Vysoké Učení Technické v Brně
Fakulta Informacních Technológií


# Databázové systémy
# Zadanie č. 39. Reťazec multikín

**Samuel Líška** (xliska20)
**Boris Štrbák** (xstrba05)

22. Apríl 2020

# Úvod

Práca na projekte prebehla bezproblémovo, projekt sme si rozdelili a následne sa navzájom kontrolovali. Na prácu so školskou databázou Oracle sme používali softvare **DataGrip** od spoločnosti Jet Beans. Komunikácia prebiehala hľavne prostredníctvom internetu.

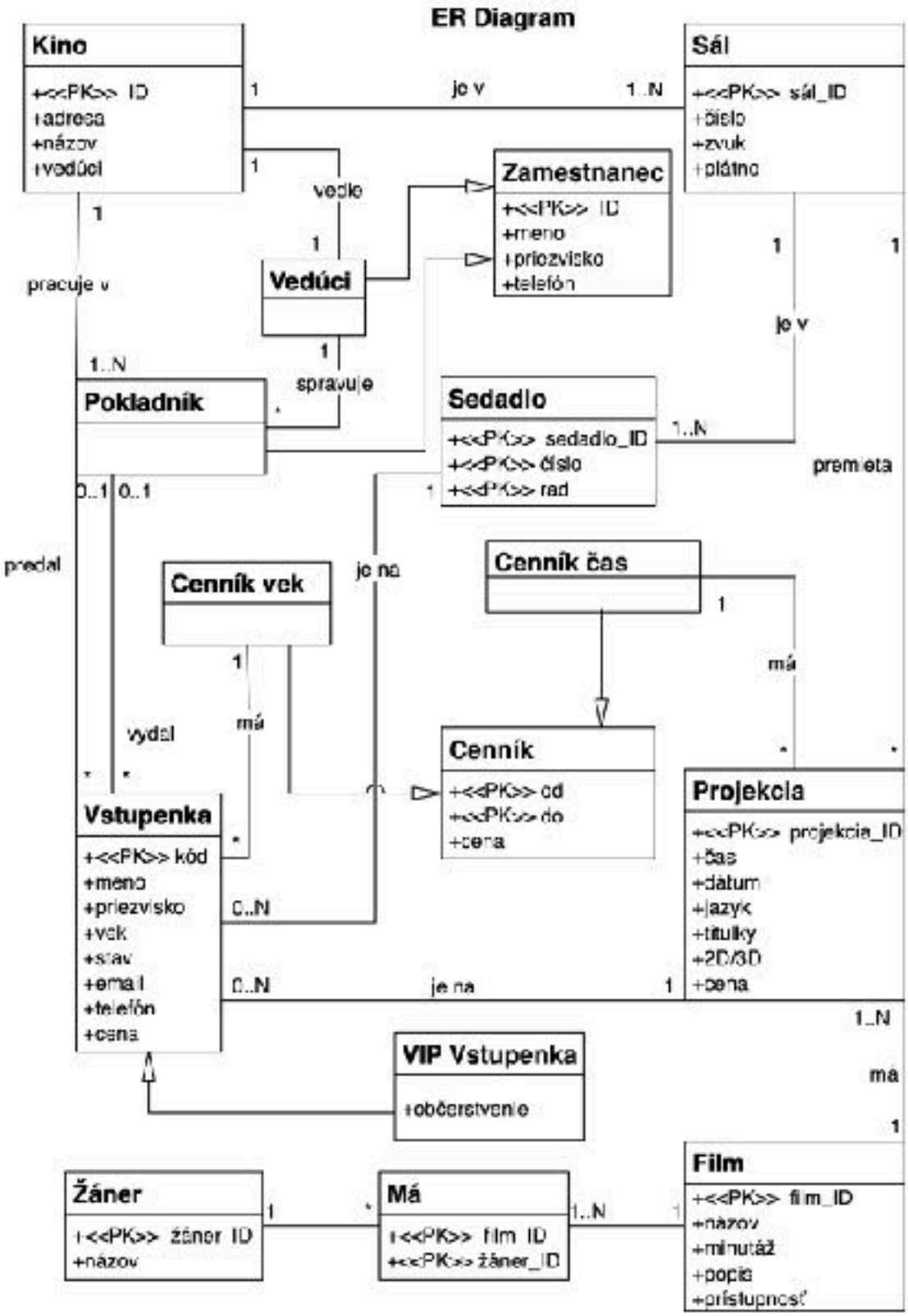
# Zadanie

Vytvořte návrh informačního systému pro řetězec multikin. Řetězec vlastní několik multikin a každé multikino obsahuje několik promítacích sálů. V sálech probíhají projekce filmů. Systém musí umožnit klientům přípojeným přes webové rozhraní vyhledávání projekcí podle názvu filmu, žánru, kina, apod. Systém umožní klientovi zarezervovat si na zvolenou projekci konkrétní sedadla (jeden klient si může zarezervovat i více sedadel)(není třeba uvažovat překročení kapacity sedadel promítacího sálu), či dříve zadanou rezervaci zrušit. Cena za vstupenku se liší podle toho, zda je divák dítě, mládež, dospělý, či důchodce, zda se jedná o dopolední, odpolední, či večerní představení, atd. Pokladní musí mít možnost prodat vstupenku, ať už při koupi na místě, tak i na základě webové rezervace. Není třeba pamatovat si, který prodavač prodal kterou vstupenku. Vedoucí pracovníci jednotlivých poboček mají možnost zjistit tržby jednotlivých multikin, filmů, apod. Zákazníci si mohou přes webové rozhraní prohlížet program kin.

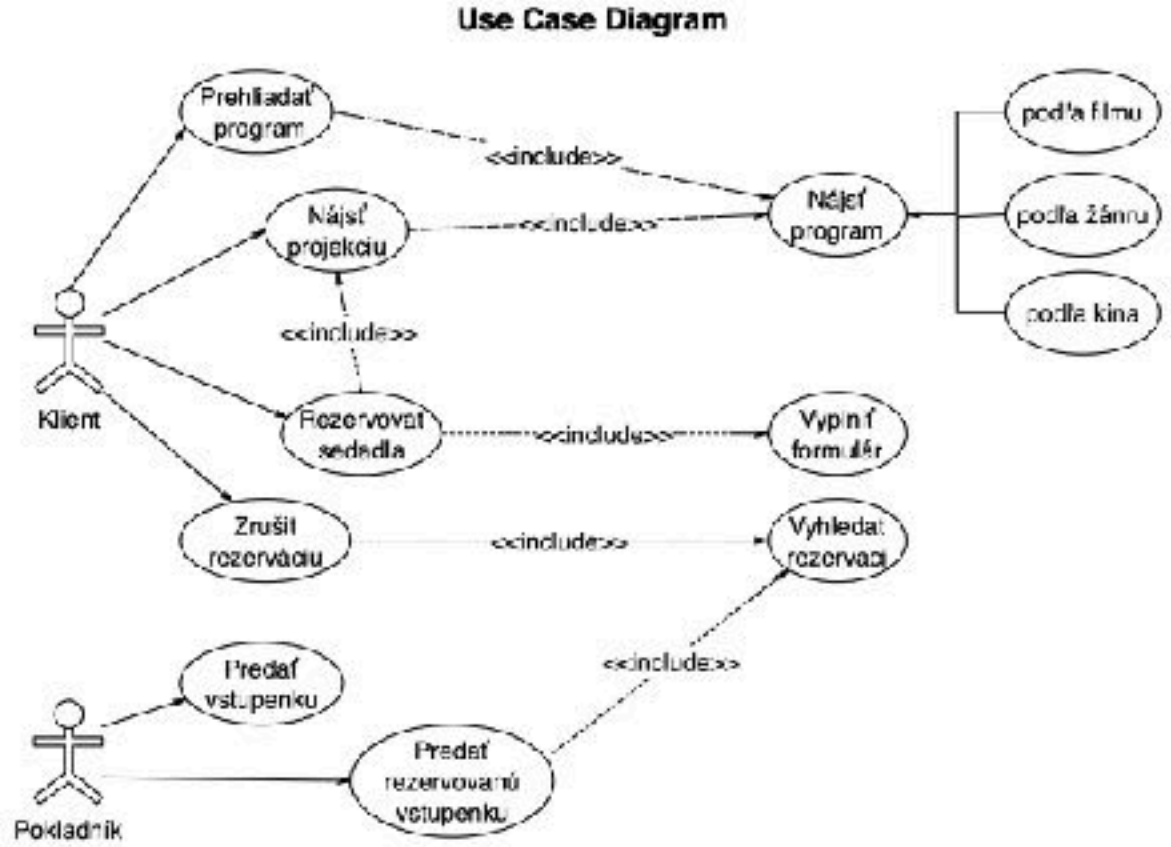
# Časti projektu

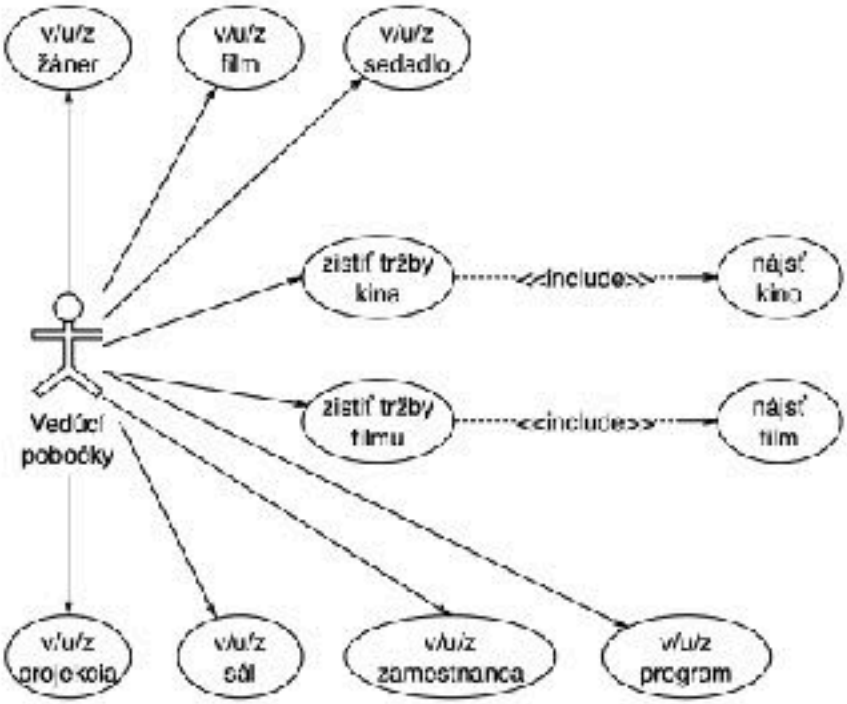
1. *Dátový model (ERD) a model prípadu užitia*
2. *SQL skript pre vytvorenie základných objektov databázi*
3. *SQL skript s niekoľkými SELECT dotazmi*
4. *SQL skript pre vytvorenie pokročilých schématov databázi*
5. *Dokumentácia*

# 1. a) **Dátový model (Entity relationship diagram)**



# b) **Diagram prípadu užitia (Use case diagram)**



# 2. Vytvorenie základných objektov

Bolo potrebné vytvoriť tabuľky na základe ER diagramu. Konkrétne tabuľky: **kino, sal, sedadlo, film, film_ma_zaner, projekcia, zamestnanec, zaner, cennik_vek, cennik_cas** a **vstupenka**. Tabuľky sú poprepájané pomocou *foreign* kľúčov, a tabuľký v ktorých bolo nutné využiť primárne kľúče sme aplikovali pomocou sekvencií. Pri dropovaní tabuliek sme implementovali kaskádové vymazávanie, kôli odkazovaniu sa na ďalšie tabuľky.

Tabuľky sme následne naplnili vzorovými dátami.

# 3. Výber z tabuliek prostredníctvom SELECT

Pri výbere z viacerých tabuliek sme využili SQL príkaz **JOIN** s variáciami **INNER**(prienik) a **LEFT**(Prienik + prvá tabuľka).

## SELECT Príklad

Jednoduchý výber ktorý vyberie počet kúpených vstupeniek na danú projekciu, a vuyžije agregátnu funkcie ***COUNT()*** na spočítanie kúpených vstupeniek.

```
SELECT
        projekcia_ID,
        COUNT(cena) bought
FROM
        vstupenka
GROUP BY
        projekcia_ID;
```

## Zložitejší SELECT

Spojenie prieniku **4** tabuliek a následné aplikovanie 2 agregátnych funkicí na výpočet zárobku na danom žánri. Pomocou **GROUP BY** špecifikujeme agregáciu podľa názvu žánru.

```
SELECT
        z.nazov,
        COUNT(z.nazov),
        SUM(v.cena)
FROM
     film
INNER JOIN film_ma_zaner fmz on film.ID = fmz.film_ID
INNER JOIN zaner z on fmz.zaner_ID = z.ID
INNER JOIN projekcia p on film.ID = p.film_ID
INNER JOIN vstupenka v on p.ID = v.projekcia_ID

GROUP BY
        z.nazov;
```

## SELECT s využitím EXISTS

Výber zamestnanca ktorý kúpil viac ako 2 vstupenky na svoje telefénne číslo uložené v tabuľke zamestnancov. **HAVING** v tomto prípade selectne iba prípady, kedy je počet vybratých riadkov väčší ako 2.

```
SELECT *
FROM zamestnanec z
WHERE
    EXISTS(
            SELECT
                COUNT(*)
            FROM
                vstupenka v
            WHERE
                z.telefon = v.telefon
            GROUP BY
                z.Meno
            HAVING
                COUNT (*) > 2
        );
```

# 4. Pokročilé schématy

## TRIGGER

Implementovali sme 2 triggery, ktorých úlohy sú:

### TRIGGER pre vstupenky

Generovanie unikátneho 4 (a viac) miestného kódu pre identifikáciu vstupeniek.

```
CREATE OR REPLACE TRIGGER kod_num
    BEFORE INSERT ON vstupenka
    FOR EACH ROW
BEGIN
    IF :NEW.kod IS NULL THEN
        :NEW.kod := kod_num.NEXTVAL+1000;
    END IF;
END;
```

### TRIGGER hash

Hashovanie hesla pre zamestnancov, ktorý sú vedúcimi a majú prístup do systému.

```
CREATE OR REPLACE TRIGGER adn
    BEFORE INSERT ON zamestnanec
    FOR EACH ROW
    WHEN(new.veduci = 1)
BEGIN
    IF :NEW.heslo IS NOT NULL THEN
        :NEW.heslo :=
        DBMS_OBFUSCATION_TOOLKIT.MD5(
            input => UTL_I18N.STRING_TO_RAW( DATA :NEW.heslo)
        );
    END IF;
END;
```

## PROCEDÚRY

Procedúra *film_earnings spočíta zárobok daného fílmu, podľa zadaného ID filmu* v parametri. Vo funkcií je implementovaný **KURZOR** na základe ktorého je urobená smyčka ktorá napĺňa premennú *total_cost* priebežnými cenami dokiaľ nenastane prípad:
**EXIT WHEN** cursor_cost%**NOTFOUND**;
Ktorý breakne LOOP.

Procedúra *under18_attendance počíta percentuálnu návštevnosť zadaného filmu(ID) pod 18 rokov.* ***Nevyužíva*** *kurzor. Pri výpočte percent môže nastať delenie nulou(napríklad ak je film 18+), preto je využitý* ***EXCEPTION****:*
**EXCEPTION WHEN** ZERO_DIVIDE **THEN**

## EXPLAIN PLAN A OPTIMALIZÁCIA

Úlohou príkazu ***EXPLAIN PLAN*** je zobrazeniedetailu určitého príkazu. My sme pre príklad použili príkaz ***SELECT***, konkrétne:

```
EXPLAIN PLAN FOR
SELECT Z.meno,Z.priezvisko,count(*) AS count_tickets
FROM zamestnanec Z, vstupenka V
WHERE Z.telefon = V.telefon
GROUP BY Z.meno, Z.priezvisko;
SELECT * FROM TABLE(DBMS_XPLAN.display());
```

ktorý vyberie takých zamestnancov, ktorý si na svoje číslo zakúpili niekedy lístok, a počet zakúpených lístkov.

### Výstup EXPLAIN PLAN:

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |    10 |   360 |     7  (15)| 00:00:01 |
|   1 |  HASH GROUP BY      |             |    10 |   360 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN         |             |    10 |   360 |     6   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| VSTUPENKA   |     7 |    49 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| ZAMESTNANEC |    10 |   290 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

### Oprimalizácia pomocou INDEX-u

Použitie indexovania urýchluje proces vyhľadávanie v tabuľkách, kde je nutné často vyhľadávať, ak ale tabuľku často upravujeme zavedenie *INDEXU* môže čítanie naopak spomaliť. V tomto prípade sa jedna o urýchlenie operácie. Je dôležité aby bol ***index*** vytvorený zo stĺpcov ktoré su priamo použité, inak nám to nijako nepomôže.  My sme konkrétne použili ***index:***

```
CREATE INDEX index_telefon ON vstupenka(telefon);
```

### Výstup EXPLAIN PLAN po zavedení indexu:

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                 |    10 |   360 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY      |                 |    10 |   360 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS      |                 |    10 |   360 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| ZAMESTNANEC     |    10 |   290 |     3   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN | INDEX_ZAMESTNANEC |     1 |     7 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

## PRÍSTUPOVÉ PRÁVA

Prideľuje práva druhému členovy tímu, u nás konkrétne všetky práve t.j. **zápis, čítanie, …**

## MATERIALIZOVANÝ POHĽAD

Materializovaný pohl'ad je databázový objekt, pri ktorom sa výsledok SQL niekam uloží. Dotaz na materializovaný pohl'ad je preto omnoho rýchlejší(pretože data sú už pripravené).

Náš  príklad sa týka klasického výpisu zamestnancov a následného zápisu.

```
DROP MATERIALIZED VIEW zamestnanci;
DROP MATERIALIZED VIEW LOG ON zamestnanec;

CREATE MATERIALIZED VIEW LOG ON zamestnanec WITH PRIMARY KEY,ROWID(mena)INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW zamestnanci
CACHE                       --postupne optimalizuje čítanie z pohľadu
BUILD IMMEDIATE             --naplní hned po vytvorení
REFRESH FAST ON COMMIT      --optimalizuj čítanie
AS SELECT id,meno, priezvisko as zamestnanec_info FROM zamestnanec;

GRANT ALL ON zamestnanci TO xstrba05;

SELECT * from zamestnanci;
INSERT INTO zamestnanec (meno,priezvisko, telefon, veduci, kino_ID) VALUES ('Patrik', 'Kotula', '0918765234', 0, 2);
COMMIT;
SELECT * from zamestnanci;
```