliśmy sie jak długi i złożony jest proces poprzedzający implementację oraz jak wielkie znaczenie ma dla projektu. Etapy modelowania założeń i diagramów zajęły troche czasu, jednak rezultaty pozwoliły nam go nadrobić w etapach implementacyjnych, w których encje zostały generowane z diagramu fizycznego , a klasy na podstawie encji. Jedyne elementy projektu, których nie zdążyliśmy rozszerzyć to aplikacja, której funkcjonalność została zaprezentowana, jednak do funkcji pozwalającej obsługiwać ligę pozostaje jeszcze wiele godzin i setek linii kodu. Niestety na tak wczesnym etapie jakim było definiowanie tematu projektu i ilości tabel nie zdawaliśmy sobie sprawy z potencjalnego rozmiaru aplikacji i w ostatecznym etapie skala nas delikatnie przerosła.