

POLITECHNIKA KRAKOWSKA im. T. KOŚCIUSZKI

WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ

Projekt z przedmiotu:

**Systemy Baz Danych**

**Ciepiela Adrian**

**Góra Patryk**

**Iwanowicz Michał**

**Semestr 5**

**Rok akademicki 2016/2017**

Grupa 31i

**Projekt nadzorował:**

Mgr Krzysztof Czajkowski

Kraków, styczeń 2017

1. **Założenia projektowe**

Głównym celem projektu jest utworzenie bazy danych opartej o technologię relacyjnych baz danych firmy Oracle. Język użyty do implementacji bazy danych to OracleSQL oraz PL/SQL. Utwożona baza danych będzie wystarczająca aby umożliwić skuteczną pracę aplikacji do obsługi sieci kin, rozwinąć ją o dodatkowe funkcjonalności, dane. Baza danych zawierać będzie informacje potrzebne do pracy kasjera jak również menadżera kina. Produkt będzie także służył jako archiwum danych, które może być wykorzystane do rozszerzenia możliwości bazy.

Do implementacji bazy danych dodawany jest opis, który pomaga używać bazę, zrozumieć jej strukturę, a także umożliwić jej dalsze rozwijanie przez programistów.

Dokumentacja przystosowana jest do użytku zarówno przez programistów jak   
i użytkowników.

Baza przechowywać będzie informacje na temat:

* oddziałów kina,
* zatrudnionych pracowników,
* klientów,
* poszczególnych sal w kinie,
* obecnie granych filmów,
* archiwum granych filmów,
* rodzajów biletów,
* sprzedanych biletów.

Dzięki bazie aplikacja będzie w stanie oferować następnujące funkcje:

Funkcje obsługiwane przez kasjera:

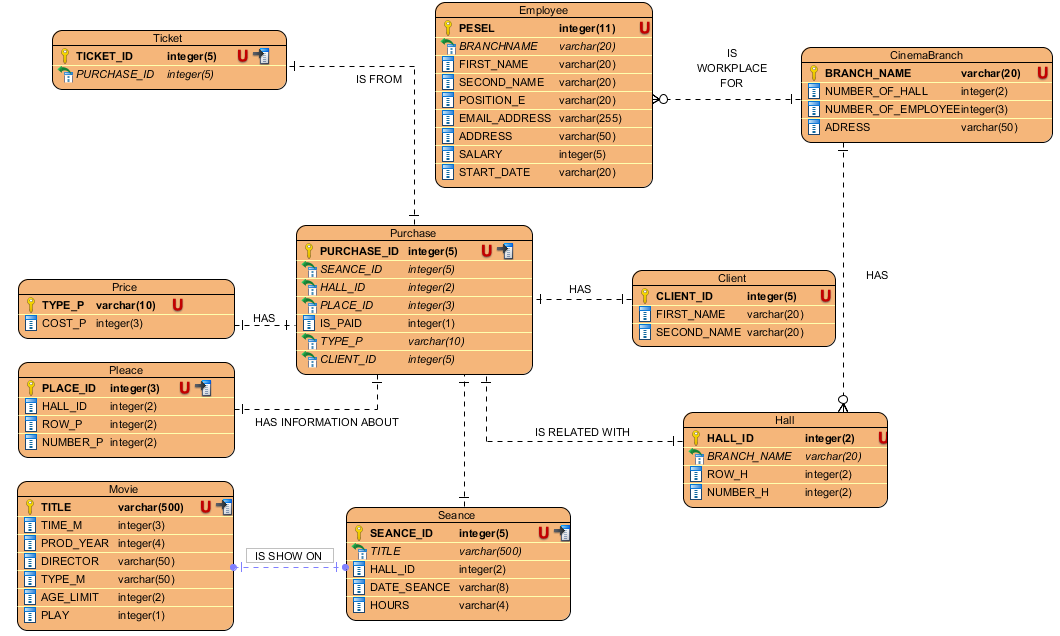
* wydawanie biletów,
* zwrot biletów,
* wymiana biletu,
* rezerwacja miejsc,
* wyświetlanie informacji o cenach i zniżkach,
* wyświetlanie repertuaru.

Funkcje obsługiwane przez menadżera:

* zatrudnianie pracowników,
* zwalnianie pracowników,
* dodawanie filmów do bazy,
* wprowadzanie filmów do repertuaru,
* usuwanie filmów z repertuaru,
* ustalanie w jakich salach będzie wyświetlany dany film,
* modyfikacja cen biletów,
* wyświetlanie informacji o pracownikach.

W naszej bazie nie przewidujemy zakładania dodatkowych indeksów, ponieważ operacje obciążone kosztownymi wyszukiwaniami będą wykonywane na kluczach głównych relacji, które w Oracle SQL zapewniają nam automatyczne założenie indeksu unikalnego. Klucze główne wraz z indeksami będą utworzone w tabelach, które przedstawiają naturalne byty w świecie rzeczywistym. Dzięki temu system bazodanowy będzie mógł dokonywać wyboru, czy skanować wartości tabel, czy szukać „po indeksach”, w zależności od szacunkowych kosztów, które są oparte o wcześniejsze wykonywanie podobnych lub takich samych zapytań. Szacunki te nie zawsze są dobre i czasami powodują błędny wybór, ale na wysokim etapie prac z bazą danych system dobrze dobiera obliczenia szacunkowe i wybiera co raz to lepsze opcje wyszukiwania.

1. Diagram ERD



1. Opis relacji, diagram

**Ticket:**

Relacja ta jest odzwierciedleniem biletu w kinie.

*TICKET\_ID* – samogenerujący się, unikalny numer biletu,klucz główny, założony jest na nim indeks

*PURCHASE\_ID* – klucz obcy, unikalny numer zakupu, realizuje powiązanie pomiędzy biletem i zakupem

**Employee:**

Relacja ta jest odzwierciedleniem pracownika kina.

*PESEL* – pesel pracownika, unikalny klucz główny

*BRANCHNAME* – klucz obcy, nazwa oddziału kina, w którym pracuje dany pracownik

*FIRST\_NAME* – imię pracownika

*SECOND\_NAME* – nazwisko pracownika

*POSITION\_E* – stanowisko na którym zatrudniony jest pracownik

*EMAIL\_ADDRESS* – adres email pracownika

*ADDRESS* – adres zamieszkania pracownika

*SALARY* – wynagrodzenie pracownika

*STARD\_DATE* – data zatrudnienia pracownika

**Client:**

Relacja reprezentuje klienta kina.

*CLIENT\_ID* – klucz główny, unikalny numer klienta

*FIRST\_NAME* – imię klienta

*SECOND\_NAME* – nazwisko klienta

**Purchase:**

Relacja reprezentuje zarezerwowany lub kupiony bilet.

*PURCHASE\_ID* – klucz główny, unikalny numer zakupu  
*SEANCE\_ID* – klucz obcy, numer seansu  
*HALL\_ID* – klucz obcy, numer sali  
*PLACE\_ID* – klucz obcy, numer miejsca w Sali

*IS\_PAID* – pole określające czy została już uiszczona opłata za bilet  
*TYPE\_P* – klucz obcy, rodzaj ceny (np. ulgowy, normalny)  
*CLIENT\_ID* – klucz obcy, numer klienta

**Price:**

Relacja opisuje rodzaje ceny (np. ulgowy, normalny).

*TYPE\_P* – klucz główny, rodzaj ceny

*COST\_P* – koszt danego rodzaju biletu

**CinemaBranch:**

Tabela ta opisuje oddział kina.

*BRANCH\_NAME* – nazwa oddziału kina, jest unikalnym kluczem głównym

*NUMBER\_OF\_HALL* – ilość sal w oddziale

*NUMBER\_OD\_EMPLOYEE* –ilość pracowników pracujących w danym oddziale

*ADRESS* – dokładny adres oddziału

**Hall:**

Zawiera informacje o poszczególnych salach w kinie.

*HALL\_ID* – unikalny numer sali w całej sieci kin, jest kluczem głównym tabeli

*BRANCH\_NAME* – klucz obcy pokazujący, w którym oddziale banku znajduje się sala

*ROW\_H* – ilość rzędów w sali

*NUMBER\_H* – ilość miejsc w rzędzie

**Seance:**

Relacja zawierające informacje o seansach.

*SEANCE\_ID* – numer seansu, unikalny klucz główny, na którym założony jest indeks

*TITLE* – tytuł filmu wyświetlanego podczas seansu

*HALL\_ID* –numer sali przypisany do seansu

*DATE\_SEANCE* – data seansu

*HOURS* – godzina seansu

**Movie:**

Relacja reprezentująca film, dane o nim

*TITLE* – tytuł filmu, który jest jego kluczem głównym i założony jest na niego indeks

*TIME\_M* – czas trwania filmu

*PROD\_YEAR* – rok produkcji filmu

*DIRECTOR* – reżyser

*TYPE\_M* – gatunek filmowy

*AGE\_LIMIT* – ograniczenie wiekowe

*PLAY* - informacja o tym, czy film jest aktualnie grany czy nie (1 – grany, 0 - nie)

**Place:**

Tabela odpowiadająca za opis każdego miejsca w kinie

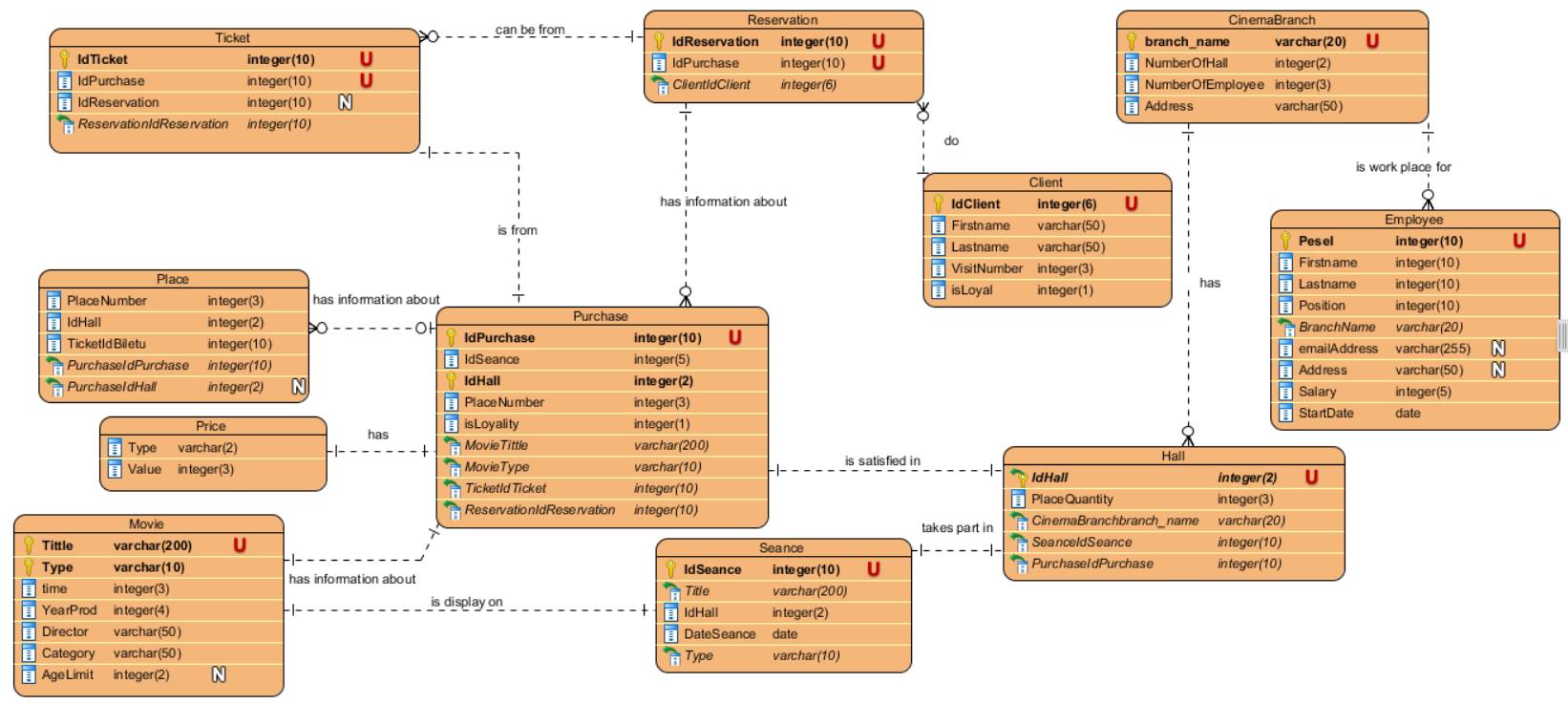
*PLACE\_ID* – numer krzesła/miejsca w kinie, unikalny klucz główny

*HALL\_ID* – numer sali, w której jest krzesło

*ROW\_P* – rząd, w którym jest krzesło

*NUMBER\_P* – miejsce w rzędzie, w którym jest krzesło

1. Wcześniejsze wersje projektu



Schemat przedstawia wcześniejszą wersję bazy, która zawierała kilka nieprzemyślanych do końca rozwiązań:

* informacje się zapętlały,
* relacja RESERVATION była niepotrzebna,
* w relacji PURCHASE i TICKET były nie potrzebne dane.

Schemat ten był za mało zoptymalizowany, przez co w procedurach pojawiały się zapętlenia dostępu do danych.

Na tym etapie chcieliśmy tworzyć funkcje naprzemiennie z procedurami, jednak stwierdziliśmy, że dla naszego problemu procedury są w pełni wystarczające.

W obecnym projekcie procedury tworzone są łatwiej, bez niepotrzebnych zawiłości, przez co są wykonywane szybciej i bardziej intuicyjne.

Na tym etapie dane były zadeklarowane jako elementarne, tak aby spełniona była pierwsza postać normalna.

Tablea PURCHASE nie była zgodna z 2 postacią normalną ponieważ dane np IdHall, MovieTittle, MovieType można było przewidzieć na podstawie idSeance.

Przez to, że druga postać nie była spełniona, trzecia także nie mogła być spełniona.

Na obecnym etapie projektu trzecia postać jest spełniona co implikuje poprawność pierwszej i drugiej.

1. Kod SQL tworzący struktury bazodanowe

Opis poszczególnych relacji oraz ich atrybutów znajduje się powyżej w punkcie 3.

CREATE TABLE CINEMABRANCH (

BRANCH\_NAME VARCHAR(20) CONSTRAINT PK\_BRANCH\_NAME PRIMARY KEY NOT NULL,

NUMBER\_OF\_HALL NUMBER(2) NOT NULL,

NUMBER\_OF\_EMPLOYEE NUMBER(3) NOT NULL,

ADDRESS VARCHAR(50) NOT NULL

);

CREATE TABLE CLIENT (

CLIENT\_ID NUMBER(5) CONSTRAINT PK\_CLIENT\_ID PRIMARY KEY NOT NULL,

FIRST\_NAME VARCHAR(20) NOT NULL,

SECOND\_NAME VARCHAR(20) NOT NULL,

IS\_LOYAL NUMBER(1) NOT NULL

);

CREATE TABLE EMPLOYEE (

PESEL NUMBER(11) CONSTRAINT PK\_PESEL PRIMARY KEY NOT NULL,

BRANCH\_NAME VARCHAR(20),

FIRST\_NAME VARCHAR(20) NOT NULL,

SECOND\_NAME VARCHAR(20) NOT NULL,

POSITION\_E VARCHAR2(20) NOT NULL,

EMAIL\_ADDRESS VARCHAR2(255) NOT NULL,

ADDRESS VARCHAR2(50) NOT NULL,

SALARY NUMBER(5) NOT NULL,

START\_DATE VARCHAR2(8) NOT NULL

);

CREATE TABLE HALL (

HALL\_ID NUMBER(2) CONSTRAINT PK\_HALL\_ID PRIMARY KEY NOT NULL,

BRANCH\_NAME VARCHAR2(20) NOT NULL,

ROW\_H NUMBER(2) NOT NULL,

NUMBER\_H NUMBER(2) NOT NULL

);

CREATE TABLE MOVIE (

TITLE VARCHAR2(500) CONSTRAINT PK\_TITLE PRIMARY KEY NOT NULL,

TIME\_M NUMBER(3) NOT NULL,

PROD\_YEAR NUMBER(4) NOT NULL,

DIRECTOR VARCHAR2(50) NOT NULL,

TYPE\_M VARCHAR2(50) NOT NULL,

AGE\_LIMIT NUMBER(2) NOT NULL,

PLAY NUMBER(1) NOT NULL

);

CREATE TABLE PLEACE (

PLACE\_ID NUMBER(3) CONSTRAINT PK\_PLACE\_ID PRIMARY KEY NOT NULL,

HALL\_ID NUMBER(2) NOT NULL,

ROW\_P NUMBER(2) NOT NULL,

NUMBER\_P NUMBER(2) NOT NULL

);

CREATE TABLE PRICE (

TYPE\_P VARCHAR2(10) CONSTRAINT PK\_TYPE\_P PRIMARY KEY NOT NULL,

COST\_P NUMBER(3) NOT NULL

);

CREATE TABLE PURCHASE (

PURCHASE\_ID NUMBER(5) CONSTRAINT PK\_PURCHASE\_ID PRIMARY KEY,

SEANCE\_ID NUMBER(5) NOT NULL,

HALL\_ID NUMBER(2) NOT NULL,

PLACE\_ID NUMBER(3) NOT NULL,

IS\_PAID NUMBER(1) NOT NULL,

TYPE\_P VARCHAR2(10) NOT NULL

);

CREATE TABLE SEANCE (

SEANCE\_ID NUMBER(5) CONSTRAINT PK\_SEANCE\_ID PRIMARY KEY NOT NULL,

TITLE VARCHAR2(500) NOT NULL,

HALL\_ID NUMBER(2) NOT NULL,

DATE\_SEANCE VARCHAR2(8) NOT NULL,

HOURS VARCHAR2(4) NOT NULL

);

CREATE TABLE TICKET (

TICKET\_ID NUMBER(5) CONSTRAINT PK\_TICKET\_ID PRIMARY KEY NOT NULL,

PURCHASE\_ID NUMBER(5) NOT NULL

);

1. Kod PL/SQL tworzący procedury
2. Procedura która dodaje oddział kina. Przyjmuje następujące argumenty:

* *BRANCH\_NAMEA – nazwa oddziału*
* *NUMBEROFHALLA – ilość sal*
* *NUMBEROFEMPLOYEEA – ilość pracowników*
* *ADDRESSA – adres oddziału*

*create or replace procedure addCinemaBranch*

*(BRANCH\_NAMEA VARCHAR2, NUMBEROFHALLA NUMBER, NUMBEROFEMPLOYEEA NUMBER, ADDRESSA VARCHAR2)*

*IS*

*BEGIN*

*INSERT INTO CINEMABRANCH(BRANCH\_NAME,NUMBER\_OF\_HALL,NUMBER\_OF\_EMPLOYEE,ADDRESS) VALUES (BRANCH\_NAMEA,NUMBEROFHALLA,NUMBEROFEMPLOYEEA,ADDRESSA);*

*END;*

1. Procedura która dodaje klienta kina. Przyjmuje następujące argumenty:

* *FIRST\_NAMEA – imię klienta*
* *SECOND\_NAMEA – nazwisko klienta*

*create or replace procedure addClient*

*(FIRST\_NAMEA VARCHAR2,SECOND\_NAMEA VARCHAR2)*

*IS*

*max\_id number(5);*

*BEGIN*

*SELECT max(CLIENT\_ID) INTO max\_id from CLIENT;*

*IF max\_id is NULL then max\_id := 1;*

*ELSE max\_id := max\_id + 1;*

*END IF;*

*INSERT INTO CLIENT(CLIENT\_ID,FIRST\_NAME,SECOND\_NAME) VALUES (max\_id,FIRST\_NAMEA,SECOND\_NAMEA);*

*END;*

1. Procedura która dodaje pracownika kina. Przyjmuje następujące argumenty:

* *PESELA – pesel pracownika*
* *BRANCH\_NAMEA – oddział kina*
* *FIRST\_NAMEA – imię pracownika*
* *SECOND\_NAMEA – nazwisko pracownika*
* *POSITION\_EA – pozycja pracownika*
* *EMAIL\_ADDRESSA – email pracownika*
* *ADDRESSA – adres pracownika*
* *SALARYA – pensja pracownika*
* *START\_DATEA – data zatrudnienia pracownika*

*create or replace procedure addEmployee*

*(PESELA NUMBER,BRANCH\_NAMEA VARCHAR2,FIRST\_NAMEA VARCHAR2, SECOND\_NAMEA VARCHAR2, POSITION\_EA VARCHAR2, EMAIL\_ADDRESSA VARCHAR2, ADDRESSA VARCHAR2, SALARYA NUMBER, START\_DATEA VARCHAR2)*

*IS*

*BEGIN*

*INSERT INTO EMPLOYEE(PESEL, BRANCH\_NAME, FIRST\_NAME, SECOND\_NAME, POSITION\_E, EMAIL\_ADDRESS, ADDRESS, SALARY, START\_DATE) VALUES (PESELA,BRANCH\_NAMEA,FIRST\_NAMEA,SECOND\_NAMEA,POSITION\_EA,EMAIL\_ADDRESSA,ADDRESSA, SALARYA, START\_DATEA);*

*END;*

1. Procedura która dodaje salę kinową. Przyjmuje następujące argumenty:

* *BRANCH\_NAMEA – oddział kina*
* *ROW\_HA – ilość rzędów w sali*
* *NUMBER\_HA – ilość miejsc w rzędzie*

*create or replace procedure addHALL*

*(BRANCH\_NAMEA VARCHAR2, ROW\_HA NUMBER, NUMBER\_HA NUMBER)*

*IS*

*max\_id number(5);*

*BEGIN*

*SELECT max(HALL\_ID) INTO max\_id from HALL;*

*IF max\_id is NULL then max\_id := 1;*

*ELSE max\_id := max\_id + 1;*

*END IF;*

*INSERT INTO HALL(HALL\_ID,BRANCH\_NAME,ROW\_H,NUMBER\_H) VALUES (max\_id,BRANCH\_NAMEA,ROW\_HA,NUMBER\_HA);*

*END;*

1. Procedura która dodaje klienta kina. Przyjmuje następujące argumenty:

* *TITLEA – tytuł filmu*
* *TIME\_MA – czas trwania filmu*
* *PROD\_YEARA – rok produkcji filmu*
* *DIRECTORA – nazwisko reżysera*
* *TYPE\_MA – gatunek filmu*
* *AGE\_LIMITA – ograniczenie wiekowe*

*create or replace procedure addMovie*

*(TITLEA VARCHAR2, TIME\_MA NUMBER,PROD\_YEARA NUMBER,DIRECTORA VARCHAR2,TYPE\_MA VARCHAR2, AGE\_LIMITA NUMBER)*

*IS*

*BEGIN*

*INSERT INTO MOVIE(TITLE, TIME\_M,PROD\_YEAR,DIRECTOR,TYPE\_M,AGE\_LIMIT,PLAY) VALUES (TITLEA,TIME\_MA,PROD\_YEARA,DIRECTORA,TYPE\_MA,AGE\_LIMITA, 1);*

*END;*

1. Procedura która dodaje miejsce (fotel). Przyjmuje następujące argumenty:

* *HALL\_IDA – id sali kinowej*
* *ROW\_PA – numer rzędu w którym stoi fotel*
* *NUMBER\_PA - numer miejsca w rzędzie*

*create or replace procedure addPlace*

*(HALL\_IDA NUMBER, ROW\_PA NUMBER, NUMBER\_PA NUMBER)*

*IS*

*max\_id number(5);*

*BEGIN*

*SELECT max(PLACE\_ID) INTO max\_id from PLEACE;*

*IF max\_id is NULL then max\_id := 1;*

*ELSE max\_id := max\_id + 1;*

*END IF;*

*INSERT INTO PLEACE(PLACE\_ID,HALL\_ID,ROW\_P,NUMBER\_P) VALUES (max\_id,HALL\_IDA,ROW\_PA,NUMBER\_PA);*

*END;*

1. Procedura która dodaje rodzaj ceny. Przyjmuje następujące argumenty:

* *TYPE\_PA – typ ceny (ulgowy, normalny)*
* *COST\_PA – koszt*

*create or replace procedure addPrice*

*(TYPE\_PA VARCHAR2,COST\_PA NUMBER)*

*IS*

*BEGIN*

*INSERT INTO PRICE(TYPE\_P,COST\_P) VALUES (TYPE\_PA,COST\_PA);*

*END;*

1. Procedura która dodaje seans w kinie. Przyjmuje następujące argumenty:

* *TITLEA – nazwa filmu*
* *HALL\_IDA – numer sali*
* *DATE\_SEANCEA – data seansu*
* *HOURSA – godzina senasu*

*create or replace procedure addSeance*

*(TITLEA VARCHAR2, HALL\_IDA NUMBER,DATE\_SEANCEA VARCHAR2, HOURSA VARCHAR2)*

*IS*

*max\_id number(5);*

*BEGIN*

*SELECT max(SEANCE\_ID) INTO max\_id from SEANCE;*

*IF max\_id is NULL then max\_id := 1; ELSE max\_id := max\_id + 1;*

*END IF;*

*INSERT INTO SEANCE(SEANCE\_ID,TITLE,HALL\_ID,DATE\_SEANCE,HOURS) VALUES (max\_id,TITLEA,HALL\_IDA,DATE\_SEANCEA,HOURSA);*

*END;*

1. Procedura która dodaje informację o zarezerwowaniu lub zakupieniu biletu. Przyjmuje następujące argumenty:

* *filmTitle – tytuł filmu*
* *dateFilm – data filmu*
* *hoursA – godzina*
* *rowNumber – numer rzędu*
* *placeNumber – numer fotela w rzędzie*
* *is\_paidA – czy bilet został opłacony*
* *client\_ida – numer klienta*

*create or replace procedure addPurchase*

*(filmTitle varchar2, dateFilm varchar2, hoursA varchar2, rowNumber number, placeNumber number, is\_paidA number, type\_pa varchar2, client\_ida number)*

*is*

*seanceID number(5);*

*hallID number(2);*

*placeID number(3);*

*max\_id number(5);*

*begin*

*select SEANCE\_ID, HALL\_ID into seanceID, hallID from SEANCE where dateFilm = DATE\_SEANCE and hoursA = HOURS and filmTitle = TITLE;*

*select PLACE\_ID into placeID from PLEACE where hallID = HALL\_ID and ROW\_P = rowNumber and NUMBER\_P = placeNumber;*

*SELECT max(PURCHASE\_ID) INTO max\_id from purchase;*

*IF max\_id is NULL then max\_id := 1;*

*ELSE max\_id := max\_id + 1;*

*END IF;*

*insert into PURCHASE(PURCHASE\_ID, SEANCE\_ID, HALL\_ID, PLACE\_ID, IS\_PAID, TYPE\_P, CLIENT\_ID) values(max\_id, seanceID, hallID, placeID, is\_paidA, type\_pa, client\_ida);*

*end;*

1. Procedura która dodaje tworzy bilet. Przyjmuje następujące argumenty:

* *PURCHASE\_IDA – numer rezerwacji lub zakupu*

*create or replace procedure addTicket*

*(PURCHASE\_IDA NUMBER)*

*IS*

*max\_id number(5);*

*BEGIN*

*SELECT max(TICKET\_ID) INTO max\_id from TICKET;*

*IF max\_id is NULL then max\_id := 1;*

*ELSE max\_id := max\_id + 1;*

*END IF;*

*INSERT INTO TICKET(TICKET\_ID,PURCHASE\_ID) VALUES (max\_id,PURCHASE\_IDA);*

*END;*

1. Procedura która powoduje, że dany film nie jest już grany. Przyjmuje następujące argumenty:

* *TITLEA – tytuł filmu*

*create or replace procedure deactivateMovie*

*(TITLEA VARCHAR2)*

*is*

*begin*

*UPDATE MOVIE SET PLAY = 0 where TITLE=TITLEA;*

*END;*

1. Procedura która pokazuje daty wyświetleń filmu. Przyjmuje następujące argumenty:

* *TITLEA – tytuł filmu*

*create or replace procedure SHOWDATEMOVIE*

*(TITLEA IN VARCHAR2)*

*IS*

*BEGIN*

*FOR t IN (SELECT DATE\_SEANCE FROM SEANCE where TITLE=TITLEA)*

*LOOP*

*dbms\_output.put\_line(t.DATE\_SEANCE);*

*END LOOP;*

*END;*

1. Procedura która pokazuje godziny wyświtleń filmu w danym dniu. Przyjmuje następujące argumenty:

* *TITLEA – tytuł filmu*
* *DATA\_SEANCEA – data seansu*

*create or replace procedure SHOWHOURSMOVIE*

*(TITLEA IN VARCHAR2, DATA\_SEANCEA IN VARCHAR2)*

*IS*

*BEGIN*

*FOR t IN (SELECT HOURS FROM SEANCE where TITLE=TITLEA AND DATE\_SEANCE=DATA\_SEANCEA)*

*LOOP*

*dbms\_output.put\_line(t.HOURS);*

*END LOOP;*

*END;*

1. Procedura która pokazuje wyswietlane aktualnie filmy. Przyjmuje następujące argumenty:

*create or replace procedure SHOWMOVIE*

*IS*

*BEGIN*

*FOR t IN (SELECT title FROM movie where play=1)*

*LOOP*

*dbms\_output.put\_line(t.title);*

*END LOOP;*

*END;*

1. Procedura która wyświetla kosz danego typu biletu . Przyjmuje następujące argumenty:

* *TYPA – typ ceny*

*create or replace procedure SHOWPRICE(TYPA IN VARCHAR2)*

*IS*

*COSTA NUMBER(2);*

*BEGIN*

*SELECT COST\_P into COSTA from PRICE where TYPE\_P=TYPA;*

*DBMS\_OUTPUT.PUT\_LINE(COSTA);*

*END;*

1. Procedura która wyświetla zarezerwowane miejsca w danym seansie. Przyjmuje następujące argumenty:

* titleA *– tytuł filmu*
* *dateA – data seansu*
* *hoursA – godzina seansu*

*create or replace procedure showReservedPlaces*

*(titleA varchar2, dateA varchar2, hoursA varchar2)*

*is*

*seanceID number(5);*

*rowPa number(5);*

*placePa number(5);*

*begin*

*select seance\_id into seanceID from seance where title = titleA and date\_seance = dateA and hours = hoursA;*

*for t IN (select place\_id from purchase join SEANCE on PURCHASE.SEANCE\_ID = SEANCE.SEANCE\_ID) LOOP*

*select row\_p, number\_p into rowPa, placePa from PLEACE where PLACE\_ID = t.place\_id;*

*dbms\_output.put\_line(rowPA || ' ' || placePa);*

*end loop;*

*end;*

1. Procedura która pozawala zmienic cenę biletu. Przyjmuje następujące argumenty:

* *TYPA – typ ceny*
* *COSTA - nowa wartość*

*create or replace procedure UPDATEPRICE*

*(TYPA IN VARCHAR2, COSTA IN NUMBER)*

*IS*

*BEGIN*

*UPDATE PRICE SET COST\_P=COSTA where TYPE\_P=TYPA;*

*END;*

1. Procedura która pokazuje email pracownika. Przyjmuje następujące argumenty:

* *FIRST\_NAMEA – imię pracownika*
* *SECOND\_NAMEA – nazwisko pracownika*

*create or replace procedure SHOW\_EMAIL*

*(FIRST\_NAMEA IN VARCHAR2,SECOND\_NAMEA IN VARCHAR2)*

*IS*

*EMAIL\_ADDRESSA VARCHAR2(255);*

*BEGIN*

*SELECT EMAIL\_ADDRESS into EMAIL\_ADDRESSA from EMPLOYEE where FIRST\_NAME=FIRST\_NAMEA AND SECOND\_NAME=SECOND\_NAMEA;*

*DBMS\_OUTPUT.PUT\_LINE(EMAIL\_ADDRESSA);*

*END;*

1. Procedura która pokazuje dane pracownika. Przyjmuje następujące argumenty:

* *SECOND\_NAMEA – nazwisko pracownika*

*create or replace procedure SHOW\_EMPLOYEE*

*(SECOND\_NAMEA IN VARCHAR2)*

*IS*

*BRANCH\_NAMEA VARCHAR2(20);*

*FIRST\_NAMEA VARCHAR2(20);*

*POSITION\_EA VARCHAR2(20);*

*EMAIL\_ADDRESSA VARCHAR2(255);*

*ADDRESSA VARCHAR2(50);*

*SALARYA NUMBER(5);*

*START\_DATEA VARCHAR2(8);*

*BEGIN*

*SELECT BRANCH\_NAME, FIRST\_NAME, POSITION\_E, EMAIL\_ADDRESS, ADDRESS, SALARY, START\_DATE into BRANCH\_NAMEA, FIRST\_NAMEA, POSITION\_EA, EMAIL\_ADDRESSA, ADDRESSA, SALARYA, START\_DATEA from EMPLOYEE where SECOND\_NAME=SECOND\_NAMEA;*

*DBMS\_OUTPUT.PUT\_LINE(BRANCH\_NAMEA||','||FIRST\_NAMEA||','||SECOND\_NAMEA||','||POSITION\_EA||','||EMAIL\_ADDRESSA||','||ADDRESSA||','||SALARYA||','||START\_DATEA);*

*END;*

1. Procedura która pokazuje filmy wyświetlane w danej sali w danym dniu. Przyjmuje następujące argumenty:

* *HALL\_IDA – numer sali*
* *DATEA – data*

*create or replace procedure SHOW\_MOVIE\_IN\_HALL*

*(HALL\_IDA IN VARCHAR2, DATEA IN VARCHAR2)*

*IS*

*BEGIN*

*for t IN (SELECT TITLE from SEANCE where HALL\_ID=HALL\_IDA AND DATE\_SEANCE=DATEA) LOOP*

*dbms\_output.put\_line(t.TITLE);*

*end loop;*

*END;*

1. Przykładowe zapytania
2. Zapytanie wyświetlające imie i nazwisko klientów, którzy opłacili już zamówienie i poszli na film o danym tytule.

select client.FIRST\_NAME, client.SECOND\_NAME from client

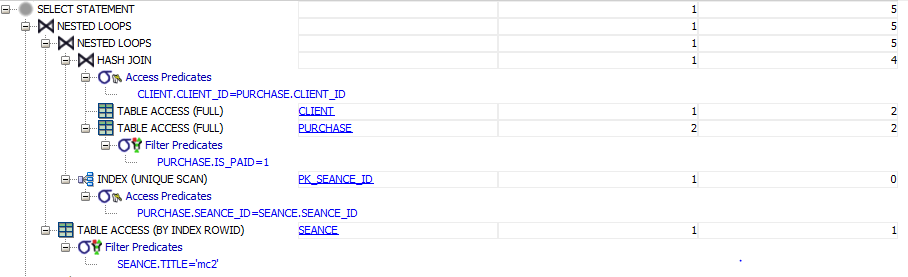
join purchase on client.CLIENT\_ID = purchase.client\_id

join seance on purchase.seance\_id = seance.seance\_id

join movie on seance.title = movie.title

where purchase.IS\_PAID = 1

and movie.title = 'mc2';



1. Zapytanie, któro wyświetla nam wszystkie tytuły filmów oraz reżysera tego filmu, na którym był klient o danym imieniu i nazwisku.

Ograniczeniem jest to, że zamówienie (Purchase) musi być wystawione dla klienta.

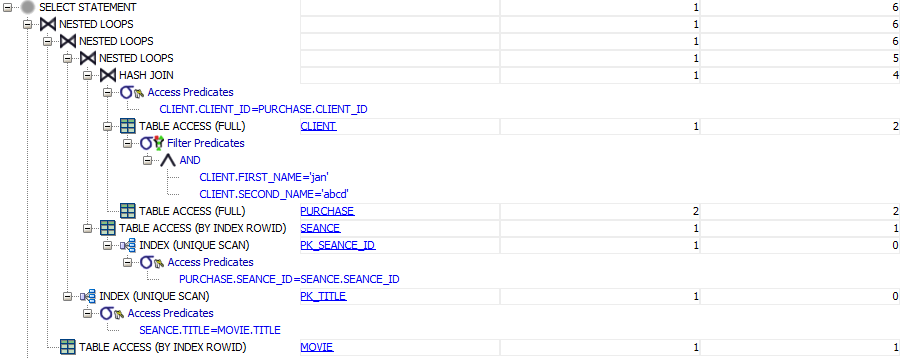
select movie.title, DIRECTOR from movie

left join seance on seance.TITLE=movie.TITLE

left join purchase on purchase.SEANCE\_ID=seance.SEANCE\_ID

left join client on client.CLIENT\_ID = purchase.CLIENT\_ID

where client.FIRST\_NAME = 'jan' and client.SECOND\_NAME='abcd';



1. Zapytanie, któro wyświetla nazwe oddziału kina, gdzie w danym dniu i o danej godzinie jest wyświetlany dany film.

select branch\_name from CinemaBranch

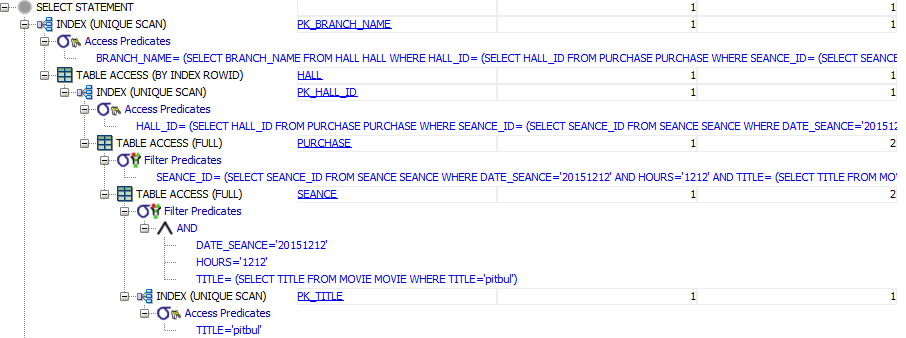
where branch\_name = (select branch\_name from hall

where hall\_id = (select hall\_id from purchase

where seance\_id = (select seance\_id from seance

where date\_seance = '20151212' and hours > '1200' and title = (select title from movie

where title = 'pitbul'))));



1. Zapytanie wyświetlające ilość osób,, które kupiły bilet na film danego reżysera

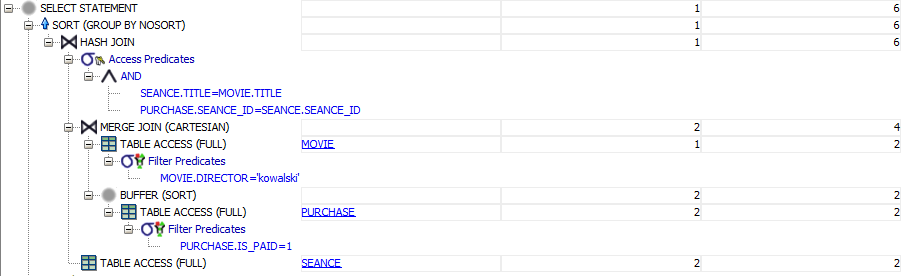
select movie.director, count(\*) as number\_of\_purchases from purchase

join seance on purchase.seance\_id=seance.seance\_id

join movie on seance.title = movie.title

where purchase.is\_paid = 1 and movie.director = 'kowalski'

group by movie.director;



Na umieszczonych planach zapytań powyżej możemy zauważyć, że system bazy danych bardzo często sięga po indeksy, które są związane z kluczami głównymi tabel. Niestety przy małej ilości danych w tabelach wyniki planów o niczym nie świadczą, więc nie można określić przydatności indeksów oraz porównanie planów po dołożeniu indeksów będzie tak samo nie miarodajne jak w przedstawionych przykładach.

W naszym przypadku dobrym przykładem indeksu byłyby indeksy bitmapowe, które stosuje się przy małym zróżnicowaniu wartości danych względem ilości rekordów w tabeli. Indeksy te można założyć w tabeli:

* PURCHASE na kolumnę IS\_PAID (2 wartości tylko),
* PURCHASE na kolumnę TYPE\_P (max 10 wartości),
* MOVIE na kolumnę PLAY (2 wartości tylko),
* PLACE na kolumnę HALL\_ID (stosunek HALL\_ID/PLACE to ok 1/100).

\*Więcej zapytań znajduje się także w procedurach.

1. Wnioski

Projekt rozpoczęliśmy od spisania bytów rzeczywistych, tak aby system wirtualny był ich odzwierciedleniem. Następnie wypisaliśmy ich atrybuty. Przełożyliśmy to na diagram ERD   
i zoptymalizowaliśmy za pomocą „postaci normalnych 1,2,3”.

Mieliśmy problemy na etapie projektowania bazy, aby zoptymalizować bazę (tworzenie się cykli miedzy tabelami).

Następnie zaimplementowaliśmy bazę (czyste relacje z atrybutami). Następnie stworzyliśmy klucze, z którymi mieliśmy małe problemy związane z kierunkami, ale dość szybko sobie z nimi poradziliśmy.

Następnie przebrnęliśmy przez tutoriale PL/SQL i stworzyliśmy procedury. Niektóre z nich spełniają potrzeby aplikacji, która będzie korzystała z tej bazy danych. Problemy w procedurach najczęściej dotyczyły wyboru typów danych i oddawania rezultatów.

Kiedy mieliśmy już gotowe relacje, klucze i kilka procedur zaczęliśmy dodawać procedurami dane do tabel.

Potem stworzyliśmy powyższe zapytania i prześledziliśmy ich plany wykonania, o których wnioskach napisaliśmy pod koniec pkt. 7.

W projekcie nie uwzględnialiśmy procedur usuwających danych, ponieważ założenie jest takie, że będzie ona też stanowiła swojego rodzaju archiwum. Dane będą wprowadzane poprawnie, co będzie kontrolowała aplikacja. Natomiast np. w przypadku filmów o ich aktualności świadczy atrybut „PLAY”.