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.

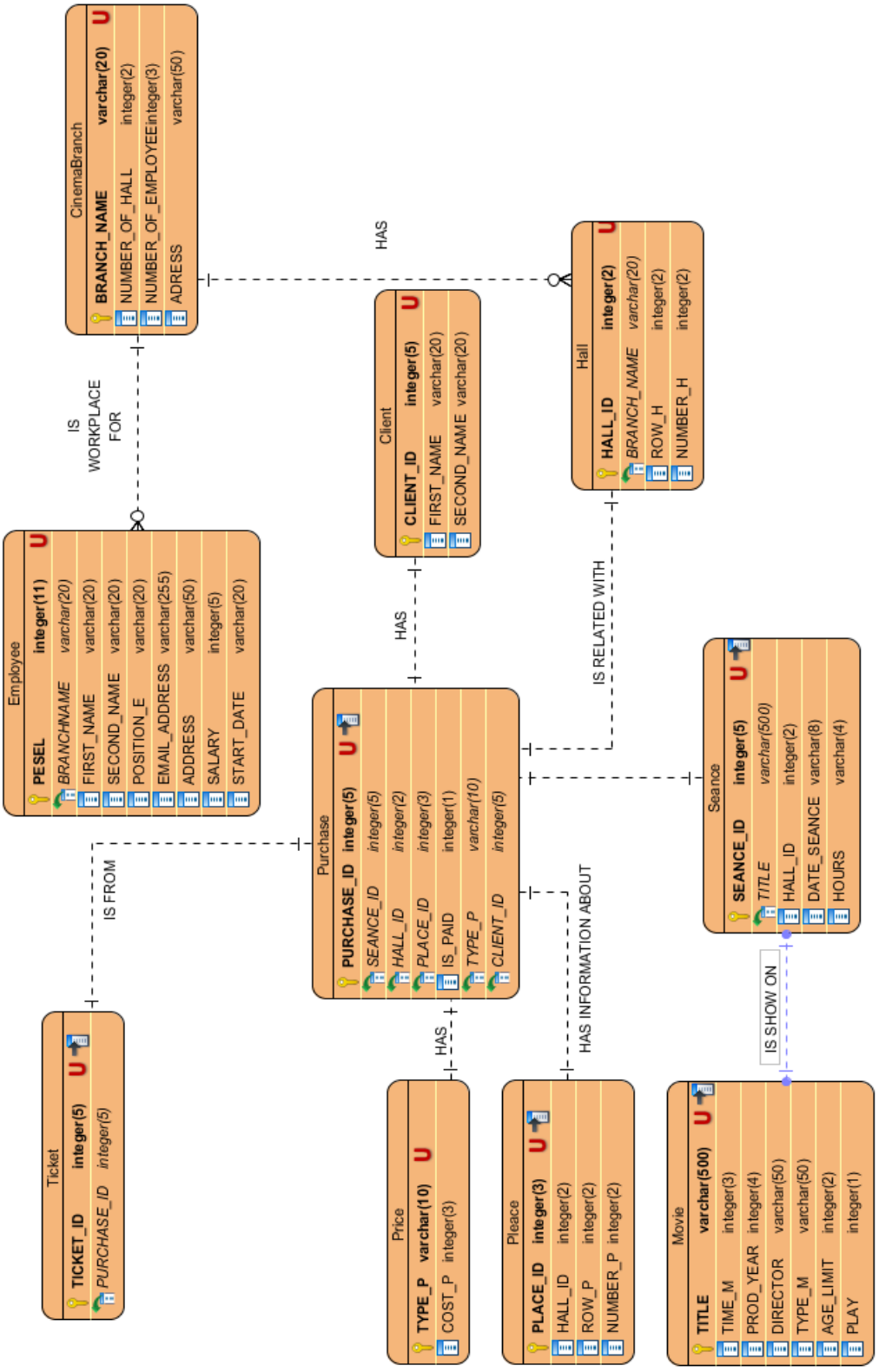
Funkcje obsługiwane przez menadżera:

- 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.

2. Diagram ERD



3. 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

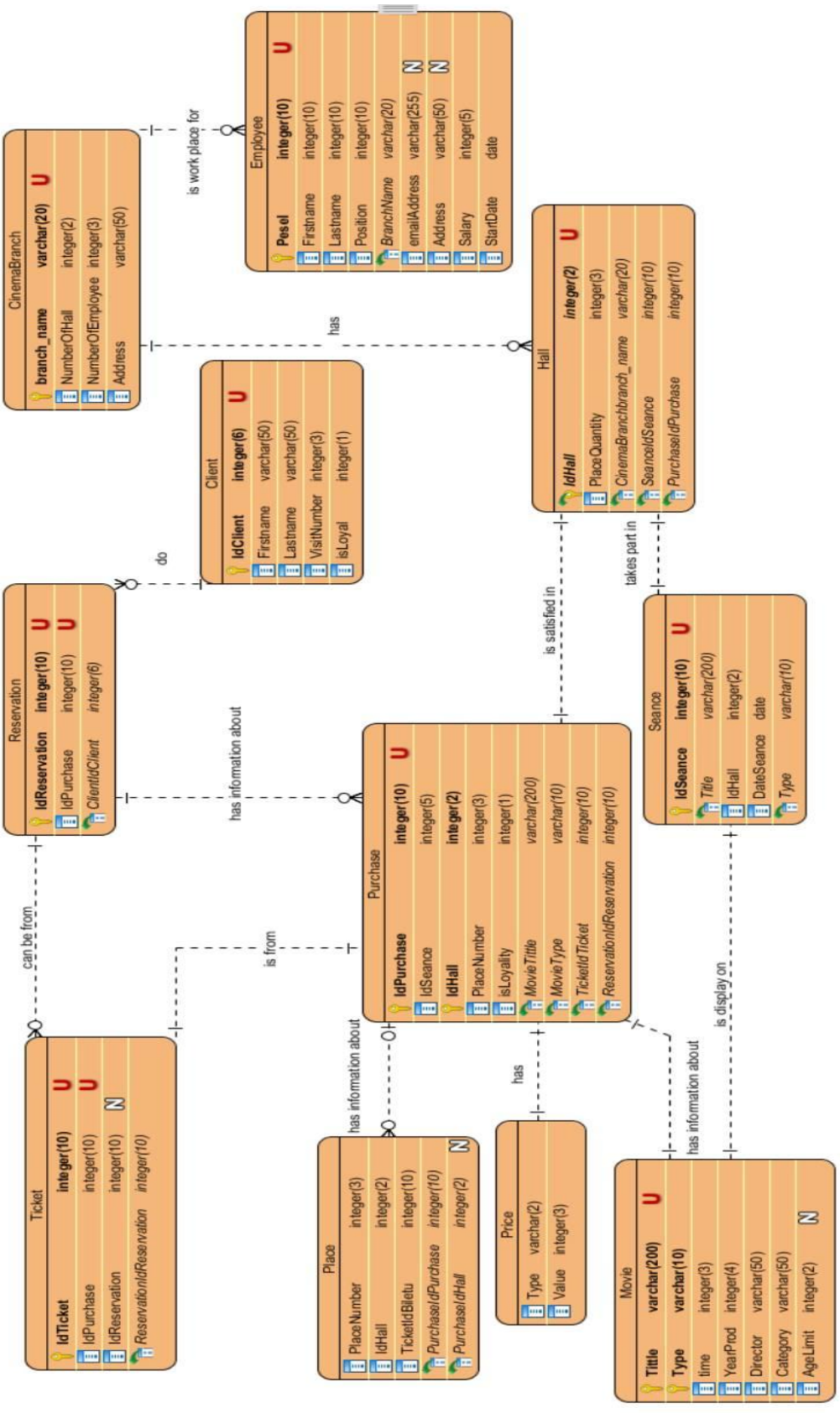
Movie:

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

4. 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.

5. 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
);
```

6. Kod PL/SQL tworzący procedury

1. 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;
```

2. 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;
```

3. 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;
```

4. 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;
```

5. 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;
```

6. 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;
```

7. 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;
```

8. 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;
```

9. 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;
```

10. 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;
```

11. 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;
```

12. 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;
```

13. 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;
```

14. 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;
```

15. 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;
```

16. 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;
```

17. 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;
```

18. 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;
```

19. 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;
```

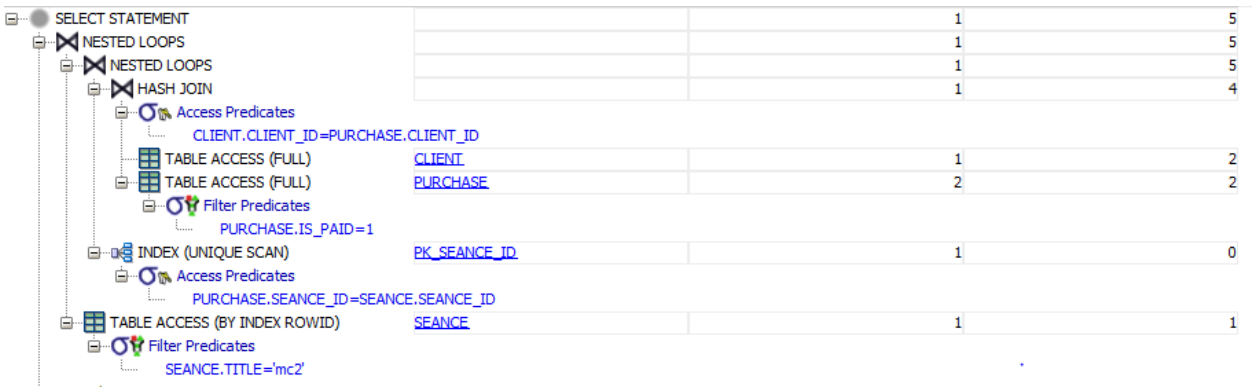
20. Procedura która pokazuje filmy wyświetlane w danej sali w danym dniu. Przyjmuje następujące argumenty:
    - *HALL_IDA – numer sali*
    - *DATEA – data*

```
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;
```

7. Przykładowe zapytania

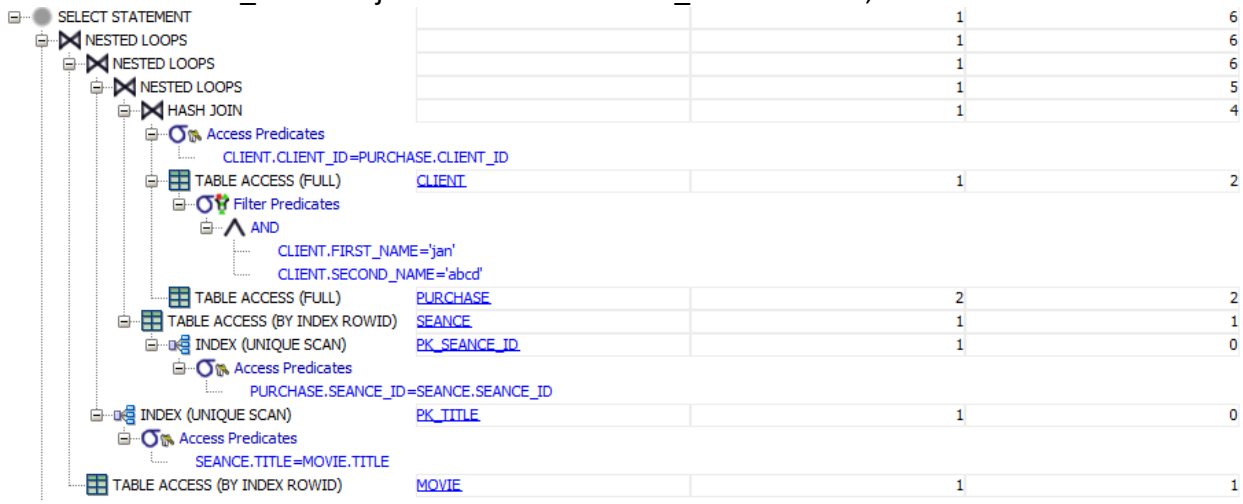
1. Zapytanie wyświetlające imie i nazwisko klientów, którzy opłacili już zamówienie i poszli na film o danym tytule.

```
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';
```



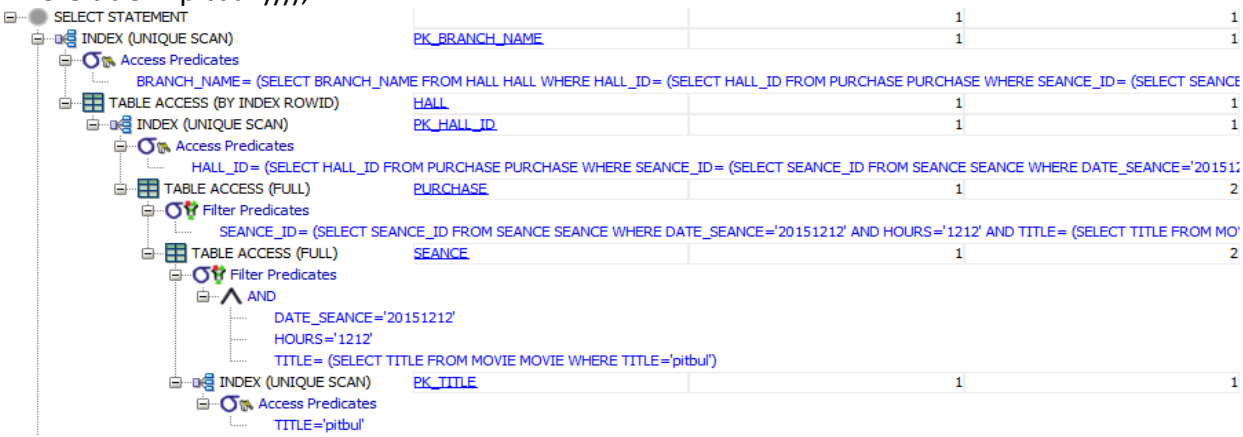
2. Zapytanie, któro wyświetla nam wszystkie tytuły filmów oraz reżysera tego filmu, na którym był klient o danym imieniu i nazwisku.
   Ograniczeniem jest to, że zamówienie (Purchase) musi być wystawione dla klienta.

```
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';
```



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

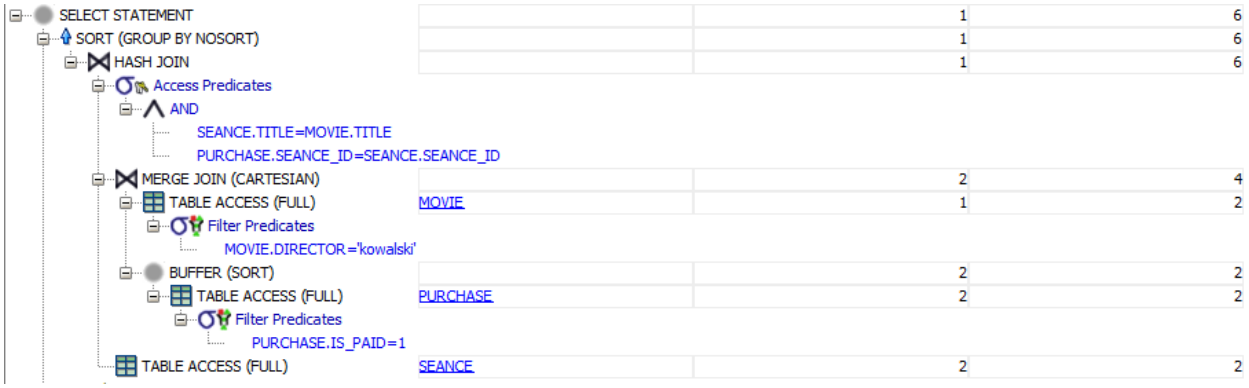
```
select branch_name from CinemaBranch
where branch_name = (select branch_name from hall
where hall_id = (select hall_id from purchase
where seance_id = (select seance_id from seance
where date_seance = '20151212' and hours > '1200' and title = (select title from movie
where title = 'pitbul'))));
```



4. 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.

8. 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”.